Universidad de Granada

Algoritmos de reducción de datos distribuidos para bases de datos grandes

MEMORIA PRESENTADA POR

Sergio Ramírez Gallego

PARA OPTAR AL GRADO DE DOCTOR EN INFORMÁTICA

Enero de 2018

DIRECTORES

Francisco Herrera Triguero y Salvador García López

Departamento de Ciencias de la Computación
e Inteligencia Artificial

Editor: Universidad de Granada. Tesis Doctorales
Autor: Sergio Ramírez Gallego
ISBN: 978-84-9163-796-7
URI: http://hdl.handle.net/10481/49922

La memoria titulada "*Algoritmos de reducción de datos distribuidos para bases de datos grandes*", que presenta D. Sergio Ramírez Gallego para optar al grado de doctor, ha sido realizada dentro del Programa Oficial de Doctorado en "*Tecnologías de la Información y la Comunicación*", en el Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada bajo la dirección de los doctores D. Francisco Herrera Triguero y D. Salvador García López.

El doctorando y los directores de la tesis garantizamos, al firmar esta tesis doctoral, que el trabajo ha sido realizado por el doctorando bajo la dirección de los directores de la tesis, y hasta donde nuestro conocimiento alcanza, en la realización del trabajo se han respetado los derechos de otros autores a ser citados cuando se han utilizado sus resultados o publicaciones.

Granada, Enero de 2018

El Doctorando　　　　　　　　Los directores

Fdo: Sergio Ramírez Gallego　　Fdo: Francisco Herrera Triguero　　Fdo: Salvador García López

Esta tesis doctoral ha sido desarrollada con la financiación de una beca predoctoral FPU (código FPU13/00047) adscrita al Ministerio de Educación y Ciencia. También ha sido subvencionada por los proyectos TIN2011-28488 y TIN2013-47210-P del Ministerio de Ciencia e Innovación.

Agradecimientos

Si bien es sabido que el trabajo del investigador es individualista por definición, y requiere de un esfuerzo y amor propio que bien se podría calificar de sobrehumano, me gustara dedicar unas líneas de agradecimiento a todas aquellas personas que han colaborado de una manera u otra a hacer realidad esta tesis doctoral.

En primer lugar, esta tesis tiene dedicación especial para mi madre Esperanza, mi tía Isabel, y mi pareja Ana. Ellas, con su ejemplo diario, su inestimable ayuda y su humildad, me han enseñado a luchar, perseverar, y finalmente, llegar a dónde estoy. También quisiera agradecer a mi padre Pedro, mi hermano Mario, y al resto de familiaries y amigos que han hecho posible gracias a su apoyo y paciencia que mis lapsus mentales y despistes recurrentes se transformen en algo importante. A mis abuelos, por su cariño, humildad y carisma. Gracias de nuevo. Esta tesis es por y para vosotros/as.

En el mundo académico, quisiera agradecer a mis directores de tesis, Francisco Herrera y Salvador García, todo el esfuerzo, paciencia y, sobre todo, confianza que han puesto en mí durante todos estos años. El camino no ha sido fácil, pero su ayuda en forma de correos, reuniones y gestiones varias ha sido fundamental para alcanzar este objetivo con excelentes resultados. Quisiera también agradecer el apoyo y la formación que durante todos estos años he recibido de grandes profesionales como José Manuel Benítez, Daniel Peralta, Isaac Triguero, Christoph Bergemir, Alberto Fernández y Manuel Parras, entre muchos otros. Gracias a ellos, no sólo he aprendido la "teoría", sino también mucho conocimiento práctico que seguro me será de gran utilidad en el futuro. Entre otras cosas, a redactar perfectas recetas de uso para el cluster, la claravidencia aplicada a las cosechas, o a llevar una rebeca al trabajo en verano (por si acaso).

Special thanks to Dr. Xiong, Dr. Woźniak and their research groups for the amazing treatment and the priceless hospitality received. I have learned so much from you!

Evidentemente no puede faltar el agradecimiento a mis compañeros doctorandos que han padecido y padecen los gajes del oficio en este país. A Isaac, Joaquín, Victoria, Sara, Pablo, Daniel, Manuel Parras y Titos, Natalia, Lala, José, Antonio, Edu, Fran, Juanan, Rafa, etc. por tantas y tantas horas de charla, de agobios y temores, y también (porqué no) de alegrías. Os deseo lo mejor en estos años, sois ejemplo del valor de esta tierra. También agradecer y mandar mucho ánimo y fuerza a la cantera: Sergio, Maillo, Elena, José Ángel, Diego, Baldán, etc.

Como no, agradecer al zurdo-team: Isa, Jorge, Javier, y algunos diestros: Pedro y Fran, por haber sido protagonistas de esta etapa tan importante, con sus altos y sus bajos. Y para todas aquellas personas que no hayan aparecido en esta página, no os preocupeis, probablemente se deba a uno de mis conocidos episodios de desmemoria.

GRACIAS A TODOS/AS

Table of Contents

Page

I PhD dissertation **1**

1 Introduction 1

Introducción 6

2 Preliminaries 12

2.1 Big Data 12

2.2 Data preprocessing and data reduction 14

2.3 Evolutionary algorithms 17

2.4 Data streaming and concept drift 17

2.5 Efficient nearest neighbor searches for instance selectors: metric tree indexing 20

3 Justification 21

4 Objectives 23

5 Methodology 25

6 Summary 26

6.1 Review on big data technologies, paradigms and strategies 27

6.2 Review on big data preprocessing 27

6.3 Enabling standard discretization on Big Data 28

6.4 Distributed discretization based on evolutionary computation 29

6.5 Review on data reduction for streaming data 30

6.6 Online discretization 31

6.7 Distributed instance selection for high-speed big data streams 31

7 Discussion of results 33

7.1 Review on big data technologies, paradigms and strategies 33

7.2 Review on big data preprocessing 33

7.3 Enabling standard discretizers to deal with Big Data 34

7.4 Distributed discretization based on evolutionary computation 34

7.5 Review on data reduction for streaming data 35

7.6 Online discretization 35
7.7 Distributed instance selection for high-speed big data streams 36
8 Concluding remarks 37
9 Future work 41

II Publications: Published and Submitted Papers **43**
1 Big Data reduction on static databases 43
1.1 Big Data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce 43
1.2 Big data preprocessing: methods and prospects 55
1.3 Multivariate Discretization Based on Evolutionary Cut Points Selection for Classification 78
1.4 Data discretization: taxonomy and big data challenge 93
1.5 A distributed evolutionary multivariate discretizer for Big Data processing on Apache Spark 111
2 Data reduction for streaming data 123
2.1 A survey on data preprocessing for data stream mining: Current status and future directions 123
2.2 Nearest Neighbor Classification for High-Speed Big Data Streams Using Spark 143
2.3 Online Entropy-Based Discretization for Data Streaming Classification 157

Bibliografía **187**

Chapter I

PhD dissertation

1 Introduction

Vast amounts of raw data is surrounding us nowadays, therefore one may say that we are living in the *big data era*. Big Data is usually characterized by the so-called 5V's (Volume, Velocity, Variety, Veracity, and Value), describing its massive volume, dynamic nature, diverse forms, different qualities and usefulness for human beings [MSC13].

Technologies as the World Wide Web, engineering and science applications or business services generates quintillions of bytes daily thanks to the arising of new technologies and services (like Cloud computing) as well as the reduction in hardware price. For instance, Large Hadron Collider experiments in CERN is able to generate 30 petabytes of raw information every year[1]. Due to this quantum leap in data size, organized knowledge can barely be understood or automatically extracted by common analytics tools. This fact has led to the development of so-called data tombs, i.e, large databases that will never be analyzed.

The current volume of data managed by our systems have certainly surpassed the processing capacity of traditional systems, and this applies to data mining as well [WZWD14]. Data mining is a specific task within the *Knowledge Discovery in Databases* (KDD) [HKP11] process responsible of detecting implicit patterns and relationships in data. On the other hand, the KDD process aims at obtaining valid and human-readable knowledge from large databases. It includes several phases commonly performed in the order listed below:

1. Target selection: what is the aim of the discovery? which kind of knowledge we want to extract?
2. Data preprocessing: aiming at cleaning, editing and reducing data in order to obtain quality data for further stages (specially for data mining).
3. Data transformation: maps raw data from one or more formats to an unique and structured format appropriate for a given objective.
4. Data mining: construct knowledge patterns through the analysis of structured data.

[1]`http://home.cern/about/computing`

5. Data visualization: present the obtained results to the end-user in a friendly shape. Patterns are usually represented in tables and/or illustrating plots.

Yet all these phases are interconnected and have a high influence between them, the data mining process is the stage that has received more attention because of its weight in the KDD process. The main purpose of data mining is to discover interesting patterns from problems informally or vaguely defined, or those do not allow a formal and/or efficient solution. Depending on the kind of pattern targeted [Alp10, WFH11, Agg15], we can classify data mining techniques in descriptive methods –discover interesting relationships among data–, and predictive techniques –discover how the model will react to future inputs–.

Additionally, depending on whether the target variable is defined or not, learning methods can be classified into two families:

- **Supervised learning**: the aim is to predict the values of the target variables for new incomes by defining the relation between input variables and the output/target variable. Two families of supervised learners can be devised:
  - *Classification* [DHS00]: the target variables is discrete and their possible values (called labels or classes) are known. For example, sunny, cloudy or rainy in simple weather forecast.
  - *Regression* [CM98] the domain of the target variable is continuous. For example, the foretasted temperature in Fahrenheit scale.
- **Unsupervised learning**: the target variable is undefined. Unsupervised algorithms are devoted to identify descriptive relations implicit in data. They can be categorized into two categories:
  - *Clustering* [Har75] is devoted to create groups of similar instances (intra-cluster distance), and at the same time being as much separate between groups as possible (inter-cluster distance).
  - *Association* [AIS93] consists of identifying interesting relation between variables.

In this thesis we focus on supervised and classification learning. Yet the design of the mining process is decisive for the quality of the decisions and knowledge extracted from the KDD process, all these factors ultimately depend on the quality and suitability of input data. Unfortunately, negative factors such as noise, missing values, inconsistent and superfluous data and huge sizes in examples and features highly influence the learning and discovery processes. It is well-known that low quality data will lead to low quality knowledge [Pyl99]. Thus data preprocessing [GLH15] is a major and essential stage whose main goal is to obtain final data sets which can be considered correct and useful for further data mining algorithms. Despite being less known than other steps like data mining, data preprocessing actually very often involves more effort and time within the entire data analysis process ($> 50\%$ of total effort) [Pyl99] than other stages.

Conversely, data reduction techniques as part of data preprocessing aim at simplifying data and their inherent complexity, while maintaining their original structure. The optimal solution here is a reduced dataset that allows to train a model without performance loss. In many cases, the output model can even be more precise due to the simplified structure obtained (Occam's razor principle). From the feature space side, we can highlight feature selection/generation [BSA15] and discretization [GLS+13] techniques. And from the instance side, Instance Selection (IS) and Instance Generation [GDCH12, TDGH12] methods are the most remarkable families of methods.

Data discretization revolves around the idea of transforming numerical features into discrete ones by creating a finite set of discrete intervals [LHTD02, FFBS17]. Yet real-word data mining tasks often involve numerical attributes, many algorithms can only deal with categorical attributes –e.g., Naïve Bayes [WHW14]–, whereas others would perform better on discrete-valued features [WBM+06]. Because of their simpler representation and its influence in learning models, discretized data is deemed as more advantageous than numerical data. For example, some decision trees are known to generate more compact, shorter, and more accurate results with discrete values [LHTD02]. Furthermore, learning algorithms have shown the effect of improving their speed and accuracy on discrete data.

On the instance side, IS techniques isolate smaller subset of instances in order to improve the learning performance without loss of generalization. In many cases, the subsequent model is not only simpler and more rapid but even more precise due to noise removal. Nevertheless, the selection of relevant instances is not a trivial task since a pairwise comparison between each instance must be performed. This is motivated by the fact that most of instance reduction algorithms rely on the Nearest Neighbor (NN) classifier [CH67] to select or eliminate examples. When IS or instance generation are applied to instance-based learning algorithms, they are commonly denominated as Prototype Selection (PS) and Prototype Generation, respectively [DGH10].

Distributed computing has been widely utilized by data scientists before the advent of Big Data. Many standard and time-consuming algorithms were replaced by their distributed versions in order to speed up their performance. However, for most of current real-world problems, a distributed approach becomes compulsory since no single-node architecture is able to tackle such magnitudes. Also many large-scale processing platforms, paradigms and tools have aroused in last years to give support the problematic of Big Data [FdRL+14]. All these technologies are devoted to bring closer cluster computing power to standard users by hiding the technical nuances derived from distributed environments.

One of the first frameworks in this field was MapReduce (MR) [DG08]. This revolutionary tool was intended to process and generate huge datasets in an automatic and distributed way. By implementing two primitives, Map and Reduce, the user is able to create scalable workflows without worrying about technical nuances, such as: failure recovery, data partitioning or job communication, among others. Whereas the procedure to be included in the Map task is mostly straightforward to determine, the hitch comes when deciding how to carry out the models' fusion within the Reduce task. At this point, the design depends on many factors, namely whether the submodels are different and independent among them, or they have a nexus for being able to join them directly. Two main types of fusion can be found in the current literature depending on how fusion is performed: direct fusion via ensembles, and exact fusion using more sophisticated designs.

Apache Hadoop [Whi12] emerged as the most popular open-source implementation of MR, maintaining the aforementioned features. In spite of its great popularity, MR and Hadoop were not thought to scale well when dealing with iterative and online processes, commonly present in machine learning and stream workflows [Lin12]. Apache Spark [ZCD+12, HKZ+15] was designed as an alternative to Hadoop, capable of performing faster distributed computing by using in-memory primitives. Thanks to its ability of loading data into memory and re-using it repeatedly, this tool was able to overcome some of Hadoop's flaws. Spark is built on top of a novel abstraction model called Resilient Distributed Datasets, a versatile structure that enables easy customization of data persistence and partitioning, among others features.

Data preprocessing is able to adapt data to the requirements posed by each data mining algorithm, enabling its processing that otherwise would be unfeasible. Unfortunately, data reduction methods also suffer from the growth in size and complexity which means that their performance may significantly deteriorate or they may even become inapplicable. Given the previous scenario, novel distributed designs that improve the scalability of standard data reduction techniques has become a major challenge for the data science community.

In many cases we do not only deal with static data collections, but rather with dynamic and unbounded datasets arriving in form of continuous batches of data, known as data streams [Gam10]. In such scenarios, one must be able to constantly update the learning model with new data, to work within time-constraints connected with the speed of arrival of instances, and to deal with memory limitations. To add a further difficulty, many modern data sources generate their outputs with very short intervals, thus creating the issue of high-speed data streams [YXZ+17].

Additionally, data streams may be non-stationary, leading to occurrences of the phenomenon called concept drift [GZB+14], where the statistical characteristics of the incoming data may change over the time. Thus, learning algorithms should take this into consideration and have adaptation skills that allow for online learning from new instances, but also for quick changes of underlying decision mechanisms.

Despite the importance of data reduction, not many proposals in this domain may be found in the literature for online learning from data streams [ZG14]. Most of methods are just incremental algorithms originally designed to manage finite datasets. Direct adaptation of static reduction techniques is not straightforward since most of techniques assume the whole training set is available from the beginning and properties of data do not change over time:

- Most of static *instance selectors* require multiple passes over data, at the same time being mainly based on time-consuming neighbor searches that makes them useless for handling high-speed data streams [GLH15].
- Online supervised *discretization* methods also remain fairly unexplored. Most of standard solutions require several iterations of sharp adjustments before getting a fully operating solution [Web14]. Other major issues, such as interval definition/labeling or how the interaction between learning and discretization components is performed, remain unattended as well.
- On the contrary, *feature selection* techniques are easily adaptable to online scenarios. Yet, they suffer from other problems such as concept evolution, or dynamic [MCG+10] and drifting [BGE15] feature space.

The present thesis addresses two different topics: data reduction on large-scale static databases (1), and data reduction for streaming data (2). Although both topics are tightly connected related by the big data nexus, we have wanted to split the development of Volume and Velocity in two differentiable parts given the long list of particularities of Velocity.

- In the first part, a complete study of the big data reduction methods proposed up to date will be performed in order to draw the current state-of-the-art in terms of scalability power of proposals, frameworks used, open and future problems, etc. We will also analyze the different strategies proposed to fuse and aggregate submodels in distributed frameworks like MR, and we will provide some guidelines to optimize these distributed models at their fullest. Afterwards, we will design a novel evolutionary-based discretization algorithm which introduces a preliminary approach for scalability improvement in the cut points selection problem.

Then, we will re-design this algorithm for distributed environments so that it is able to provide accurate solutions, and at the same time scales properly. Finally, we will propose a novel distributed algorithm inspired by one of the outstanding algorithms in discretization literature.

- In the second part, a deep study on the current state-of-the-art for data streaming reduction will be carried out. On the basis of this knowledge, we will propose a distributed IS method which expedites neighbor searches via an implicit metric-space ordering of the underlying case-base. Finally, we will pose and deal with several questions about how discretization should behave in dynamic environments. Some major issues, such as dynamic interval definition or the interaction discretizer-learner, will be jointly addressed in an unified proposal for online discretization.

After introduction, this thesis is organized as follows: Section 2 provides some background about the main concepts supporting this thesis: Big Data (Section 2.1), data reduction (Section 2.2, evolutionary algorithms (Section 2.3), data streaming (Section 2.4), and nearest neighbor searches (Section 2.5). All of them are paramount to understand the context where the following proposals have been thought.

The reasoning about the importance and justification of this thesis will be given in Section 3. Here the main problems addressed by this thesis are presented. The concrete objectives and the methodology established to provide insights for these problems are described in Section 4 and Section 5, respectively. Section 6 presents an overview of the ideas proposed, whereas Section 7 outlines the most relevant results obtained from these proposals. Particularly, this section shows how the main objectives have been addressed by the proposals. Finally, some overall concluding remarks are given in Section 8 which will serve to define the close horizon for the open future lines of work derived from this thesis (Section 9).

The second part of the document consists of eight international journal publications, organized into two main sections followings the division proposed above:

- Big data reduction on static databases:
  - Big Data: Tutorial and Guidelines on Information and Process Fusion for Analytics Algorithms with MapReduce.
  - Big Data Preprocessing: Methods and Prospects.
  - Multivariate Discretization Based on Evolutionary Cut Points Selection for Classification.
  - Data Discretization: Taxonomy and Big Data Challenge.
  - A Distributed Evolutionary Multivariate Discretizer for Big Data Processing on Apache Spark.
- Data reduction for streaming data
  - A Survey on Data Preprocessing for Data Stream Mining: Current Status and Future Directions.
  - Nearest Neighbor Classification for High-Speed Big Data Streams Using Spark.
  - Online Entropy-Based Discretization for Data Streaming Classification.

Introducción

Grandes cantidades de datos nos rodean hoy día, por lo que se puede decir que estamos viviendo en la era del Big Data. Este fenómeno se caracteriza generalmente por las llamadas 5V's (Volumen, Velocidad, Variedad, Veracidad y Valor), que describen el gran volumen, la naturaleza dinámica, el formato diverso, y las calidades diversas presentes en los datos actuales manejados por el ser humano [MSC13].

Tecnologías como la *World Wide Web*, las nuevas aplicaciones de ingeniería y ciencia, así cómo diversos servicios de negocio generan actualmente millones de *bytes* al día gracias a la aparición de nuevas tecnologías y servicios (como la computación en la nube), y la reducción en el precio del *hardware*. Por ejemplo, los experimentos del Gran Colisionador de Hadrones en el CERN han permitido generar 30 *petabytes* de información bruta todos los años[2]. Debido a este salto cuántico en el tamaño de los datos, el conocimiento estructurado derivado de dichos datos a duras penas puede ser extraído automáticamente por las herramientas estándar de análisis. Este hecho ha llevado al desarrollo de las llamadas tumbas de datos: grandes bases de datos que nunca serán analizadas.

El volumen actual de datos administrados por nuestros sistemas ciertamente ha superado la capacidad de procesamiento de los sistemas tradicionales, y esto tambián se aplica al proceso de **minado de datos** [WZWD14]. La minería de datos es una tarea específica dentro del proceso de **Descubrimiento de Conocimiento en Bases de datos** (en inglés, KDD) [HKP11], responsable de detectar patrones y relaciones implícitas en los datos. Por otro lado, el proceso del KDD tiene como objetivo la obtención de conocimiento válido y legible a partir de grandes bases de datos. Incluye varias fases comúnmente aplicadas en el orden que se detalla a continuación:

1. Selección de objetivos: ¿cuál es el objetivo del descubrimiento? ¿qué tipo de conocimiento queremos extraer?

2. Preprocesamiento de datos: con el objetivo de limpiar, editar y reducir datos para obtener mayor calidad en etapas posteriores (especialmente en la fase de minería).

3. Transformación de datos: asigna datos brutos en uno o más formatos a un formato único y estructurado apropiado para el objetivo de entrada especificado.

4. Minería de datos: construye patrones de conocimiento a través del análisis de datos estructurados.

5. Visualización de datos: presenta los resultados obtenidos al usuario final en un formato comprensible y amigable. Los patrones generalmente se representan en tablas y/o gráficas ilustrativas.

A pesar de que todas estas fases están interconectadas y tienen una gran influencia entre ellas, el proceso de minería de datos es la etapa que ha recibido más atención debido a su peso en el proceso del KDD. El objetivo principal de la minería de datos es descubrir patrones relevantes en problemas vagamente definidos, o aquellos que no permiten una solución formal y/o eficiente. Dependiendo del tipo de patrón fijado [Alp10, WFH11, Agg15], podemos clasificar las técnicas de minería en métodos descriptivos –descubren relaciones interesantes entre datos– y técnicas predictivas –descubren cómo debe reaccionar el modelo ante futuras entradas–.

[2]`http://home.cern/about/computing`

Dependiendo de si la variable objetivo está definada o no, los métodos de aprendizaje pueden ser clasificados en dos familias:

- **Aprendizaje supervisado**: el objetivo es predecir los valores de las variables objetivo para nuevas entradas, definiendo así la relación entre las variables de entrada y la variable destino. Se pueden divisar dos familias de métodos:
  - *Clasificación* [DHS00]: las variables objetivo son discretas y sus valores posibles (llamados etiquetas o clases) son conocidos. Por ejemplo, soleado, nublado o lluvioso en un pronóstico simple del tiempo.
  - *Regresión* [CM98] el dominio de la variable objetivo es continuo. Por ejemplo, la temperatura predicha en escala Fahrenheit.
- **Aprendizaje no supervisado**: la variable objetivo no está definida. Los algoritmos no supervisados se dedican a identificar relaciones descriptivas implícitas en los datos. Se pueden dividir en dos ramas:
  - *Agrupamiento* [Har75] se basa en crear grupos de instancias similares (distancia dentro del clúster), al mismo tiempo, maximizando la separabilidad entre los grupos (distancia entre clústeres).
  - *Asociación* [AIS93] consiste en identificar relaciones interesantes entre variables.

En esta tesis nos enfocamos en el aprendizaje supervisado y la tarea de clasificación. Aunque el diseño del proceso de minería es decisivo para la calidad de las decisiones y del conocimiento extraído del proceso de KDD; todos estos factores dependen en última instancia de la calidad y la idoneidad de los datos de entrada. Desafortunadamente, factores negativos como el ruido, los valores perdidos, los datos inconsistentes y superfluos, así cómo las grandes cantidades de ejemplos y atributos en las bases de datos actuales influyen enormemente en los procesos de aprendizaje y descubrimiento de conocimiento. Es bien sabido que los datos de baja calidad conducen a un conocimiento de baja calidad [Pyl99]. Por lo tanto, el preprocesamiento de datos [GLH15] es una etapa importante y esencial cuyo objetivo principal es obtener un conjunto final de datos que pueda considerarse relevante a la vez que correcto para los fases posteriores del KDD. A pesar de ser menos conocido que otros pasos, el preprocesamiento de datos a menudo implica más esfuerzo y tiempo en todo el proceso de análisis de datos ($> 50\%$ del esfuerzo total) [Pyl99] que otras etapas del KDD.

Las técnicas de reducción de datos, como parte del preprocesamiento de datos, tienen como objetivo simplificar los datos iniciales así como su complejidad inherente, al tiempo que intentan mantener su estructura original. La solución óptima es un conjunto de ejemplos reducido que permita entrenar un modelo preciso sin una pérdida excesiva de rendimiento. En muchos casos, el modelo de salida puede ser incluso más preciso debido a la estructura simplificada resultante del conjunto de entrada (principio de Occam). Desde el punto de vista de los atributos de entrada, podemos destacar las técnicas de Selección/Generación de Atributos [BSA15] y Discretización [GLS$^+$13]. Y en el espacio original, los métodos de Selección de Instancias (en inglés, IS) y Generación de Instancias [GDCH12, TDGH12] son las familias de algoritmos más notables.

La tarea de discretización de datos gira en torno a la idea de transformar atributos numéricos en otros discretos, creando así un conjunto finito de intervalos delimitados [LHTD02, FFBS17]. Aunque la mayoría problemas actuales contienen atributos numéricos, muchos algoritmos están originalmente diseñados para tratar exclusivamente datos categóricos (por ejemplo, el clasificador Naïve Bayes [WHW14]). Otros, aunque no los exigen explícitamente, se verían beneficiados por dichos valores [WBM$^+$06]. Debido a su representación más simple e influencia en los modelos de aprendizaje, los datos discretizados se consideran más ventajosos que los datos numéricos en general. Por ejemplo, se sabe que algunos árboles de decisión son capaces de generar resultados más compactos, simples y precisos gracias a estos datos [LHTD02]. Además, los algoritmos de aprendizaje tienden a mejorar su velocidad y precisión con este tipo de datos.

Poniendo el foco en el espacio original de los datos, las técnicas de IS tienen como objetivo delimitar un subconjunto pequeño de instancias que mejoren el rendimiento en el aprendizaje, sin pérdida de generalización. En muchos casos, el modelo posterior no solo es más simple y rápido, sino que incluso es más preciso debido a la eliminación del ruido. Sin embargo, la selección de instancias relevantes no es una tarea trivial ya que implica una comparación por cada par de ejemplos en el conjunto original. Esto se debe al hecho de que la mayoría de los algoritmos de reducción de instancias se basan en el regla del vecino más cercano (en inglés, NN) [CH67] para seleccionar o eliminar ejemplos. Cuando la IS o la Generación de Instancias se aplica a algoritmos de aprendizaje basados en instancias, se denominan comúnmente Selección de Prototipos (en inglés, PS) y Generación de Prototipos, respectivamente [DGH10].

La computación distribuida ha sido ampliamente utilizada por los científicos de datos antes de la llegada del Big Data. Muchos algoritmos estándar de alto coste computacional fueron reemplazados por versiones distribuidas que aceleraban su rendimiento. Sin embargo, para la mayoría de los problemas actuales del mundo real, un enfoque distribuido se torna obligatorio debido a que las arquitecturas secuenciales actuales son incapaces de abordar tales magnitudes de datos. Diversas plataformas, paradigmas y herramientas de procesamiento a gran escala también han surgido en los últimos años para dar soporte a la problemática del Big Data [FdRL$^+$14]. Todas estas tecnologías tienen como objetivo acercar el poder del cómputo distribuido al usuario estándar, ocultando a su vez los pormeneros técnicos derivados de estos entornos.

Uno de los primeros paradigmas en este campo fue MapReduce (MR) [DG08]. Esta herramienta revolucionaria fue diseñada para procesar y generar grandes conjuntos de datos de forma automática y distribuida. Mediante la implementación de dos primitivas, Map y Reduce, el usuario es capaz de crear flujos de trabajo escalables sin preocuparse por matices técnologicos, tales como: recuperación de fallos, partición de los datos o la comunicación entre los subprocesos, entre otros. Mientras que el procedimiento que se debe incluir en la tarea de mapeo es bastante sencillo de determinar, el problema surge cuando hay que decidir cómo llevar a cabo la fusión de los modelos dentro de la tarea de reducción. En este punto, el diseño depende de muchos factores; a saber, si los submodelos son diferentes e independientes entre ellos, o si tienen un nexo común para poder unirlos directamente. Se pueden encontrar dos tipos principales de fusión en la literatura actual dependiendo del *modus operandi* llevado a cabo por la tarea de reducción: fusión directa via *ensembles* y fusión exacta usando diseños más sofisticados.

Apache Hadoop [Whi12] se erigió como la implementación de código abierto más popular de MR, manteniendo las características antes mencionadas. A pesar de su gran popularidad, MR y Hadoop no fueron originalmente diseñados para escalar correctamente en procesos *on-line* e iterativos, presentes habitualmente en el aprendizaje automático [Lin12]. Apache Spark [ZCD$^+$12, HKZ$^+$15] fue diseñado como una alternativa a Hadoop, capaz de realizar una computación distribuida más rápida mediante el uso de primitivas intensivas en memoria. Gracias a su capacidad de guardar y cargar datos en memoria repetidamente, Spark es capaz de resolver algunos de los defectos presentados por Hadoop y MR. Spark está construido sobre un novedoso modelo de abstracción llamado RDD (en inglés, Resilient Distributed Datasets), una estructura versátil que permite una fácil personalización del particionamiento y la persistencia de los datos, entre otras cosas.

El preprocesamiento de datos permite adaptar los datos originales a los requisitos planteados por cada algoritmo de minería, permitiendo así un procesamiento que de otro modo sería inviable. Desafortunadamente, los métodos de reducción también sufren el crecimiento en complejidad y tamaño de las bases de datos actuales, lo que implica que su rendimiento puede verse deteriorado o incluso puede ser inviable su aplicación. Dada el escenario previo, el desarrollo de nuevos diseños distribuidos que mejoren la escalabilidad de las técnicas estándar de reducción se ha convertido en un reto importante para la comunidad científica.

En muchos casos, no solo se procesan grandes colecciones de datos estáticas, sino que también es necesario procesar conjuntos de datos dinámicos e ilimitados que llegan en forma de constantes lotes de ejemplos, comúnmente conocidos como flujos de datos [Gam10]. En tales escenarios, uno debe ser capaz de actualizar constantemente el modelo de aprendizaje con cada entrada, teniendo en cuenta las limitaciones de tiempo y velocidad de llegada de las instancias, así como las limitaciones de memoria del sistema. Como dificultad extra, muchas fuentes de datos modernas generan sus resultados en intervalos muy cortos, creando así el problema de los flujos de datos de alta velocidad [YXZ$^+$17].

Además, los flujos de datos pueden ser no estacionarios, lo que conlleva la aparición del fenómeno del *concept drift* [GZB$^+$14]. Este fenómeno implica que las características estadísticas de los datos entrantes pueden cambiar a lo largo del tiempo. Los algoritmos de aprendizaje deben disponer de habilidades de adaptación adecuadas que permitan el aprendizaje en línea a partir de nuevas instancias, teniendo en cuenta a su vez los rápidos cambios que se pueden dar en los mecanismos de decisión subyacentes.

A pesar de la importancia de la reducción de datos, pocas propuestas se pueden encontrar en la literatura de flujos de datos sobre este tema en particular [ZG14]. La mayoría de los métodos se basan en procedimientos incrementales diseñados originalmente para administrar conjuntos de datos finitos. La adaptación directa de las técnicas de reducción estática se presenta como compleja, ya que la mayoría de dichas técnicas asumen que todo el conjunto de entrenamiento está disponible desde el inicio y que las propiedades de los datos no cambian con el tiempo:

- La mayoría de los métodos *IS* estáticos requieren múltiples iteraciones sobre los datos, al mismo tiempo que se basan en búsquedas de vecinos costosas que los hacen inservibles para manejar flujos de datos de alta velocidad [GLH15].
- Los temática de la *discretización* online también permanece bastante inexplorada. La mayoría de las soluciones hasta la fecha requieren varias iteraciones de ajustes bruscos en los intervalos antes de generar una solución completamente funcional [Web14]. Otras cuestiones importantes, como la definición de intervalos y su etiquetado, o la interacción entre los componentes del aprendizaje y la discretización, también han sido obviadas.
- Por el contrario, las técnicas de *selección de atributos* pueden ser fácilmente adaptadas al escenario online. Sin embargo, éstas sufren de otros problemas tales como el *concept evolution*, el *dynamic feature space* [MCG$^+$10], o el *drifting feature space* [BGE15].

La presente tesis aborda dos temas: reducción de datos voluminosos (1) y reducción de flujos de datos (2). Aunque ambos temas están estrechamente relacionados entre sí por el nexo del Big Data, hemos querido dividir el desarrollo de Volumen y Velocidad en dos partes diferenciables dada la larga lista de particularidades relacionadas con la Velocidad.

- En la primera parte, se realizará un estudio completo de los métodos de reducción de datos para Big Data propuestos hasta la fecha para así conocer el estado actual en términos de poder de escalabilidad de dichas propuestas, así como de las tecnologías, paradigmas, y problemas actuales y futuros en este campo. También analizaremos las diferentes estrategias propuestas para fusionar y agregar submodelos en *frameworks* distribuidos como MR. También se proporcionarán algunas pautas para optimizar al máximo estos modelos distribuidos. Posteriormente, diseñaremos un novedoso algoritmo basado en computación evolutiva, el cuál introduce un enfoque preliminar para la mejora de la escalabilidad en el problema de selección de puntos de corte. Luego, rediseñaremos este discretizador para entornos distribuidos, de modo que sea capaz de proporcionar soluciones precisas a la vez que escalables. Finalmente, propondremos un novedoso algoritmo distribuido inspirado en uno de los algoritmos más relevantes en la literatura actual.

- En la segunda parte, se llevará a cabo un estudio profundo sobre el estado actual de la tecnología para la reducción de flujos de datos. Sobre la base de este conocimiento, propondremos un método IS distribuido que agilice las búsquedas de vecinos mediante un ordenamiento implícito del espacio de la base de casos. Finalmente, plantearemos y abordaremos varias preguntas acerca del comportamiento deseado de la discretización en entornos dinámicos. Algunos problemas importantes, como la definición de intervalos o la interacción discretizador-clasificador, se abordarán conjuntamente en una propuesta unificada para la discretización online.

Tras la introducción, esta tesis se organiza de la siguiente manera. La Sección 2 aporta información básica sobre los conceptos principales que soportan esta tesis: Big Data (Sección 2.1), reducción de datos (Sección 2.2), algoritmos evolutivos (Sección 2.3), flujos de datos (Sección 2.4), y las búsquedas de vecinos cercanos (Sección 2.5). Todos ellos serán de gran importancia para comprender el contexto en torno al que gira esta tesis.

El razonamiento sobre la importancia y la justificación de esta tesis se dará en la Sección 3. Aquí se presentan los principales problemas abordados en este documento. Los objetivos concretos y la metodología establecida para desarrollar las soluciones a estos problemas se describen en la Sección 4 y en la Sección 5, respectivamente. La Sección 6 presenta una descripción general de las ideas propuestas, mientras que la Sección 7 describe los resultados más relevantes obtenidos. En particular, esta sección muestra cómo los objetivos principales han sido abordados por las propuestas ideadas. Finalmente, algunas observaciones finales se presentan en la Sección 8 con el objetivo de definir el horizonte cercano de las futuras líneas de trabajo derivadas de esta tesis (Sección 9).

La segunda parte del documento consta de ocho publicaciones en revistas internacionales, organizadas en dos secciones principales siguiendo la división propuesta anteriormente:

- Reducción de datos voluminosos:
  - Big Data: Tutorial and Guidelines on Information and Process Fusion for Analytics Algorithms with MapReduce.
  - Big Data Preprocessing: Methods and Prospects.
  - Multivariate Discretization Based on Evolutionary Cut Points Selection for Classification.
  - Data Discretization: Taxonomy and Big Data Challenge.
  - A Distributed Evolutionary Multivariate Discretizer for Big Data Processing on Apache Spark.

- Reducción de flujos de datos:
  - A Survey on Data Preprocessing for Data Stream Mining: Current Status and Future Directions.
  - Nearest Neighbor Classification for High-Speed Big Data Streams Using Spark.
  - Online Entropy-Based Discretization for Data Streaming Classification.

2 Preliminaries

In this section we describe in detail all the major concepts involved in this thesis. First of all, Section 2.1 gives a brief overview of the big data phenomenon as well as outlines the modern paradigms, frameworks and tools devised for this context. Section 2.2 presents the different family branches of data preprocessing methods, paying special attention to data reduction (instance selection, and discretization). Then, a couple of lines are dedicated to the evolutionary algorithms in Section 2.3, used as optimizers in some of our works. Section 2.4 shows the main problems associated to the processing of streaming data as well as its negative impact in data reduction techniques. Finally, Section 2.5 describes how neighbor searches in instance selection methods can be expedited by imposing an implicit ordering to case-bases.

2.1 Big Data

The Internet continues generating quintillions of bytes of data. By 2018, 400 zettabytes of data will be generated according to some reports of Cisco [Cis16]. A solution for the problem of handling large collections of data is therefore becoming increasingly urgent [MSC13]. Exceptional technologies are now required to efficiently collect, maintain, transmit and process large datasets within tolerable time intervals. Extracting relevant information from these collections of data is now one of the most important and complex challenges facing data analytics research, especially since many knowledge extraction algorithms have been rendered obsolete in the face of such vast amounts of data.

Big Data, a term coined to describe the exponential growth and availability of data nowadays, has become a serious pain for classical data analytics. Gartner [Lan01] referred to Big Data in terms of volume, velocity, and variety, that is, the 3Vs, to which a further 2Vs were added, namely, veracity and value. Information thus defined requires a radically new approach to large-scale processing. An under-explored but no less important topic is big dimensionality in Big Data [ZOT14]. This phenomenon, also known as the "curse of big dimensionality", reflects the explosion of features and the combinatorial impact of new large incoming datasets with thousands or even millions of features.

Distributed computing has been widely used by data scientists before the advent of big data phenomenon. Many standard and time-consuming algorithms were replaced by their distributed versions with the aim of speeding up the learning process. However, for most of current massive problems, a distributed approach becomes mandatory nowadays since no single-node architecture is able to cope with them. As a result of the fast evolving of big data environment, a myriad of tools, paradigms and techniques have surged to tackle different use cases in industry and science [FdRL$^+$14]. These platforms bring closer cluster computing to the standard user (engineers and data scientists) by hiding the technical nuances derived from distributed environments.

The MR execution environment [DG08] is the most common paradigm used in the distributed processing scenario. Being a privative tool, its open source counterpart, known as Hadoop, has been traditionally used in academia research [Whi12]. It has been designed to enable distributed computation in a transparent way, also providing fault tolerance, automatic data partition and management, and automatic job-resource scheduling. To benefit from MR, any algorithm must be divided into two main stages: Map and Reduce. The first one is devoted to split the data for further processing, whereas the second collects and aggregates the results.

Additionally, the MR model is defined with respect to an essential data structure: the (key,value) pair. The processed data, the intermediate and final results work in terms of (key,value) pairs. To summarize its procedure, Figure 1 illustrates a typical MR program with its *Map* and *Reduce* steps.

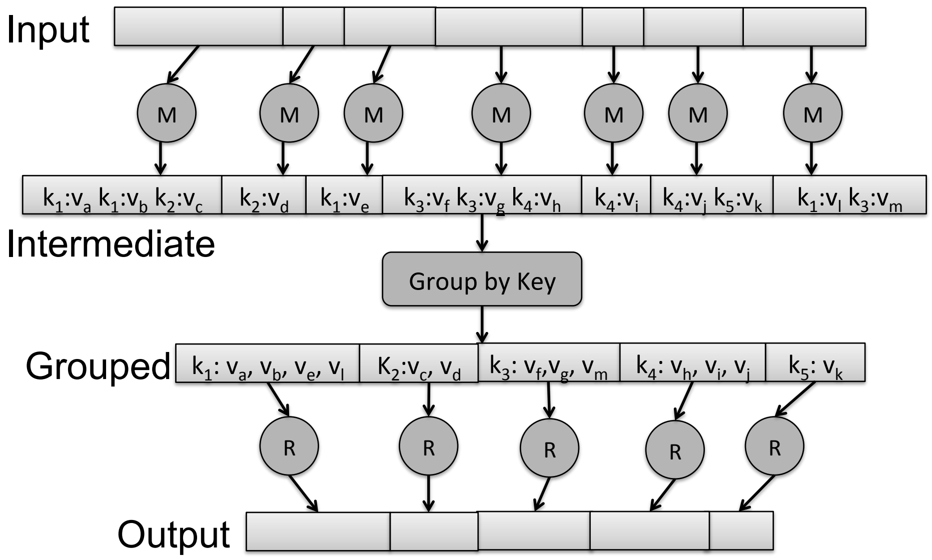


Figure 1: The MapReduce programming model. k elements represent the keys in the pairs, whereas v the values.

The MR scheme can be described as follows.

- Map function first reads data and transforms records into a key-value format. Transformations in this phase may apply any sequence of operations on each record before sending the tuples across the network.
- Output keys are then shuffled and grouped by key value so that coincident keys are put together to form a list of values.
- Finally, the Reducers perform some kind of fusion on the lists to eventually generate a single value for each pair. As a further optimization, the reducer is also used as a combiner on the map output. This improvement reduces the total amount of data sent across the network by combining each item generated in the Map phase into a single pair.

Apache Hadoop and MR, despite being popular tools, have been criticized by its poor performance on certain applications, including online or iterative computing, high inter-process communication procedures or in-memory computing [Lin12].

In recent years, Apache Spark has been included in the Hadoop ecosystem [ZCD+12, HKZ+15] as a powerful framework that performs faster distributed computing on Big Data. Spark relies on in-memory primitives to perform up to 100 times faster than Hadoop for certain applications. The fact that this platform enables user programs to load data into memory and to make repeated queries means that it is particularly useful for online and iterative processing, especially for Machine Learning (ML) algorithms. Spark is also versatile tool that integrates other programming models such as Pregel and MapReduce.

Spark is based on a distributed data structure called resilient distributed datasets (RDDs). RDDs are an immutable, partitioned set of records that can be generated by either stored data or other RDDs. Users can also control other distributed features such as persistence and data partitioning. For instance, users may *cache* a dataset in memory to be reused in further iterations. RDDs can also be persisted on disk if we consider that re-execution may be more costly than spilling to disk. Placement of key-based data can be optimized by choosing between different Spark partitioning schemes (*range* or *hash*) or even your custom partitioner.

Finally, as a Spark's subproject, MLlib [MBY$^+$16] was created to provide native learning and statistical components to this platform. Among its many functionalities includes classification, regression, clustering, collaborative filtering, optimization and dimensionality reduction (mostly feature extraction).

Although several golden standard algorithms for ML tasks have been redesigned to incorporate a distributed implementation for big data technologies, this is not yet the case for preprocessing algorithms. Then, novel scalable designs (sometimes, completely new) of standard reduction algorithms [CS17] are required to extend and maintain these libraries, which generalizes the use of ML in large-scale scenarios.

2.2 Data preprocessing and data reduction

The set of techniques used prior to the application of data mining are named as data preprocessing [GLH15], and are known to be one of the most meaningful issues within the famous KDD process [HKP11]. Since raw data will likely contain imperfect, containing inconsistencies and redundancies in their initial shape, they will not be valid for further data mining process. We must also mention the fast growing of data generation rates and their size in business, industrial, academic and science applications. The huge amounts of data collected nowadays require more sophisticated mechanisms to properly analyze them. Data preprocessing is able to adapt the data to the requirements posed by each data mining algorithm, enabling its processing which would be unfeasible otherwise.

Albeit data preprocessing is a powerful tool that can enable the user to treat and process complex data, it may consume large amounts of processing time [Pyl99]. It includes a wide range of disciplines, as data preparation and data reduction techniques as can be seen in Figure 2. The former includes data transformation, integration, cleaning and normalization; while the latter aims to reduce the complexity of the data by applying feature or instance selection, or data discretization (see Figure 3). After the application of a successful data preprocessing stage, the final data set can be regarded as a reliable and suitable source for any data mining algorithm.

Among the long list of data preprocessing techniques, this thesis is focused on data reduction, concretely, on discretization and IS. The aim of data reduction is to provide a more manageable training set in terms of complexity and size in order to improve accuracy, memory and time performance of the subsequent DM phase. Different families of techniques are part of data reduction, here we highlight most relevant:

- Feature selection (FS): is "the process of identifying and removing as much irrelevant and redundant information as possible" [BSA15]. The goal is to obtain a subset of features from the original problem that still appropriately describe it. This subset is commonly used to train a learner, with added benefits reported in the specialized literature [BCSMAB13, CS14]. FS can remove irrelevant and redundant features which may induce accidental correlations in learning algorithms, diminishing their generalization abilities. The utilization of FS is also known to decrease the risk of over-fitting, as well as to reduce the feature space, thus making the learning process faster and less memory-consuming.

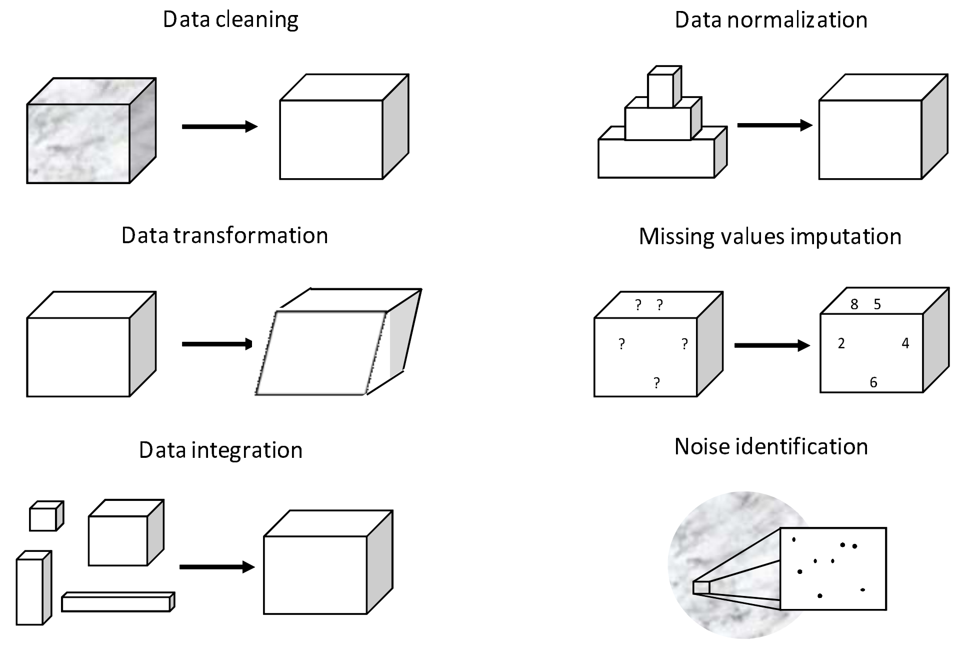


Figure 2: Subcategories in data preprocessing. It includes cleaning, normalization, integration, among others.

- Feature Extraction (FE): is the process of generating new features by transforming the training input space to a new space that better describes the problem [vdMPvdH09]. In FE, original attributes can be removed, maintained or they may serve to create new artificial attributes. Linear and non-linear space transformations or statistical techniques such as principal component analysis [Jol02] or single value decomposition are classical algorithms in this field.

- Instance selection: is comprised by a series of techniques aiming at selecting a subset of data that replaces the original data set, at the same time being able to fulfill the learning goal defined at the start [GDCH12, LGP15]. We must distinguish between instance selection, which implies a smart operation of instance categorization, and data sampling, which constitutes a more randomized approach [GLH15].

- Instance generation (IG): besides selecting data, may generate and replace the original data with new artificial examples [TDGH12]. IG allows us to fill regions of the input domain in case no representative examples exist there, or to condensate large amounts of instances in crowded regions. IG methods are often called prototype generation methods, as the artificial examples created tend to act as a pivotal example in a region or a subset of the original instances.

- Discretization: transforms quantitative data into qualitative data by dividing the numerical features into a limited number of non-overlapped intervals [LHTD02, GLS$^+$13]. Using the boundaries generated, each numerical value is mapped to each interval, thus becoming discrete.

As mentioned before, in this thesis we will focus on discretization and IS for being less explored than other topics like FS. The first one focus on the simplification of the feature space whereas IS

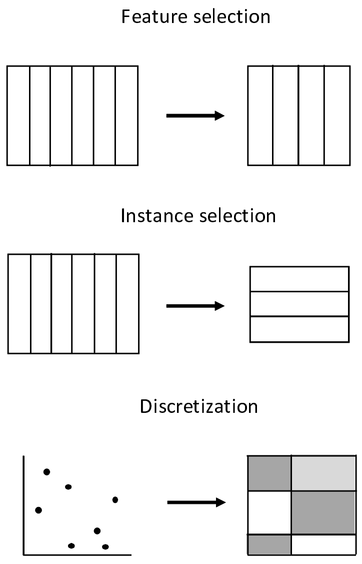


Figure 3: The most relevant data reduction families: feature selection, instance selection and discretization.

focus on the reduction of the instance space. In next lines, we will deepen in the main benefits and features of both artifacts.

Discretization is crucial in the KDD process [LHTD02] as many DM algorithms explicitly impose constraints to the domain and type of input data. Some decision trees, for instance, perform splits based on information or separability measures that require categorical values in most cases. If continuous data is present, the discretization of the numerical features becomes mandatory, either prior to the tree induction or during its building process. This is specially relevant given the high number of real-world applications with continuous attributes. In fact, three of the ten methods considered as the top ten in data mining [WK09] need external or embedded discretization: C4.5 [Qui93], Apriori [AS94] and Naïve Bayes [WHW14].

Discretization also produce extra benefits, such as: data simplification –promoting faster and more precise learning–, and improved readability –features are easier to understand and explain– [LHTD02]. Nevertheless these benefits come at price, any discretization process is expected to generate a subsequent loss in information. Minimizing this loss is the main goal pursued by the discretizer, and a NP-complete problem known as the cut points selection problem. Cut point selection problem can be represented as a binary search problem and optimized using metaheuristics.

Several heuristic models have been proposed to cope with discretization, for example, those based on information entropy [FI93], statistical tests [LS97], likelihoods [Bou04] or rough sets [ZHJ04]. Discretization methods can be categorized according to the nature of its evaluation measure. Other criteria have been used to properly categorize discretizers [GLH15], such as univariate/multivariate, supervised/unsupervised, top-down/bottom-up, global/local, static/dynamic, etc.

Yet discretization reduces complexity of features' spaces, it does not directly apply a straight selection on samples or features as done by IS techniques. However, IS shares with discretization similar computing cost. In IS a pairwise comparison between each instance is required ($O(n^2)$, where n is the training set size). This is due to most of IS methods explicitly or implicitly relies on the k-NN technique to discern between promising and irrelevant examples.

Depending on the type of search implemented, IS methods may be classified into three categories [GDCH12]: 1) condensation –aiming at only retaining boundary points that are close to the borders–; 2) edition –aiming at removing noisy boundary points–; or 3) hybrid methods –combining the two previous approaches by removing both internal and border points–.

2.3 Evolutionary algorithms

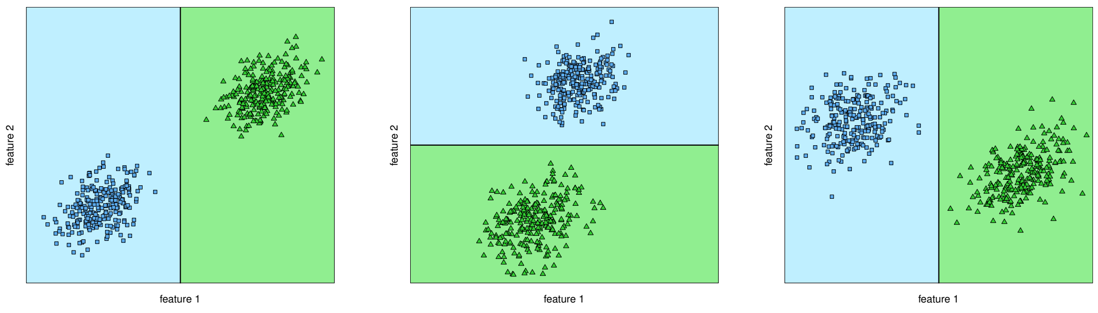
Evolutionary algorithms (EAs), and concretely genetics algorithms (GAs), have proved to perform well in many optimization problems [Gol89]. They have been used specially in engineering, biology and health care [Sim08, NMH$^+$14, CCP13, NZD11]. GAs are search heuristic methods that mimic the process of natural selection, being mainly inspired by evolutionary techniques such as inheritance, mutation, selection and crossover [ES03]. Benefits of EAs lies on the flexibility of the fitness measure as well as the robust learning system behind them.

EAs have been used for data preparation with promising results [Fre02]. As the cut points selection problem (in discretization) can be seen as an optimization problem with binary search space, we resort to the use of GAs to address this problem. For both of our large-scale discretization methods, we have elected CHC as optimizer. CHC [Esh91] is a classical evolutionary model thought for binary coding that tries to get a suitable trade-off between a deep exploration of search space (diversity) and the ability of exploiting the local properties of the search to avoid a premature convergence (exploitation). CHC implements HUX operator for crossover and replaces mutation with a reseeding process to unblock local objectives.

2.4 Data streaming and concept drift

Data stream is a potentially unbounded and ordered sequence of instances that arrive over time [Gab12]. Therefore, it imposes specific constraints on the learning system that cannot be fulfilled by canonical algorithms from this domain. The main differences between static and streaming scenarios are:

- instances are not given beforehand, but become available sequentially (one by one) or in the form of data chunks (block by block) as the stream progresses;
- instances may arrive rapidly and with various time intervals between each other;
- streams are of potentially infinite size, thus it is impossible to store all of incoming data in the memory;
- each instance may be only accessed a limited number of times (in specific cases only once) and then discarded to limit the memory and storage space usage;
- instances must be processed within a limited amount of time to offer real-time responsiveness and avoid data queuing;
- access to true class labels is limited due to high cost of label query for each incoming instance;
- access to the true labels may be delayed as well, in many cases they are available after a long period, i.e., for credit approval could be 2-3 years;
- statistical characteristics of instances arriving from the stream may be subject to changes over time.



(a) Initial distribution. (b) Real concept drift. (c) Virtual concept drift.

Figure 4: Two main types of concept drift with respect to their influence over decision boundaries.

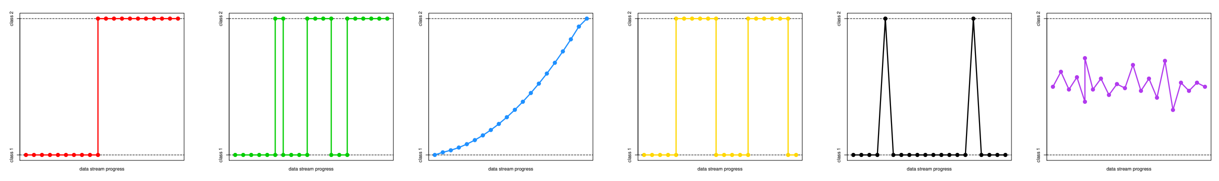
Let us assume that our stream consists of a set of states $\mathbf{S} = \{\mathrm{S}_1, \mathrm{S}_2, \cdots, \mathrm{S}_n\}$, where S_i is generated by a distribution D_i. By a stationary data stream we will consider a sequence of instances characterized by a transition $\mathrm{S}_j \rightarrow \mathrm{S}_{j+1}$, where $\mathrm{D}_j = \mathrm{D}_{j+1}$. However, in most modern real-life problems the nature of data may evolve over time due to various conditions. This phenomenon is known as concept drift [GZB+14, LA11] and may be defined as changes in distributions and definitions of learned concepts over time. Presence of drift can affect the underlying properties of classes that the learning system aims to discover, thus reducing the relevance of used classifier as the change progresses. At some point the deterioration of the quality of used model may be too significant to further consider it as a meaningful component. Therefore, methods for handling drifts in data streams are of crucial importance to this area of research.

There are several aspects to be accounted when analyzing the nature of drift:

- **Influence on the learned classification boundaries**: two types of concept drift are distinguished here. A **real** concept drift affects the decision boundaries (posterior probabilities) and may impact unconditional probability density function, thus poses a threat to the learning system. A **virtual** concept drift does not impact the decision boundaries (posterior probabilities), but affect the conditional probability density functions, thus not influencing the currently used learning models. However, it should still be detected. Visualization of these drift types is presented in Figure 4.

- **Types of change**: three main types of concept drift are outlined here taking into consideration its rapidness ratio. **Sudden** concept drift is characterized by S_j being rapidly replaced by S_{j+1}, where $\mathrm{D}_j \neq \mathrm{D}_{j+1}$. **Gradual** concept drift can be considered as a transition phase where examples in S_{j+1} are generated by a mixture of D_j and D_{j+1} with their varying proportions. **Incremental** concept drift has a much slower ratio of changes, where the difference between D_j and D_{j+1} is not so significant, usually not statistically significant.

  We may also face with so-called **Recurring** concept drift, what means that a concept from k-th previous iteration may reappear $\mathrm{D}_{j+1} = \mathrm{D}_{j-k}$ and it may happen once or periodically. **Blips**, also known as outliers which should be ignored as the change it represents is random [Kun08]. **Noise**, which represents insignificant fluctuations of the concept and should be filtered out [Brz15]. **Mixed** concept drift is a hybrid phenomenon, where more than a single type of concept drift may appear during the stream mining process. One should note that in real-life scenarios types of changes to appear are unknown beforehand and must be determined



(a) Sudden. (b) Gradual. (c) Incremental. (d) Recurring. (e) Blips. (f) Noise.

Figure 5: Six types of drifts with respect to the ratio of changes. Graphs show transitions between the concepts along during the data stream progress.

during the stream processing. Visualization of these types of drifts are presented in Figure 5.

- Unfortunately, in real classification tasks concept drift may appear as a mixture of mentioned above changes.

As mentioned before, managing concept drift is a crucial issue in learning from data streams. Three common solutions are typically used to deal with drift in classification: (a) retrain classification system from scratch every time a new instance or chunk becomes available; (b) detecting changes and retraining classifier only when the degree of changes has been considered as significant enough; and (c) using adaptive learning method that can follow the shifts and drifts in stream on its own. Obviously, the first approach is characterized by an unacceptable computational cost and therefore two remaining solutions remain valid.

There exist four main approaches to efficiently tackling drifting data streams:

- **Concept drift detectors**: are external tools used together with the classification module [GMCR04]. They send signals to the the learning system informing about the severity of current trend; the learning module decides whether the old classifier should be replaced or not.
- **Sliding windows**: assume that we keep a buffer of fixed size containing most recent examples [HSD01]. They are used for the classification purposes and then discarded when new instances become available.
- **Online learners**: are updated instance by instance, thus accommodating changes in stream as soon as they occur [DH00]. Note that some standard classification algorithms may work in online mode, e.g., Neural Networks [LWL$^+$17] or Naïve Bayes [WHW14].
- **Ensemble learners**: are a popular family of methods for data stream mining [WGC14]. Due to their compound structure they can easily accommodate changes in the stream, offering gains in both flexibility and predictive power [KMG$^+$17].

As in large-scale scenarios, standard reduction techniques tend to respond poorly to streaming contexts where a new variable, time, appears on the scene. Direct adaptation of static reduction techniques is not straightforward task since most of techniques assume the entire training set is available from the beginning and properties of data do not change over time.

While it is true that most of filtering FS methods may be easily adapted for dynamic processing (as most are based on cumulative functions), that is not the case for discretization and IS methods.

Instance selection methods helps to maintain a polished base of relevant cases in memory. Yet, case-bases naturally deteriorate and grow in size over time. In data stream scenario, past preserved cases that belong to a previous concept may degrade the performance of the learner if a new concept appears. Likewise, new instances that represent a new concept may be classified as noise and removed by a misbehavior of the IS mechanism, because they disagree with past concepts [LLZdM16].

Some enhancement (**edition**) and maintenance (**condensation**) [GLH15] should be thus performed on case-bases in form of sophisticated IS processes, which select those cases that best represent the current state of the data stream. However, most of current techniques are designed for stationary environments and ignore the concept drift phenomenon.

Discretization algorithms for data stream scenarios must also be able to handle the appearance of concept drifts. Definition and number of discretization intervals may change over time, following shifts in data characteristics. It is then desirable that discretization intervals are able to smoothly adapt to concept drift, without imposing increased computational and memory cost when being recalculated. Otherwise any minor alteration in the meaning and/or the definition of discrete intervals will mean a subsequent drop in prediction power. Degradation in learning is highly related to several factors [Web14]: number of intervals affected, impact of new labels, etc.

Standard labeling schemes used in canonical discretization become outdated in when they face real-world streaming problems. For instance, in cutpoint-based labeling, interval limits (labels) are constantly shifted, and therefore, low information is retained. Likewise, standard interval labeling entails a bunch of major issues to be addressed, such as: abrupt changes in the original meaning of intervals provoked by "label swapping", or steady transfer of instances between intervals (instance relabel).

How interval labels are defined and labeled by discretizers in the streaming context (*interval labeling and definition*), or what type of discrete information is passed to dynamic learners (*interval interaction*) are two open problems whose importance has been neglected by the community. Modern discretization models and schemes that explicitly address these problems are required.

2.5 Efficient nearest neighbor searches for instance selectors: metric tree indexing

As mentioned in Section 2.2, the vast majority of instance selection methods exploit the Nearest Neighbor (NN) rule in one way or another to decide which instances will compose the final selection subset [GLH15]. k-NN [CH67] is an intuitive and effective non-parametric model used in many machine learning problems and can be considered as one of top-ten most influential algorithms in data mining [WK09]. Nevertheless, k-NN is also a costly method that requires to evaluate the distance between each pair of instances (quadratic complexity) to delimit neighborhoods.

Many techniques have been proposed to alleviate the k-NN search complexity. They range from metric trees [Sam05], which index data through a metric-space ordering; to locally sensitive hashing [GIM99], which map (with high probability) those elements near in the space to the same bins. In our instance selection methods, we utilize metric trees (M-tree) in order to impose a metric order to the case-base maintained. M-trees [LMGY04] rely on properties such as the triangle inequality to perform efficient searches on average ($O(log(n))$, thus improving linear search in k-NN. As in other tree-based structures, M-trees are composed of nodes –where pointers to sub-trees are distributed according to the input partitioning computed at the start–, and leaves –where small subsets of data objects reside–.

3 Justification

From previous sections, we may assert that there exist a considerable gap between the storing and the processing capabilities of current systems. We have the right tools but we need proper algorithms that support ML tasks in distributed scenarios. This implies that there exist a pressing need for scalable ML proposals for modern distributed frameworks nowadays, specially in the data preprocessing field. Moreover, the development of data preprocessing in Big Data will also serve to enable ML algorithms where were inapplicable. Beyond voluminous data, velocity is another factor that we must address elegantly, and if possible, in a scalable way.

In order to promote the development of ML, and concretely, data reduction in large-scale static and dynamic environments, the followings major issues should be properly addressed:

- Large-scale data reduction is a quite young field within the more general ML field in Big Data analytics. Few scalable proposals addressing data reduction are available in the literature. Most of them either have been developed for single-node architectures, or have been tested on fairly small datasets. To go further in the design of modern scalable solutions for data reduction we consider that:
    - Firstly, it is important to learn about the current technologies, paradigms and tools used for big data preprocessing, as well as to study the current designs available in the main ML commercial libraries for Big Data. This will give us the right knowledge to decide which platform will support our solution according to our prior requisites, or which distributed strategy is more appropriate for our task.
    - Secondly, it is necessary to go deeper into this topic by analyzing the state-of-the-art of big data reduction in the specialized literature. Study the models developed up to date, categorize them, examine their scalability power, and learn from their pros and cons to create proper further models. It is also relevant to know the main challenges to be tackled on big data reduction for static databases.
    - Data mining, and in particular, data reduction techniques have shown their lack of scaling up capabilities to cope with large-scale problems. Therefore, the study and design of scalable methods will be needed. Firstly, one needs to know which are the most relevant and adequate data reduction methods in the standard literature to be adapted. Once one has shown a preference for one algorithm, the knowledge from previous points will be applied to properly design an equivalent distributed proposal.
- Data reduction for streaming data is another scarcely explored field whose main contributions revolve around FS and IS. Fields like streaming discretization are left unattended and present a bunch of major issues to be tackled (discretizer-learner interaction, interval labeling, etc.). Moreover, high-speed big data streams present in many real-world problems urgently needs modern algorithms that process in parallel data from multiple inputs, and quickly respond to changes.
    - As in the big data field, it is mandatory to perform a throughout study about the active development of data reduction for data streaming. This will allows us to know what are the strengths and weaknesses of current models, which problems remain under-explored or even unexplored, and which subcategory of methods demands more attention, among other issues.

- The previous study will allow us to know the long list of open problems present in online discretization. The development of a binding algorithm tackling all these problems at once would be of great interest for the community.
- Finally, systems fed by high-speed data streams may benefit from the application of IS. However, neighbor searches used in IS must be firstly accelerated in order to comply with time and memory constraints demanded by Velocity in Big Data. The application of some kind of instance-based indexing may boost searches, and make IS an enabling technique for dynamic databases.

All these issues can be encompassed within the subject of this thesis: The development of scalable data reduction models for large-scale static and dynamic databases.

4 Objectives

Once the background have been properly described, we can move to set the aims of this thesis. They include the study and analysis of Big Data, data reduction and data streaming, as well as the development of rapid and effective algorithms for dynamic, and distributed environments (e.g., Apache Spark).

Specifically, the two main objectives behind this thesis are: analysis, design, implementation, and evaluation of scalable data reduction algorithms for (1) large-scale static data and (2) streaming data. In the following list, we will divide each objective into a more manageable set of items that define the milestones to be achieved in this thesis.

- Big data reduction on static databases.
  - **To study the current state on big data technologies, paradigms and strategies**: An analytical study of current big data tools which will allow us to gain further knowledge about the suitability and performance of current platforms and strategies. The end objective is to provide an introduction of the characteristics of these technologies and methodologies, as well as giving some guidelines about the design of novel algorithms for Big Data.
  - **To study the current state-of-the-art on big data preprocessing**: A theoretical and empirical study of the state-of-the-art on big data preprocessing in order to discover the strengths and weaknesses of current developments in the area. Also by studying the scalability limits of current algorithms, we can define performance goals for further developments in the field. To the best of our knowledge, there is no overview in the literature that provides such knowledge.
  - **To prove that standard discretization algorithms can be redesigned to harness the potential of big data**. Concretely, the aim here is to prove that standard discretization methods can be parallelized in Big Data platforms, boosting both performance and accuracy. The resulting algorithm will generate similar (or identical, if possible) intervals to those generated by canonical methods, at the same time showing a scalable behavior in their performing. Moreover, the development of open-source software would prove to be useful for practitioners –by providing an useful tool not available before–, and scientists –by promoting reproducible research–.
  - **To provide a scalable discretizer based on evolutionary computation**. After analyzing current advances on discretization, our objective is to provide more simple and precise remedies to the cut points selection problem by harnessing the potential of evolutionary optimization. The end goal here is to provide a distributed version that proves evolutionary-based algorithms may erect as a feasible alternative in the big data discretization field.
- Data reduction for streaming data.
  - **To study the current state-of-the-art on data preprocessing for streaming data**: A theoretical and empirical study of the state-of-the-art on streaming preprocessing. The aim is to study the features of current methods, and evaluate them both empirically and analytically. We want to discover what are the open challenges in this field in order to propose new models relevant for the community.

- **To provide an scalable solution for online discretization**: Given the long list of open problems in streaming discretization, we focus on providing smart solutions to this topic. The aim here is two-folded: (1) to provide a formal definition for the open problems discovered in the previous study, and (2) to design a compounding solution based on adaptive discretization that encompasses all the aforementioned problems.
- **To incorporate scalable instance selection to high-speed big data scenarios**. The application of IS to high-speed big data streams may erect as a good trend to mitigate the accumulation of noisy instances in multi-input unbounded systems. The IS method developed will leverage metric trees to expedite neighbor searches. Thanks to this achievement, IS will be able to act in large-scale dynamic scenarios for the first time.

5 Methodology

This thesis requires the application of a methodology that is both theoretical and practical. Therefore, we need a strategy that, while maintaining the guidelines of the traditional scientific method, is able to provide the special needs of such methodology. In particular, the following guidelines about the research work and the experiments will be applied:

1. **Observation**: thorough study of the large-scale and dynamic ML problem along with the application of scalable data reduction as a fundamental step in KDD, as well as the possibilities offered by distributed computing technologies to give a proper solution to this problem.

2. **Hypothesis formulation**: design of new preprocessing algorithms (concretely, data reduction methods) that make use of distributed strategies to notably reduce volume in databases. The methods developed must comply with the objectives previously mentioned in order to properly face the big data problem.

3. **Observation gathering**: retrieving the results obtained by the application of the new methods on real-world, voluminous and dynamic databases. Both efficiency and accuracy have to be measured and considered in the designs.

4. **Contrasting the hypothesis**: comparison of the results obtained by the learning algorithms after reduction in order to analyze quality in the new proposals. For that purpose we will rely on scalable ML libraries, such as MLlib or Mahout. Other proposals in the literature will serve to validate effectiveness and efficiency in our model.

5. **Hypothesis proof or refusal**: acceptance or rejection and modification, if proceed, of the developed techniques as a consequence of the experiments performed and the subsequent results. If necessary, the previous steps should be redone to create new hypothesis to be proven.

6. **Scientific thesis or theory**: extraction, redaction and acceptance of the conclusions obtained throughout the research process. All the proposals and results gathered along the entire process should be synthesized into journal publications, and eventually, in this thesis.

6 Summary

This section summarizes the ideas described in the publications associated to this thesis. After that, Section 7 will present a brief overview of the associated results. The research developed for this thesis, and the associated results obtained are collected into the published journal publications listed below:

- S. Ramírez-Gallego, A. Fernández, S. García, M. Chen, F. Herrera, Big Data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce. Information Fusion 42 (2018) 51–61, doi: 10.1016/j.inffus.2017.10.001
- S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez and F. Herrera, Big data preprocessing: methods and prospects. Big Data Analytics 1 (2016) 1–9, doi: 10.1186/s41044-016-0014-0
- S. Ramírez-Gallego, S. García, J. M. Benítez and F. Herrera, Multivariate discretization based on evolutionary cut points selection for classification. IEEE Transactions on Cybernetics 46 (3) (2016) 595–608, doi: 10.1109/TCYB.2015.2410143
- S. Ramírez-Gallego, S. García, H. Mouriño Talín, D. Martínez-Rego, V. Bolón-Canedo, A. Alonso-Betanzos, J. M. Benítez, F. Herrera, Data discretization: taxonomy and big data challenge. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 6 (1) (2016) 5–21, doi: 10.1002/widm.1173I.
- S. Ramírez-Gallego, S. García, J. M. Benítez and F. Herrera, A distributed evolutionary multivariate discretizer for Big Data processing on Apache Spark. Swarm and Evolutionary Computation 38 (2018) 240–250, doi: 10.1016/j.swevo.2017.08.005.
- S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak and F. Herrera, A survey on data preprocessing for data stream mining: current status and future directions. Neurocomputing 239 (2017) 39–57, doi: 10.1016/j.neucom.2017.01.078.
- S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, J. M. Benítez and F. Herrera, Nearest neighbor classification for high-speed Big Data streams using Spark. IEEE Transactions on Systems, Man, and Cybernetics: Systems 47 (10) (2017) 2727–2739, doi: 10.1109/TSMC.2017.2700889.
- S. Ramírez-Gallego, S. García, and F. Herrera, Online Entropy-Based Discretization for Data Streaming Classification. Submitted to Future Generation Computer Systems.

The remainder of this section is organized following the objective-driven scheme presented in Section 4. First, Section 6.1 provides some insights about the review performed on big data artifacts. Second, Section 6.2 analyzes the review performed on large-scale static data reduction. Then, Section 6.3 details the first scalable proposal proposed to deal with the big data discretization problem, whereas Section 6.4 provides an evolutionary-based alternative to the same problem. Regarding the streaming context, Section 6.5 analyzes the thorough study performed on data reduction for data streams. Our first contribution to this topic, concretely to online discretization, is briefly described in Section 6.6. Finally, we conclude the chapter by detailing our distributed IS design for high-speed data streams.

6.1 Review on big data technologies, paradigms and strategies

The prevailing exploitation of big data analytics tools has proven its worth on both industry and academia. The MR programming framework can be stressed as the main paradigm related with such tools. By means of two simple functions, Map and Reduce, any implementation can be automatically parallelized in a transparent way for the programmer, also supporting by default the aforementioned fault-tolerant scheme. Whereas the procedure to be included in the Map task is, most times, straightforward to determine, the hitch comes when deciding how to carry out the models' fusion within the Reduce task. At this point, the design depends on many factors, namely whether the submodels are different and independent among them, or they have a nexus for being able to join them directly.

In this work, we analyzed different paradigms and strategies implemented by modern scalable learning algorithms. Their behavior was also analyzed into detail. The analysis mainly focused on the level of discrepancy between the distributed models and their corresponding single-machine versions. We considered it as the most remarkable aspect to categorize the existing solutions for large-scale ML.

From this perspective, we identified two unlike groups: (1) approximate fusion of models (one submodel per partition, eventually fused), and (2) exact fusion for scalable models (compounding model with the same output as the sequential version). Other relevant aspects considered in this model categorization were their scope (local vs. global), iteration nature (1-step vs. multistage), or possible guidance by a master thread (guided vs. unguided).

A practical study on scalability was performed on each fusion model for the sake of contrasting the variance of time and accuracy performance as resources increase. In order to provide a better understanding of each type of implementation, we also presented some case studies regarding some well-established algorithms from two well-known libraries, such as Mahout (from Apache Hadoop) and MLlib (from Apache Spark).

The publication associated to this part is:

- S. Ramírez-Gallego, A. Fernández, S. García, M. Chen, F. Herrera, Big Data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce. Information Fusion 42 (2018) 51–61, doi: 10.1016/j.inffus.2017.10.001

6.2 Review on big data preprocessing

Novel distributed designs that improve the scalability of standard data reduction techniques has become a major challenge for the data science community. To respond to this challenge more than 30 different algorithms have been proposed so far in the specialized literature, showing the positive impact of large-scale learning and the demand for these algorithms present nowadays.

However, these methods are implemented in different platforms, following different design strategies and patterns inherited. Despite this, many of the published methods show duplicity partly or entirely in their algorithmic descriptions. Moreover, yet many methods have been tagged with the Big Data term, neither they have been tested on really large databases, nor directly follow a distributed approach. As all these problems can lead to confusion, we have proposed a complete overview about this topic. At the time of writing this thesis there is no general categorization and review of large-scale preprocessing methods in the literature.

The definition, characteristics, and categorization of data preprocessing approaches in Big Data have been introduced here. The connection between Big Data and data preprocessing throughout all families of methods and big data technologies are also examined, including a full review of the state-of-the-art. Additionally, research challenges are discussed with special remark on developments on different big data framework, such as Hadoop, Spark and Flink, as well as the encouragement in devoting substantial research efforts in some families of data preprocessing methods and applications on new big data learning paradigms.

We have identified some basic features to categorize large-scale reduction algorithms, including: the supporting framework (Hadoop MR, MPI, Apache Spark, Twister, etc.), the maximum empirical dataset size, number of features and their shape (dense or sparse format), and number of instances addressed by each method. This thorough analysis allowed us to properly categorize the methods, as well as to establish an illustrative ranking about their scalability power.

The journal paper attached to this part is:

- S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez and F. Herrera, Big data preprocessing: methods and prospects. Big Data Analytics 1 (2016) 1–9, doi: 10.1186/s41044-016-0014-0

6.3 Enabling standard discretization on Big Data

Data preprocessing aims at adapting incoming data in raw shape to the strict requirements posed by each learning algorithm. However, canonical data reduction methods are not expected to scale well when managing huge data –both in number of features and instances– so that its application can be undermined or even become impracticable. Despite the importance of data preprocessing in KDD, few proposals have been thought to reduce voluminous data. Specially scarce is the impact reached by large-scale discretization.

In order to fill this gap, in this part we proposed a distributed re-design of the entropy minimization discretizer proposed by Fayyad and Irani in [FI93] using Apache Spark. Our main objective was to prove that well-known discretization algorithms such as Fayyad's discretizer can be parallelized in these frameworks, providing reliable solutions for big data analytics tools.

Several problems arouse during the adaptation of the algorithm. The first issue revolved around the distribution of complexity burden, mainly determined by two time-consuming operations:

- the sorting of candidate points, whose operation exhibits a $O(|A|log(|A|))$ complexity where A represents the set of candidate points (originally composed by all distinct points).
- the evaluation of candidates, which conveys a $O(|B_A|^2)$ operation where B_A means the set of points close to class borders.

Note that the previous complexity is bounded to a single attribute. To avoid repeating the previous process on all attributes, we designed the algorithm to sort and evaluate all points in a single step. Only when the number of boundary points in an attribute is higher than the maximum per partition, computation by feature is compulsory. Our algorithm was denoted by Distributed Minimum Distance Length Principle (DMDLP).

Finally, to demonstrate the effectiveness of DMDLP, a complete experimental evaluation was launched. This framework was composed by two large datasets with up to 60 millions of instances, two distributed classifiers from MLlib, and another discretizer.

The journal contribution associated to this part is:

- S. Ramírez-Gallego, S. García, H. Mouriño Talín, D. Martínez-Rego, V. Bolón-Canedo, A. Alonso-Betanzos, J. M. Benítez, F. Herrera, Data discretization: taxonomy and big data challenge. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 6 (1) (2016) 5–21, doi: 10.1002/widm.1173I.

6.4 Distributed discretization based on evolutionary computation

EAs have been used for data preparation with promising results [Fre02]. However, few evolutionary approaches can be found in the discretization literature. They encompass a wide range of techniques and combinations, such as: GA+clustering [HTH06], GA+rough sets [CLQW03], or GA+Naïve-based wrapper [FILn07]. Given the outstanding results generated by EAs in other fields, a further study in discretization can be quite positive for the literature.

Then we proposed the reformulation of the cut points selection problem as an optimization problem with binary search-space, and the subsequent application of a EA-based optimizer. The EA proposed relies on a wrapper fitness function based on a tradeoff between the classification error provided by the application of two classifiers, and the number of boundary points produced. Another advancement introduced in our method is the multivariate ability to leverage from the existing interactions and dependencies among input attributes and the class. Our proposal is denoted by Evolutionary Multivariate Discretizer (EMD).

EMD have been contrasted with the top-7 in discretization according to [GLH15]. The empirical study consists of 45 datasets, 4 classifiers for comparison, and an analysis based on nonparametric statistical testing.

The next target was to design an scalable approach on Spark able to deal with thousands of cut points, while keeping the original idea behind EMD. Three main problems were faced in this extension:

- In the cut points selection problem, discretizers are mainly affected by the number of boundary points to evaluate (long chromosomes). In particular, this problem is influenced by two factors: the number of instances and features present in the problem. Another hidden factor that influences the complexity is the number of distinct points present in each feature. If this value is high, the algorithm will process a high number of boundary points.

  In order to keep the multivariate philosophy and to alleviate the complexity derived from these two problems, our distributed proposal introduced some major changes, such as to divide the complete set of features into partitions so that point evaluation is performed in a parallel way.

- For the second problem (high number of instances), we proposed to partition the set of instances into a set of equal-sized partitions. Each data partition will serve to evaluate different parts of the chromosome. Once data partitions have been evaluated following the standard scheme, the subsequent partial solutions are aggregated through a voting scheme.

- Regarding the evaluation, one classifier was removed from the fitness function. Naïve Bayes was elected to evaluate fitness because of its simplicity and efficiency in its close-form expression (linear order).

The distributed approach, called Distributed Evolutionary Multivariate Discretizer (DEMD), was tested in a thorough experimental evaluation with several large-scale datasets (up to $O(10^7)$

instances and $O(10^4)$ features). Experiments on real-world datasets have shown the simplicity and precision of outcomes.

The journal papers associated to this part are:

- S. Ramírez-Gallego, S. García, J. M. Benítez and F. Herrera, Multivariate discretization based on evolutionary cut points selection for classification. IEEE Transactions on Cybernetics 46 (3) (2016) 595–608, doi: 10.1109/TCYB.2015.2410143
- S. Ramírez-Gallego, S. García, J. M. Benítez and F. Herrera, A distributed evolutionary multivariate discretizer for Big Data processing on Apache Spark. Swarm and Evolutionary Computation 38 (2018) 240–250, doi: 10.1016/j.swevo.2017.08.005.

6.5 Review on data reduction for streaming data

With the advent of Big Data comes not only an increment in data volume, but also the notion of its velocity. Velocity means data will expand itself over time and new samples will arrive continuously in form of streams. These imposes specific constraints on the learning system that cannot be fulfilled by canonical algorithms, so that new specific models for all areas contained in the KDD process are required.

In spite of the long list of benefits guaranteed by data reduction, not many proposals in this domain can be found in the literature for online learning from data streams [ZG14]. Most of methods are just incremental algorithms originally designed to manage finite datasets. Direct adaptation of static reduction techniques is not straightforward since most of techniques assume the whole training set is available at the start, and properties of data do not change over time:

- Most of static instance selectors require multiple passes over data [GLH15], at the same time being mainly based on time-consuming neighbor searches that makes them useless for handling high-speed data streams.
- Online supervised discretization methods also remain fairly unexplored. Most of standard solutions require several iterations of sharp adjustments before getting a fully operating solution [Web14]. Other major issues, such as interval definition/labeling or the interaction between the learning and discretization components also remain unattended.
- On the contrary, feature selection techniques are easily adaptable to online scenarios. Yet, they suffer from other problems such as concept evolution, or dynamic [MCG$^+$10] and drifting [BGE15] feature space.

This analysis encouraged us to perform a deep study in this area, which includes a thorough enumeration, classification, and analysis of existing contributions for data stream preprocessing. Although there exist previous studies that have performed a coarse-grained analysis on some tasks individually –feature selection and instance selection– [BCnAB15, LLZdM16], this work is the first complex overview of advances in this field, additionally outlining vital future challenges that need to be addressed to ensure meaningful progress and development of novel methods.

In addition to discussing the literature in preprocessing methods for mining data streams, we performed a conscious experimental study to further enrich the work. We have analyzed predictive, reduction, time and memory performance of selected most relevant algorithms in this field. Additionally, nonparametric statistical tests were used to give support to the final conclusions. The discussed experimental framework involved a total of 20 datasets and 10 reduction methods: three feature selectors, three discretizers, and four instance selectors.

The journal publication associated to this section is:

- S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak and F. Herrera, A survey on data preprocessing for data stream mining: Current status and future directions. Neurocomputing 239 (2017) 39–57, doi: 10.1016/j.neucom.2017.01.078.

6.6 Online discretization

Data streams demand novel learning schemes that not only adapt well, but also that constantly revise their time and memory requirements. Nevertheless, up to date few supervised approaches for online discretization have been presented in the literature. Another requirement to face is the likely non-stationary of incoming data (concept drift). Sudden or abrupt changes in data distribution require outstanding adaptation abilities to follow drifting movements in decision borders. Also adjustments should not imply complex rebuilding processes, but should be solved rapidly.

Finally, how interval labels are defined and labeled by online discretizers, or what kind of discrete information is passed to learners are other unsolved problems that have received even less attention in the literature. Recent proposals on online discretization confirm that canonical label indexing is unable to cope with all these flaws, thus showing a deficient behavior in the subsequent learning phase.

In this section we have tackled the previous problems by developing a modern discretization solution that smoothly and efficiently adapts to incoming drifts. Our method, called Local Online Fusion Discretizer (LOFD), mainly relies on highly-informative class statistics to generate accurate intervals at every step. Furthermore, local nature of operations implemented in LOFD offers low response times, thereby making it suitable for high-speed streaming systems.

On the other hand, we have designed two alternatives to effectively improve the underlying interaction between the discretization and the learning phases. The first approach naturally provides class information to learners, whereas the second one (called smooth shift) is a renovated version of the standard scheme.

In order to measure the performance of our method, we have evaluated it on a conscious experimental framework, composed by 12 streaming datasets, 2 online learning algorithms, the current state-of-the-art for online discretization, and a set of non-parametric and Bayesian statistical tests. Plus a study concerning the impact of the proposed alternatives for interaction was applied.

The journal paper associated to this section is:

- S. Ramírez-Gallego, S. García, and F. Herrera, Online Entropy-Based Discretization for Data Streaming Classification. Submitted to Future Generation Computer Systems.

6.7 Distributed instance selection for high-speed big data streams

Lazy learning is considered as one of the simplest, yet most effective schemes in supervised learning. As generalization in these learners is deferred until the test phase, they tend to have much slower classification phase than other learners. Lazy learners (e.g., k-NN) also tend to accumulate instances from data streams, thus leading to using data related to the outdated concepts may for the decision-making process. As of these reasons lazy learning has not been widely used in streaming environments in spite of its attractive properties.

Data reduction techniques may be applied to improve the performance of lazy learners. Concretely, instance selection techniques can be very effective as they reduce the total number of samples stored in the case-base, and therefore, they simplify the underlying search space. Yet to the best of our knowledge, there is no instance selection method up to date able to deal with the phenomenon of high-speed big data streams.

In order to fill this gap, we have proposed an efficient nearest neighbor solution to classify high-speed and massive data streams using Apache Spark. Our algorithm, called DS-RNGE, consists of a distributed case-base and an IS method that enhances its performance and effectiveness. A distributed metric tree has also been integrated with the aim of imposing an order to the preserved case-base, so that further neighbor searches are boosted. Finally, performance is further improved thanks to a distributed edition-based IS technique whose objective is to select correct examples and remove the noisy ones.

Several experiments were performed to asses the time performance of different components contained in DS-RNGE: the distributed tree, the selection, and the lazy learner. Likewise, the precision gradient obtained after each selection phase is measured and analyzed here. 6 large-scale datasets were used to evaluate the performance and precision of our entire system.

The research work associated to this part is:

- S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, J. M. Benítez and F. Herrera, Nearest Neighbor Classification for High-Speed Big Data Streams Using Spark. IEEE Transactions on Systems, Man, and Cybernetics: Systems 47 (10) (2017) 2727–2739, doi: 10.1109/TSMC.2017.2700889.

7 Discussion of results

The following subsections are devoted to summarize and provides further discussion regarding the results obtained in each stage of this thesis.

7.1 Review on big data technologies, paradigms and strategies

This review has been devoted to the the design of scalable ML algorithms following the MR programming model and similar where the process fusion is the core in the design. A thorough taxonomy of distributed models for ML based on both the fusion tactic and the model scope has been analyzed, distinguishing between two main categories: **approximate fusion**, and **exact fusion** of models.

To obtain well-founded conclusions about these different types of methodologies, we have carried out an experimental study to contrast the scalability of the different schemes. Our results have determined the higher quality of those algorithms based on exact fusion of models. In addition, we have observed the best option to a 2x partition rate for Spark-based Machine Learning implementations.

The strong and weak points for both types of the fusion models have been also analyzed. This have allowed us to provide several guidelines on this topic:

- The main point is to favor the development and adoption of global and exact model over the use of approximate fusion as they are more precise and more consistent with theory behind standard methods.
- There is a necessity in developing more theoretical studies to facilitate the migration of current ML models towards Big Data. This way, a direct connection between a certain learning methodology and its distributed design can be established.
- Approximate models may still offer many interesting opportunities for research, specially when no reliable exact alternative is feasible and speed is a must. In particular, the case of ensemble learning is of extreme importance in this scenario.
- Finally, a smart utilization on the distributed operators beyond MR is mandatory in order to implement robust and scalable solutions for the fusion process from a practical point of view. In this respect, Apache Spark stands as a outstanding option in the market as it natively supports several paradigms and operators in its engine.

7.2 Review on big data preprocessing

Different families in big data preprocessing (IS, FS, discretization, imbalanced preprocessing, incomplete treatment, etc.) have been thoroughly studied in this review throughout the categorization and the analysis of the few techniques proposed so far in the literature. The main features and achievements showed by them have been measured and captured in our work.

The results of our comparison have highlighted some negative aspects to be considered in the further development of big data preprocessing methods. We highlight here some:

- The vast majority of methods studied have been implemented on platforms (e.g., Apache Hadoop) that show clear deficiencies when dealing with iterative process such as those present in preprocessing methods. Only 5 methods are built on top of distributed frameworks based on in-memory processing (e.g., Apache Spark or Twister).

- The size of datasets tested are on average far from reality. Most of real-world problems exceeds hundreds of gigabytes of memory, whilst only 6 methods in our study are able to cope with such magnitudes.

- In fact, more than 50% of methods studied either do not specify the size of the problems, or process datasets expanding less than 1 GB on disk.

In the future, significant challenges and topics must be addressed by the industry and academia, especially those related to the use of new platforms such as Apache Spark and Flink, the enhancement of scaling capabilities of existing techniques (e.g., IS or missing values imputation) and the approach to new big data learning paradigms (e.g., unsupervised learning and semi-supervised learning, or data streaming).

7.3 Enabling standard discretizers to deal with Big Data

In the corresponding section, we have proposed a completely distributed method for data discretization which is inspired by Fayyad's discretizer. The aim of this work is to prove that canonical discretizers can be parallelized in big data platforms, boosting both performance and accuracy.

According to the experiments, our solution, called DMDLP, is able to perform 270 times faster than the sequential version in our cluster, improving the accuracy of baseline Gaussian estimation in all the datasets used. The latter fact highlights the positive impact of proper discretization in both small and large-scale learning. This means that not only the learning performance is boosted thanks to discretization but also the precision of derived schemes. This is notable the case study of ECBDL14, where the learning would be likely impractical without the contribution of DMDLP as the dataset is too large for common single-node architectures.

Another important feature to remark from DMDLP is that is able to produce identical discretization schemes to those generated by the original proposal. We thus commit the previous statement of promoting exact models in contrast to approximate-based ones. We also rely on an in-memory computing platform and its complex operators to implement our model, again following the guidelines appointed by the previous survey.

7.4 Distributed discretization based on evolutionary computation

In this section, we have presented a new evolutionary-based discretization algorithm called EMD, which selects the most adequate combination of boundary cut points to create discrete intervals. The proposed algorithm follows a multivariate approach, being able to take advantage of the existing interactions and dependencies among the set of input attributes and the class output to improve the discretization process. It also includes a chromosome reduction mechanism to tackle larger problems.

The large experimental study performed allows us to show that EMD is a suitable method for discretization in small and large problems. It requires a lower number of cut points than the other discretizers, thus producing much simpler discretization solutions. Additionally, EMD outperforms the state-of-the-art discretizers on classification accuracy. It can be considered as the best choice for classifiers, such as C4.5, Naive Bayes, PART, or PUBLIC. Positive outcomes were confirmed by a non-parametric statistical test with high confidence.

In a further extension, called DEMD, a distributed multivariate discretization algorithm based on evolutionary optimization was proposed for Apache Spark. In this remodeled design, a new system of cross-evaluation between partitions of instances and cut points was introduced to evaluate cut points. Despite its approximate nature, the previous evaluation showed promising discretization schemes supported by precise splits, and a quite similar behavior to the inspiring method EMD.

The experimental framework applied on several big datasets (up to $O(10^7)$ instances and $O(10^4)$ features) have shown the improvement on accuracy and simplicity when relying on DEMD. Moreover, DEMD enables to tune the simplicity/accuracy rate of the generated solutions regarding our available computing power.

7.5 Review on data reduction for streaming data

Here we have presented a thorough survey of data reduction methods applied on data streams. We have focused on analyzing basic related concepts, existing works, and present and future challenges. Based on a number of relevant characteristics, we have proposed a simple, yet useful taxonomy of current developments in dynamic preprocessing.

Starting with FS, we have divided the methods according to the type of selection operator implemented –filter, wrapper, or hybrid–, the type of feature space conversion between different steps –Lossy-F, Lossy-L, Lossless–, or whether new classes can emerge from streams –concept evaluation–. In case of IS methods, we provide a taxonomy based on: the type of selection measure –case accuracy, instance weighting, competence-based, and many more–, whether they include drift detection, and whether they perform edition and/or condensation as treatment. Finally, discretization categorization only depends on a single factor: the splitting/merging strategy implemented by the methods –only merge, only split, both–.

Most relevant methods have also been analyzed empirically through a conscious experimental framework, which includes a long and diverse list of artificial and real datasets with different types of drift. A statistical analysis based on non-parametric tests have been conveyed to support the resulting conclusions. The previous results have shown the wide range of phenomenons affecting features in data stream mining, as well as the lack of methods capable of dealing with such shortcomings. Furthermore, the results witness the unfavorable performance of instance selectors regarding time and memory requirements, thus preventing their meaningful impact on high-speed data stream mining.

According to these evidences, it can be claimed that data preprocessing for data streams is still in its early days. Great progress has been made in instance and feature selection, but other tasks like discretization remains yet to be properly addressed.

7.6 Online discretization

This section was thought to gain even further knowledge from the major issues to be faced by contemporary online discretizers. As a potential solution for the interval labeling and interaction problems, we have proposed and analyzed two opposing strategies. Both alternatives have shown in the experiments their positive effect on the transition between consecutive discretization states, and the most advantageous scenarios for them.

In order to solve the adaptation problem we have integrated all the labeling schemes in a novel online discretization algorithm. LOFD generates self-adaptive and highly-informative discretization schemes, in which precise intervals are supported by updated class statistics. LOFD presents a high level of responsiveness as well due to the fully local strategy implemented (only a reduced subset of intervals is considered in each split/merge phase).

The experimental framework has proven that LOFD is by far the most competitive solution in terms of predictive accuracy. Compared with other options, which either barely cover the search space or generate too many meaningless intervals, LOFD is able to achieve an excellent trade-off between simple and precise solutions. LOFD is also ranked as one of the most rapid supervised discretizers.

7.7 Distributed instance selection for high-speed big data streams

In this part of the thesis, we have presented DS-RNGE, a nearest neighbor classification solution for processing massive and high-speed data streams using Apache Spark. Up to our knowledge, DS-RNGE is the first lazy learning solution designed for high-speed streaming scenarios. DS-RNGE includes an IS echnique that constantly improves the performance and effectiveness of the built-in learner by only allowing the insertion of correct examples, and the subsequent removal of outdated samples.

The experimental analysis, involving several datasets with millions of instances, shows that DS-RNGE combines high accuracy with significantly reduced processing time and memory consumption. DS-RNGE without removal of outdated cases showed more precise results than those held by the baseline model in all cases. DS-RNGE also performed faster in the prediction phase, whereas the baseline model performed faster in updating the case-base. In general, both algorithms present similar performance ratios in terms of overall runtime.

8 Concluding remarks

In this thesis, we have addressed several problems focusing on a common objective: the analysis, design, implementation and final evaluation of algorithmic solutions for the the problem of data reduction in large-scale databases. This thesis revolves around two main research lines: (1) big data reduction on static databases, and (2) data reduction for data streaming.

In the first research line, our primary objective was to gain full understanding of the current technologies, paradigms, and tools present in the big data environment, as well as the algorithmic designs already present in these platforms. The results derived from this study encouraged us to shift towards exact theoretically-founded models instead of approximate and ad-hoc solutions. To gain even further foundation, we carried out a second study about the current state-of-the-art in the specialized literature on big data reduction. We found that few reduction methods were adapted to deal with large-scale static data, and most of them either are built on poor designs, inadequate platforms, or have overestimated their scalability power.

Bearing this knowledge in mind, we proposed an scalable adaptation for a golden discretization algorithm in the literature which guarantees identical results to the original inspiring method. This means that the theoretical assumptions supporting the latter method has also served as a basis for the new model. This algorithm, called DMDLP, showed that canonical algorithms may actually develop well in distributed environments, and at the same time, be handy for the KDD process. The next challenge in this field was to leverage from the outstanding capacity of EAs to create a competitive discretizer whose scope was not only bounded to the single-thread scenario. The first commitment was achieved with excellency by EMD. This empirically outperformed the current state-of-the-art in discretization thanks to its evolutionary nature and the multivariate approach implemented. The second goal was eventually achieved with a extension work that originated DEMD, a distributed multivariate discretizer based on evolutionary optimization. Despite emulating its predecessor through an approximate procedure, DEMD provided competitive discretization schemes for large-scale real-world problems. Empirical outcomes acknowledge the superiority in simplicity and precision of DEMD's schemes with respect to those generated by its closest competitor (DMDLP).

The second part of this dissertation is dedicated to reduction in data streaming. Once again, we started by studying the current state-of-the-art, as well as by providing a survey that analyzes, categorizes and evaluates the last developments in this area. We also analyzed the close-future goals to be tackled in streaming reduction. The outcomes derived from here were uneven: some fields, such as FS, proved to be quite advanced, while others, such as online discretization, were in the early days and with many present issues to be addressed (even discovered). Yet IS for data streaming is a fairly well-established field, the results in our study witness the poor adequacy of current proposals to cope with high-speed big data streams in terms of time performance.

Two promising research branches and commitments arise from the previous analysis. Firstly, the necessity of efficient and effective evolving schemes in discretization, as well as the long list of open problems to be faced in this field. We arranged both concerns in LOFD, a self-adaptive discretizer which leverages from updated class statistics to smoothly shift interval limits. As a potential solution for the interval labeling and interaction problems, we proposed and integrated in LOFD two opposing and functional strategies. The experiments confirmed LOFD is by far the most competitive solution in the field nowadays. The second branch lies in the requirement for rapid IS methods able to process several high-speed data streams in parallel. The answer to this demand was DS-RNGE, the first known proposal in this area. DS-RNGE applies IS to each streaming iteration so that a sanitized and downsized case-base is maintained over time. The subsequent output is an accurate and scalable system capable of processing thousands of instances per second in modest cluster of machines.

Conclusiones

En esta tesis hemos abordado varios problemas centrados en un objetivo común: el análisis, diseño, implementación y evaluación final de soluciones algorítmicas para el problema de la reducción de datos en bases de datos de gran escala. Esta tesis gira en torno a dos líneas principales de investigación: la reducción de (1) datos voluminosos y (2) flujos de datos.

En la primera línea de investigación, nuestro principal objetivo fue estudiar en profundidad las tecnologías, paradigmas y herramientas presentes en entorno de datos grandes, así como los diseños algorítmicos ya presentes en estas plataformas. Los resultados derivados de este estudio nos animaron a promover desarrollos exactos con una consistente base teórica, en contraposición a soluciones aproximadas y ad-hoc. Para obtener un mayor fundamento teórico, realizamos un segundo estudio sobre el estado actual de la literatura especializada. Descubrimos que pocos métodos de reducción fueron adaptados para manejar datos estáticos a gran escala, y que la mayoría de ellos están construidos sobre diseños deficientes, plataformas inadecuadas, o que éstos han sobreestimado su capacidad real de escalabilidad.

Teniendo en cuenta este conocimiento, propusimos una adaptación distribuida para un algoritmo de discretización de gran relevancia en la literatura. Dicha aproximación garantiza esquemas idénticos a los generados por el método original. Esto implica que los fundamentos teóricos que sustentan el método original también han servido de base para el nuevo modelo. El desarrollo final, llamado DMDLP, demostró que los algoritmos tradicionales de reducción tambien pueden desempeñar bien en entornos distribuidos y, al mismo tiempo, ser útiles para el proceso KDD. El siguiente reto en este campo consistía en aprovechar la extraordinaria habilidad de los algoritmos evolutivos para crear un discretizador competitivo cuyo alcance no se limite únicamente al escenario secuencial. El primer compromiso fue alcanzado con excelencia por EMD. Esto superó empíricamente el actual estado del arte en discretización gracias a su naturaleza evolutiva y al enfoque multivariado implementado. El segundo objetivo fue finalmente en un trabajo de extensión que originó DEMD, un discretizador multivariado distribuido basado en optimización evolutiva. A pesar de emular a su predecesor vía un procedimiento aproximado, DEMD proporcionó esquemas de discretización competitivos para diversos problemas reales de gran escala. Los resultados empíricos reconocen la superioridad en simplicidad y precisión de los esquemas de DEMD con respecto a los generados por su competidor (DMDLP).

La segunda parte de esta tesis está dedicada a la reducción de flujos de datos. Una vez más, comenzamos estudiando el estado actual de la temática, así como proporcionando una revisión que analiza, categoriza y evalúa los últimos desarrollos en el área. También analizamos los próximos objetivos a ser abordados en la reducción para datos dinámicos. Los resultados derivados de este estudio fueron desiguales: algunos campos, como la selección de atributos resultaron estar bastante avanzados, mientras que otros, como la discretización, estaban todavía en sus primeros días con muchos problemas por tratar, incluso descubrir. Aunque la IS se eregía como campo bastante establecido, los resultados de nuestro estudio atestiguaron una claro insuficiencia en el rendimiento de las propuestas actuales cuando hacen frente a flujos de datos de alta velocidad.

Del análisis anterior se desprenden dos prometedoras ramas de investigación a abordar. En primer lugar, la necesidad de contar con esquemas evolutivos eficientes y eficaces en la discretización, así como la necesidad de hacer frente a la larga lista de problemas abiertos en este campo. Ambos compromisos fueron abordados en LOFD, un discretizer auto-adaptativo que se basa en estadísticas de clase actualizadas para aplicar cambios suaves en los límites de los intervalos. Como solución potencial para los problemas de etiquetado de intervalos y de interacción, propusimos e integramos en LOFD dos estrategias opuestas y completamente funcionales. Los experimentos confirmaron que LOFD es con diferencia la solución más competitiva en la literature actualmente. La segunda rama descansa sobre la necesidad de rápidos selectores de instancias que sean capaces de procesar varios flujos de datos de alta velocidad en paralelo. La respuesta a esta demanda fue DS-RNGE, la primera propuesta conocida en abordar este problema. DS-RNGE aplica IS a cada iteración para mantener una base de casos saneada y reducida a lo largo del tiempo. El resultado final es un sistema preciso y escalable capaz de procesar miles de instancias por segundo en un modesto clúster de máquinas.

9 Future work

From the conclusions drawn from this thesis, we envision some promising research lines for future work. They aim at either improving previous models, or to address new problems that have already emerged or surely will emerge from the ever-evolving Big Data scenario.

- **Evolutionary data reduction for large-scale instance selection**: EAs have shown to be well-reputed in dealing with IS in standard environments [GLH15]. Some of the most effective IS algorithm in the state-of-the-art include some type of evolutionary-based procedure to optimize selections. Benefits of using EAs come from the flexibility provided and their fitness to the objective target in combination with a robust behavior. However, the main flaw exposed by EAs is its poor scalability power [CLS+16]. Our next aim focuses on adapting these type of IS algorithms to be run in distributed platforms thus expecting a considerable leap in effectiveness.

- **Spark GPU-CPU integration and data reduction**: Recent technology advances have displaced CPU in rankings in favor of special-purpose acceleration provided by GPUs. Nevertheless, single-GPU or GPU-only cluster now cope with the challenges of processing big datasets. Spark, on the other hand, provides fast in-memory computing power for large cluster of nodes, not always with optimal performance and energy efficiency. A plausible solution would be to combine both perspectives to create faster clusters –accelerated by GPUs– with less machines –less deployment effort and less energy waste–. In fact, there already exist some young projects working to make this idea true [3], even some algorithms have been already proposed mixing GPUs and Spark in the specialized literature [GLBH17]. Our plan is to complement the current developments with data reduction alternatives that leverage from both sides.

- **Address the problem of dimensionality reduction with feature weighting**: Compared to FS techniques, feature weighting algorithms offer less constrained solutions to the dimensionality reduction problem (ranking vs. closed selection) [GLS+13]. On the other hand, some algorithms in the feature weighting literature (e.g., Relief) [KŠRŠ97] hinges upon class separability of instances to rank features. We may leverage the potential and knowledge of previous NN-based developments to create a preliminary approximation to the large-scale feature weighting problem.

- **Redundancy elimination on high-speed big data streams**: Although DS-RNGE perfectly fits its role of controlling the entry of noise in massive streaming systems, it circumvents the strong redundancy commonly attached to real-world datasets. In order to control the ever-growing size of case-bases over time and the constant flow of redundant data, we will put further effort on adding some kind of IS condensation technique to DS-RNGE. On doing so, the time cost cost derived from the edition and prediction phase will be sharply alleviated, at the same time maintaining the original effectiveness.

- **Concept-drift adaptation for instance selection**: Stationary of streaming data is being less and less frequent in modern problems. Renovated approaches that incorporates mechanisms to follow drifting movements are required. These mechanisms are aimed at rebuilding completely or partially the devised model as soon as the change occurs. For instance, in DS-RNGE, the distributed metric tree is built at the very beginning and left *ad infinitum* until

[3] `http://www.spark.tc/gpu-acceleration-on-apache-spark-2/`

the end. Data streams, however, may modify their properties thus demanding a shift in the host tree. We plan to tackle this challenge by implementing a drift detection module, and/or by using instance weighting with forgetting to allow for smooth adaptation to changes.

- **Promoting active learning solutions:** Many works on supervised learning in streaming scenarios assume that class labels become available soon after the instance was being classified by the system, or arrive with some delay. However, the costs of labeling the entire data stream are far from realistic and thus we must deal with limited availability of true class labels. Our plan is to move towards more realistic designs that assume the lack of labels. In online discretization, for instance, it would be possible to only inspect labels from those samples with highest probability of influencing the intervals' definitions.

Chapter II

Publications: Published and Submitted Papers

1 Big Data reduction on static databases

The journal paper associated to this part is:

1.1 Big Data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce

- S. Ramírez-Gallego, A. Fernández, S. García, M. Chen, F. Herrera, Big Data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce. Information Fusion 42 (2018) 51–61, doi: 10.1016/j.inffus.2017.10.001
  - Status: **Published**.
  - Impact Factor (JCR 2016): 5.667
  - Subject Category: Computer Science, Artificial Intelligence. Ranking 9 / 133 (**Q1**).
  - Subject Category: Computer Science, Theory & Methods. Ranking 4 / 104 (**Q1**).

Information Fusion 42 (2018) 51–61

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Big Data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce

Sergio Ramírez-Gallego*,[a], Alberto Fernández[a], Salvador García[a], Min Chen[b], Francisco Herrera[a]

[a] Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain
[b] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China

A R T I C L E I N F O

Keywords:
Big Data Analytics
MapReduce
Information fusion
Spark
Machine learning

A B S T R A C T

We live in a world were data are generated from a myriad of sources, and it is really cheap to collect and storage such data. However, the real benefit is not related to the data itself, but with the algorithms that are capable of processing such data in a tolerable elapse time, and to extract valuable knowledge from it. Therefore, the use of Big Data Analytics tools provide very significant advantages to both industry and academia. The MapReduce programming framework can be stressed as the main paradigm related with such tools. It is mainly identified by carrying out a distributed execution for the sake of providing a high degree of scalability, together with a fault-tolerant scheme.

In every MapReduce algorithm, first local models are learned with a subset of the original data within the so-called Map tasks. Then, the Reduce task is devoted to fuse the partial outputs generated by each Map. The ways of designing such fusion of information/models may have a strong impact in the quality of the final system. In this work, we will enumerate and analyze two alternative methodologies that may be found both in the specialized literature and in standard Machine Learning libraries for Big Data. Our main objective is to provide an introduction of the characteristics of these methodologies, as well as giving some guidelines for the design of novel algorithms in this field of research. Finally, a short experimental study will allow us to contrast the scalability issues for each type of process fusion in MapReduce for Big Data Analytics.

1. Introduction

Big Data Analytics is nowadays one of the most significant and profitable areas of development in Data Science [1–6]. One of the main reasons of its success is related with the Internet-of-Things (IoT), the Web 2.0 and the social networks, and all the myriad of data from different sources that can be collected and processed [6–8]. In this sense, corporations that are able to extract valuable knowledge from large volumes of data in a reasonable time, may obtain significant advantages over their competitors [9,10]. Researchers from academia are also aware of the interest in developing robust and accurate models for Data Mining in Big Data applications [11,12]. There is a clear growing rate in the number of research studies [13,14], and the trend is not expected to change in the short future.

However, even years after the boom of Big Data, there is still a misleading definition for the concept itself [15]. We must stress that the topic of Big Data is strongly linked with the scalability issue [16]. Those models developed in this context must be able to adapt dynamically the data growth, as well as being fault tolerant to be reliable in case of time consuming operations. In order to fulfill these requirements, a change in the traditional technology and framework for carrying out the learning process is mandatory [17].

MapReduce (MR) has established as a de-facto solution that comprises all the previous capabilities [18–20]. It is basically an execution environment which lays over a distributed file system [21]. By means of two simple functions, Map and Reduce, any implementation can be automatically parallelized in a transparent way for the programmer, also supporting by default the aforementioned fault-tolerant scheme.

- The Map function is devoted to divide the computation into different subparts, each one related to a partial set of the data.
- The Reduce function needs to fuse the local outputs into a single final model.

Whereas the procedure to be included in the Map task is, most times, straightforward to determine, the hitch comes when deciding how to carry out the models' fusion within the Reduce task. At this point, the design depends on many factors, namely whether the

* Corresponding author.
E-mail addresses: sramirez@decsai.ugr.es (S. Ramírez-Gallego), alberto@decsai.ugr.es (A. Fernández), salvagl@decsai.ugr.es (S. García), minchen2012@hust.edu.cn (M. Chen), herrera@decsai.ugr.es (F. Herrera).

http://dx.doi.org/10.1016/j.inffus.2017.10.001
Received 26 September 2017; Accepted 3 October 2017
Available online 10 October 2017
1566-2535/ © 2017 Elsevier B.V. All rights reserved.

submodels are different and independent among them, or they have a nexus for being able to join them directly.

In this paper, we aim at analyzing the different alternatives on process fusion for Big Data Analytics models under the MR framework. To do so, we propose a brief taxonomy distinguishing two types of approaches.

1. Direct fusion of models: approximate methods. We refer to those that carry out a direct fusion of partial models via an ensemble system [22].
2. Exact fusion for scalable models: distributed data and models' partition. In this case, they are those designs that carry out a global distribution of both data and sub-models (the prior types mentioned just considered data division), and iteratively build the final system.

We will carry out a practical study on scalability for each of the different fusion proposals with the sake of contrasting how time and accuracy performance vary as resources increase. In order to provide a better understanding of each type of implementation, we will present some case studies of these algorithms from both well-known Machine Learning libraries such as Mahout [23–25] (from Apache Hadoop [26,27]) and MLlib [28] (from Apache Spark [29,30]).

In order to address all these objectives, this paper is organized as follows. First, Section 2.1 presents an introduction on the MR programming framework, also stressing some alternatives for Big Data processing. Section 3 includes an overview on those technologies currently available to address Big Data problems from a distributed perspective. Section 4 presents the core of this paper, analyzing the different design options for developing Big Data Analytics algorithms regarding how the partial data and models are fused. Then, we show a case study in Section 5 to contrast the capabilities regarding scalability of the different approaches previously introduced. In Section 6 we present a discussion on the findings obtained in this research, as well as several guidelines for future study on the topic. Finally, Section 7 summarizes and concludes this paper.

2. MapReduce as information and process fusion

The rapid growth and influx of data from private and public sectors, and the novel opportunities derived from the IoT [31], have popularized the notion of "Big data [3,4,11]". This scenario has led to the development of custom paradigms for distributed processing that are able to extract significant value and insight in different areas such as Bioinformatics [32], health care [33], social mining [34,35], and so on.

Although focused on standard processing, distributed paradigms have also been widely utilized for fusing information [36,37] (ontologies and genetic data) and learning models [38,39] (trees and fuzzy rules). This section describes in detail these paradigms, paying more attention to the most widespread paradigm in the market: MR. Furthermore, several examples on how MR is applied as a fusion process are given here.

2.1. MapReduce programming model

The MR execution environment [18] is the most common paradigm used in the distributed processing scenario. Being a privative tool, its open source counterpart, known as Hadoop, has been traditionally used in academia research [27]. It has been designed to allow distributed computations in a transparent way for the programmer, also providing fault tolerance, automatic data partition and management, and automatic job/resource scheduling. To take advantage of this scheme, any algorithm must be divided into two main stages: Map and Reduce. The first one is devoted to split the data for processing, whereas the second collects and aggregates the results.

Additionally, the MR model is defined with respect to an essential data structure: the (key,value) pair. The processed data, the intermediate and final results work in terms of (key,value) pairs. To summarize its procedure, Fig. 1 illustrates a typical MR program with its *Map* and *Reduce* steps.

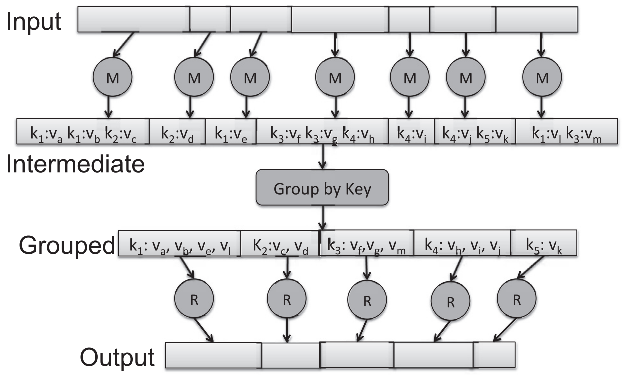

Input
M
k_1:v_a k_1:v_b k_2:v_c
k_2:v_d
k_1:v_e
k_3:v_f k_3:v_g k_4:v_h
k_4:v_i
k_4:v_j k_5:v_k
k_1:v_l k_3:v_m
Intermediate
Group by Key
Grouped
k_1: v_a, v_b, v_e, v_l
K_2:v_c, v_d
k_3: v_f,v_g, v_m
k_4: v_h, v_i, v_j
k_5: v_k
R
Output

Fig. 1. The MapReduce programming model. *k* elements represent the keys in the pairs, whereas *v* the values.

The MR scheme can be described as follows.

- Map function first reads data and transforms records into a key-value format. Transformations in this phase may apply any sequence of operations on each record before sending the tuples across the network.
- Output keys are then shuffled and grouped by key value so that coincident keys are grouped together to form a list of values. Keys are then partitioned and sent to the Reducers according to some key-based scheme previously defined.
- Finally, the Reducers perform some kind of fusion on the lists to eventually generate a single value for each pair. As a further optimization, the reducer is also used as a combiner on the map outputs. This improvement reduces the total amount of data sent across the network by combining each word generated in the Map phase into a single pair.

Apart from considering MR as a processing paradigm, this scheme (concretely, the Reduce stage) can also be seen as a fusion process that allows to blend partial models and information schemes into a final fused outcome. Fusion of models in MR is typically performed following some sort of ensemble strategy that combine multiple hypothesis through voting or attachment. Also other proposals exist that go beyond ensemble learning, and offers as outcome a single coalesced model. For example, logistic regression in Spark is composed by several sub-gradients that are locally computed and eventually aggregated (more examples will be given in Section 4). From another perspective, we may refer to aggregation of partial information collected within different maps. In this case, the fusion process will be more dependent from the input domain. The amount of scenarios that can be found in this context is highly diverse. Use cases for MR in the literature range from fusion of ontologies [36] to the composition of fuzzy rules [39], among others.

Word Count comes to be one of the most widespread examples to illustrate the intrinsic information fusion process behind MR. WordCount is intended to count the number of occurrences per word in a set of input text files. Each mapper reads a set of blocks formed by lines, and splits them into words. It then emits a key-value pair with the word as key, and 1 as value. Afterwards, each reducer sums the scores for each word, and outputs a single key-value pair with the word and sum.

Consider the phrase 'knock, knock, who is there?". A single mapper would receive and split this sentence as words, and then, it would form the initial pairs as: (knock,1), (knock,1), (who,1), (is,1), (there,1). In reducers, the keys are grouped together and the count values for identical keys (words) are added. In this case only one pair of similar keys 'knock' would be aggregated so that the output pairs would be as

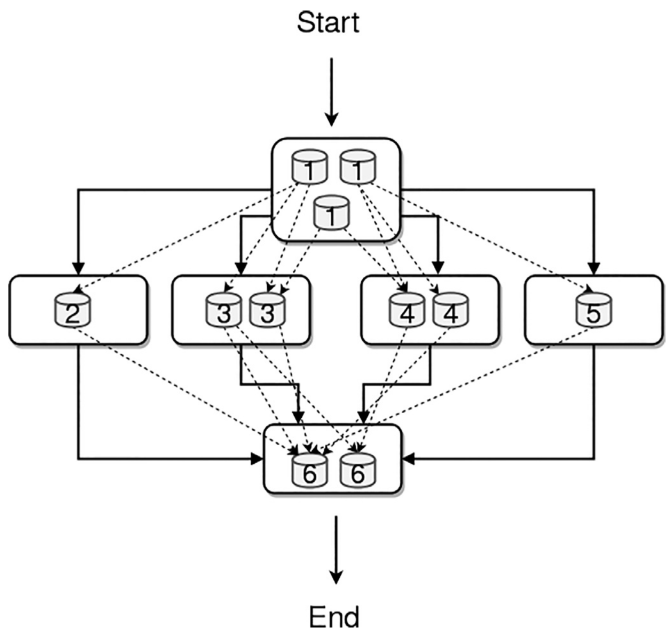


Fig. 2. Direct Acyclic Graph Parallel Processing. Squares represent the tasks to process and the nodes in the graph, arrows connecting nodes represent the data flow between nodes and the vertexes in the graph, dashed lines represent the dependencies between data blocks (cylinders).

follows: (knock,2), (who,1), (is,1), (there,1). As result, the user would receive the final number of hits for each word.

2.2. *Alternative distributed processing paradigms*

Although MR is the most popular paradigm to tackle distributed processing, there exist other modern distributed frameworks, conceived prior to the dawn of MR, that offer other alternatives for information and model fusion. Most of them follows a Single Instruction Multiple Datasets (SIMD) [40] scheme to execute the same sequence of instructions simultaneously on a distributed set of data partitions. In this section we focus on two paradigms (graph and bulk processing) relevant for current Big Data platforms.

2.2.1. *Directed Acyclic Graph Parallel processing*

All distributed frameworks based on Directed Acyclic Graph (DAG) [41], like Spark, organize their jobs by splitting them into a smaller set of atomic tasks. In this model vertexes correspond with parallel tasks, whereas edges are associated with exchange of information. As shown in Fig. 2, vertexes can have multiple connections between inputs and outputs, which imply that the same task can be run in different data and the same data in different partitions. Data flows are physically supported by shared memory, pipes, or disks. Instructions are duplicated and sent from the master to the slave nodes for a parallel execution. Notice that MR can be deemed as an specific implementation of DAG-based processing, with only two functions as vertexes.

2.2.2. *Bulk Synchronous Parallel processing*

Bulk Synchronous Parallel (BSP) [42] systems are formed by a series of connected supersteps, implemented as directed graphs. In this scheme input data is the starting point. From here to the end, a set of Supersteps are applied on partitioned data in order to obtain the final output. As mentioned before, each Superstep correspond with an independent graph associated with a subtask to be solved. Once all compounding subtasks end, bulk synchronization of all outputs is committed. At this point vertexes may send messages to the next Superstep, or receive some from previous steps, and also to modify its state and outgoing edges. Fig. 3 shows a toy example for BSP processing with two Supersteps and one synchronization barrier.

3. Big Data technologies for analytics

Nowadays, the volume of data currently managed by our storage systems have surpassed the processing capacity of traditional systems [11], and this applies to Data Mining as well. Distributed computing has been widely used by experts and practitioners before the advent of Big Data to boost up sequential solutions in medium-size data. Nevertheless, for most of current massive problems, a distributed approach becomes mandatory nowadays since no batch architecture is able to address such magnitudes.

Beyond High Performance Computing solutions, new large-scale processing platforms are intended to bring closer distributed processing to practitioners and experts by hiding the technical nuances derived from these environments. Novel and complex designs are required to create and maintain these platforms, which generalizes the utilization of distributed computing for standard users.

As a result of the fast evolving of Big Data environment, a myriad of tools, paradigms and techniques have surged to tackle different use cases in industry and science. However, because of this large number of alternatives, it is often difficult for practitioners and experts to analyze and select the right tool for each goal.

In this section we present and analyze three well-known alternatives for distributed processing belonging to the "Apache Hadoop ecosystem." The objective is providing the necessary knowledge that helps users to decide which alternative better fits their requirements. We also outline the software libraries that gives support to the distributed learning task in these platforms, being summarized in Table 1.

3.1. *Apache Hadoop*

Undoubtedly Hadoop MapReduce may be deemed as the primary platform in the Big Data space. After the presentation of MR by Google designers [43], Hadoop MapReduce was grown by the community, and became the most used and powerful open-source implementation of MR. Nowadays leading companies such as Yahoo has scaled from 100-node Hadoop clusters to 42K nodes and hundreds of petabytes [44] thanks to the outstanding performance of Hadoop.

The main idea behind Hadoop was to create a common framework which can process large-scale data on a cluster of commodity hardware, without incurring in a high cost in developing (in contrast to HPC solutions) and execution time. Hadoop MapReduce was originally composed by two elements: the first one was a distributed storage system called Hadoop Distributed File System (HDFS), whereas the second one was a data processing framework that allows to run MR-like jobs. Apart from these goals, Hadoop implements primitives to address cluster scalability, failure recovery, and resource scheduling, among others.

But Hadoop is today more than a single technology, but a complete software stack and ecosystem formed by several top-level components that address diverse purposes. For instance, Apache Giraph for graph processing or Apache Hive for data warehousing. The common factor is that all of them rely on Hadoop, and are tightly linked to this technology. Some projects are actually Apache top-level projects [45], whereas others are continuously evolving or being created.

HDFS [46] can be deemed as the main module of Apache Hadoop. It supports distributed storage for large-scale data through the use of distributed files, which themselves are composed by fixed-size data blocks. These blocks or partitions are equally distributed among the data nodes in order to balance as much as possible the overall disk usage in the cluster. HDFS also allows replication of blocks across different nodes and racks. In HDFS, the first block is ensured to be placed in the same processing node, whereas the other two replicas are sent to different racks to prevent abrupt ends due to inter-rack issues.

HDFS was thought to work with several storage formats. It offers several APIs to read/write registers. Some relevant APIs are:

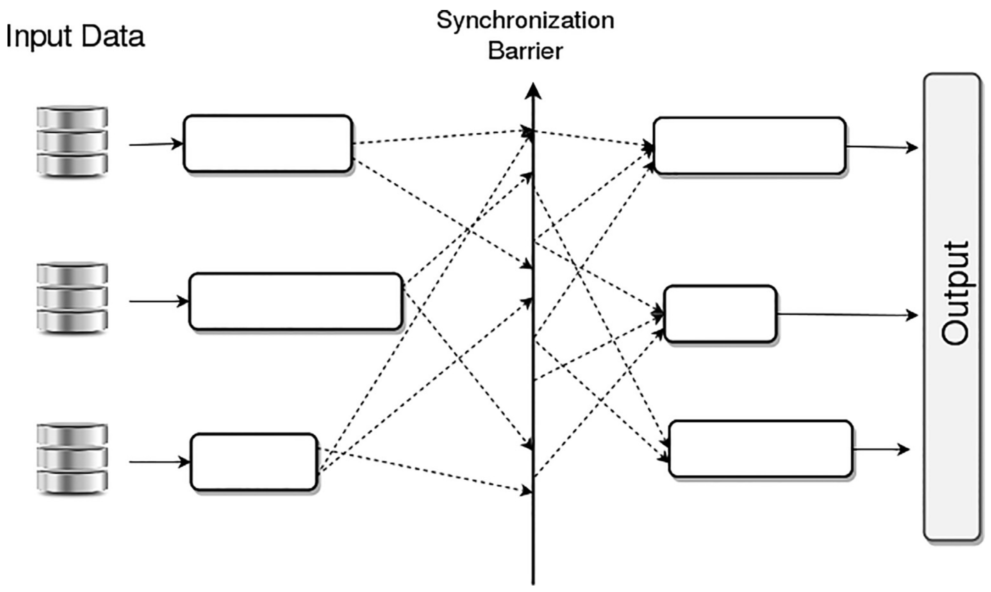


Fig. 3. Bulk Synchronous Parallel processing. Subtasks in each Superstep are depicted as rectangles with variable height (task duration), and data flows as dashed lines. Synchronization barrier acts as a time proxy between stages.

Table 1
Analytics tools for each Big Data platform.

Big Data distributed platforms	Analytics tools
Hadoop	Mahout
Spark	MLlib
Flink	FlinkML

InputFormat (to read customizable registers), or RecordWriter (to write record-shaped elements). Users can also developed their own storage format, and to compress data according to their requirements. Persistence in Hadoop is mainly performed in disk. However, there are some novel advances to optimize persistence by introducing some memory usage. For instance, in Apache Hadoop version 3.0 was introduced the option of memory usage as temporary storage.

Although **MR** [43] is the native processing solution in Apache Hadoop, today it supports multiple alternatives with different processing schemes. All these solutions have in common that use a set of data nodes to run tasks on the local data blocks, and one master node (or more) to coordinate these tasks. For instance, **Apache Tez** [47] is a processing engine that transforms processing jobs into direct acyclic graphs (DAGs). Thanks to Tez, users can run any arbitrary complex DAG of jobs in HDFS. Tez thus efficiently solves interactive and iterative processes, like those present in machine learning processes. Its most relevant contribution is that Tez translate any complex job to a single MR phase. Furthermore, it does not need to store intermediate files and reuses idle resources, which highly boost the overall performance.

Hadoop MapReduce evolves to a more general component, called **Yet Another Resource Negotiator (YARN)** [48], which provides extra management and maintenance services relied to other components in the past. YARN also acts as a facade for different types of distributed processing engines based on HDFS, such as Spark, Flink or Storm. In short, YARN was intended as a generic purpose system that separates the responsibilities of resource management (performed by YARN), and running management (performed by top-level applications).

Among the full set of advantages claimed by YARN, we can highlight its capacity to run several application on the same cluster without the necessity of moving data. In fact, YARN allows reusing resources across alike applications in parallel, which improves the overall usage of resources.

3.1.1. *Apache Mahout*

Since the magnitude of learning problems has been growing exponentially, data scientists demands rapid tools that efficiently extract knowledge from large-scale data. This problem has been solved by MR and other platforms by providing scalable algorithms and miscellaneous utilities in form of machine learning libraries. These libraries are compatible with the main Hadoop engine, and use as input the data stored in the storage components.

Apache Mahout [49] was the main contribution from Apache Hadoop to this field. Although it can be deemed as mainly obsolete nowadays, Mahout is considered as the first attempt to fill the gap of scalable machine learning support for Big Data. Mahout comprises several algorithms for plenty of tasks, such as: classification, clustering, pattern-mining, etc. Among a long list of golden algorithms in Mahout, we can highlight Random Forest or Naïve Bayes.

The most recent version (0.13.0) provides three new major features: novel support for Apache Spark and Flink, a vector math experimentation for R, and GPU support based on large matrix multiplications. Although Mahout was originally designed for Hadoop, some algorithms have been implemented on Spark as a consequence of the latter one's popularity. Mahout is also able to run on top of Flink, being only compatible for static processing though.

3.2. *Spark*

Apache Spark Framework [50] was born in 2010 with the publication of Resilient Distributed Datasets (RDD) structures [30], the keystone behind Spark. Although Spark has a close relationship with Hadoop Ecosystem, it provides specific support for every step in the Big Data stack, such as its own processing engine, and machine learning library.

Apache Spark [51] is defined as a distributed computing platform which can process large volume data sets in memory with a very fast response time due to its memory-intensive scheme. It was originally thought to tackle problems deemed as unsuitable for previous disk-based engines like Hadoop. Continued use of disk is replaced in Spark by memory-based operators that efficiently deal with iterative and interactive problems (prone to multiple I/O operations).

As stated previously, the heart of Spark is formed by **RDDs**, which transparently controls how data are distributed and transformed across the cluster. Users just need to define some high-level functions that will be applied and managed by RDDs. These elements are created whenever data are read from any source, or as a result of a transformation.

RDDs consist of a collection of data partitions distributed across several data nodes. A wide range of operations are provided for transforming RDDs, such as: filtering, grouping, set operations, among others. Furthermore RDDs are also highly versatile as they allows users to customize partitioning for an optimized data placement, or to preserve data in several formats and contexts.

In Spark, fault tolerance is solved by annotating operations in a structure called lineage. Spark transformations annotated in the lineage are only performed whenever a trigger I/O operations appears in the log. In case of failure, Spark re-computes the affected brach in the lineage log. Although replication is normally skipped, Spark allows to spill data in local disk in case the memory capacity is not sufficient.

Spark developers provided another high-level abstraction, called **DataFrames**, which introduces the concept of formal schema in RDDs. DataFrames are distributed and structured collections of data organized by named columns. They can be seen as a table in a relational database or a dataframe in R, or Python (Pandas). As a plus, relational query plans built by DataFrames are optimized by the Spark's Catalyst optimizer throughout the previously defined schema. Also thanks to the scheme, Spark is able to understand data and remove costly Java serialization actions.

A compromise between structure awareness and the optimization benefits of Catalyst is achieved by the novel Dataset API. **Datasets** are strongly typed collections of objects connected to a relational schema. Among the benefits of Datasets, we can find compile-time type safety, which means applications can be sanitized before running. Furthermore, Datasets provide encoders for free to directly convert JVM objects to the binary tabular Tungsten format. These efficient in-memory format improves memory usage, and allows to directly apply operations on serialized data. Datasets are intended to be the single interface in future Spark for handling data.

3.2.1. MLlib

MLlib project [28] was born in 2012 as an extra component of Spark. It was released and open-sourced in 2013 under the Apache 2.0 license. From its inception, the number of contributions and people involved in the project have been growing steadily. Apart from official API, Spark provides a community package index [52] (Spark Packages) to assemble all open source algorithms that work with MLlib.

MLlib is a Spark library geared towards offering distributed machine learning support to Spark engine. This library includes several out-of-the-box algorithms for alike tasks, such as: classification, clustering, regression, recommendation, even data preprocessing. Apart from distributed implementations of standard algorithms, MLlib offers:

- Common Utilities: for distributed linear algebra, statistical analysis, internal format for model export, data generators, etc.
- Algorithmic optimizations: from the long list of optimizations included, we can highlight some: decisions trees, which borrow some ideas from PLANET project [53] (parallelized learning both within trees and across them); or generalized linear models, which benefit from employing fast C++++-based linear algebra for internal computations.
- Pipeline API: as the learning process in large-scale datasets is tedious and expensive, MLlib includes an internal package (*spark.ml*) that provides an uniform high-level API to create complex multi-stage pipelines that connect several and alike components (preprocessing, learning, evaluation, etc.). *spark.ml* allows model selection or hyper-parameter tuning, and different validations strategies like k-fold cross validation.
- Spark integration: MLlib is perfectly integrated with other Spark components. Spark GraphX has several graph-based implementations in MLlib, like LDA. Likewise, several algorithms for online learning are available in Spark Streaming, such as online k-Means. In any case, most of component in the Spark stack are prepared to effortlessly cooperate with MLlib.

3.3. Flink

Apache Flink [54] is a distributed processing component focused on streaming processing, which was designed to solve problems derived from micro-batch models (Spark Streaming). Flink also supports batch data processing with programming abstractions in Java and Scala, though it is treated as a special case of streaming processing. In Flink, every job is implemented as a stream computation, and every task is executed as cyclic data flow with several iterations.

Flink provides two operators for iterations [55], namely, standard and delta iterator. In standard iterator, Flink only works with a single partial solution, whereas delta iterator utilizes two worksets: the next entry set to process and the solution set. Among the set of advantages provided by iterators is the reduction of data to be computed and sent between nodes [56]. According to the authors, new iterators are specially designed to tackle machine learning and data mining problems.

Apart from iterators, Flink leverages from an optimizer that analyzes the code and the data access conflicts to reorder operators and create semantically equivalent execution plans [57,58]. Physical optimization is then applied on plans to boost data transport and operators' execution on nodes. Finally, the optimizer selects the most resource-efficient plan, regarding network and storage.

Furthermore, Flink provides a complex fault tolerance mechanism to consistently recover the state of data streaming applications. This mechanism is generating consistent snapshots of the distributed data stream and operator state. In case of failure, the system can fall back to these snapshots.

FlinkML is aimed at providing a set of scalable ML algorithms and an intuitive API to Flink users. Until now, FlinkML provides few alternatives for some fields in machine learning: SVM with CoCoA [59], or Multiple Linear regression for supervised learning, k-NN join for unsupervised learning, scalers and polynomial features for preprocessing, Alternating Least Squares for recommendation, and other utilities for validation and outlier selection, among others. FlinkML also allows users to build complex analysis pipelines via chaining operations (like in MLlib). FlinkML pipelines are inspired by the design introduced by sklearn in [60].

3.4. Comparison among Big Data alternatives (Hadoop, Spark and Flink)

Main divergence between Big Data frameworks rests on its design philosophy, and how they respond to different data formats such as data streams. Starting from Hadoop, we can directly assert that Hadoop is essentially batch-oriented due to its intensive disk usage. In contrast, Flink is a native streaming technology originally designed to work with memory-based streams. Although Apache Spark was not originally designed for static problems, it provides a micro-batching strategy capable of easily processing streaming data. Spark micro-batching however may be deteriorated when it faces low latency requirements.

Unlike Hadoop MapReduce, Spark and Flink both offer native support for in-memory persistence and iterative processing. Spark allows users to persist data in memory, and load them in several occasions. Regarding the execution engine, Spark relies on an acyclic graph planner formed by vertices and edges, which can be seen as a strict generalization of MapReduce. In counterpart, Flink utilizes a thoroughly iterative processing scheme created from the scratch, which is based on cyclic data flows (a single iteration per schedule).

Moving to optimization matters, Spark and Flink both provide mechanisms to analyze, control and optimize user code so that the best execution plan for each program is obtained. Spark mainly exploits new SQL-based DataFrame API and the Tungsten engine for optimization, whereas Flink did that as first citizen. Manual optimization can also be carried out by controlling how data are partitioned, or transmitted across the network.

Finally, Apache Spark and Flink offer plenty of alternatives to coders specially attached with a given programming language. Namely, Spark offers ad-hoc APIs for R, Java or Python, and a native support for the Scala language, whereas Apache Flink only focus on Java Virtual Machine, providing full APIs for Scala and Java. In contrast, Hadoop only supports Java. For further comparison insights, please refer to García-Gil et al. [61].

4. Big Data analytics as a process fusion

"Synchronizing flags" in every distributed algorithmic approach is undoubtedly the most challenging part during the design process. This issue is not an exception in the MR scheme. In this particular case, developers must take two significant decisions. On the one hand, the components selected for the key-value representation in both the Map and Reduce input and outputs. On the other hand, how the list of values are aggregated in the Reduce step. In this section, we want to analyze in detail this characteristic of the MR programming environment by introducing a taxonomy to organize current state-of-the-art approaches regarding how partial models from the Map task are aggregated. We will determine how this fact may also affects the actual scalability of the whole system.

Specifically, we distinguish among two different type of models according to the fusion strategy implemented. First, those that produce an approximate model by applying a direct process fusion on partial submodels (Section 4.1). Second, those that distribute both data and models to iteratively build a final exact system (Section 4.2).

Besides this standard categorization, some further considerations must be taken into account in order to properly classify large-scale learning algorithms. Among others, we must distinguish between:

- Whether the model (or required statistics) is evaluated (or are computed) on all the distributed partitions (global), or local submodels are independently yielded by each task and then fused (local).
- Whether algorithms only consist of a single stage (1-step) or several iterations/stages are required (multistage).
- Whether a master node guides the model construction process (guided) or it is fully decoupled (unguided).

Table 2 enumerates and categorizes the distributed methods described below according to the previous taxonomy. Note that although some categories appear more frequently with others, all categories are independent.

4.1. Direct fusion of models: approximate methods

Roughly, approximate distributed models are those that emulate the learning behavior of sequential algorithms, yet generating a completely different and non-exact solution. Most of them follows an ensemble paradigm, which is straightforwardly parallelizable in the MR framework following the rule of thumb: one submodel per mapper. The compounding model will be eventually generated by directly joining all submodels in the reduce phase [71]. Note that the MR approach reminds of the traditional *bootstrap aggregating* (bagging) approach; yet in this case with no replacement.

Table 2
Categorization of distributed models for large-scale machine learning. Pseudonym and reference for each method are provided.

Method	Fusion tactic	Model scope	Model phases	Model guidance
Mahout-RF [23]	approximate	local	1-step	unguided
FRBS [62,63]	approximate	local	1-step	unguided
Boost.PL [64]	approximate	local	1-step	unguided
Spark-PR [65]	approximate	local	1-step	unguided
Spark-Trees [66]	exact	global	multistage	guided
Spark-Gradient [67]	exact	global	multistage	unguided
Spark-kMeans [68]	exact	global	multistage	guided
Spark-kNN [69]	exact	global	1-step	unguided
IFSF [70]	exact	local	multistage	guided

The premise for this type of methodologies is simple yet effective. Each Map task works independently on its chunk of data. Depending on the user's requirements, each Map function is devoted to build one or more models. For example, a pool of 100 classifiers to be obtained from 5 Maps will result on 20 classifiers per Map, each one of them built from a 20%% of the original dataset. The "key" is shared by all values given as output from the Map. This fact makes the logic of the Reduce phase to stress for its simplicity. Every single classifier is directly joined into the final system. In the following, we shortly describe some algorithms.

- One of the first approaches to build an approximate ensemble with MapReduce was the Random Forest for Hadoop [72,73], whose implementation can be found at Mahout-MapReduce Machine Learning library [23,24]. The algorithm consist of two MR phases: the first one is devoted to the development of the model (where model fusion occurs), and the second one is focused on class prediction once the model is built.
  In the first phase, each Map creates several random trees (a subset of the forest) using the partitions given as input [74]. The reduce phase simply concatenates all the trees forming the final forest of random trees. The second task replicates the forest across the nodes, and launches the mappers on the test set partitions. Mappers predict the class for each record assigned by using the final model. Finally, reducers concatenate predictions in a single file.
- Another interesting approaches come from the boosting perspective [75], which are AdaBoost.PL, LogitBoost.PL and MultiBoost.PL [64]. These include an ensemble of ternary classifiers [76], or an ensemble of ensembles. In this case, the reduce phase is much more complex than applying a simple voting or coalescing scheme. Concretely, the whole boosting procedure, i.e. T iterations, is carried out within each Map, so M ensembles of T classifiers are obtained, being M the number of maps. Then, the T classifiers are arranged according to their score, i.e. the weighted error, and then emitted to the Reducers using their index as "key" and the model as value. Finally, each Reduce process takes all classifiers with the same index (key) and compute an average weight for this ensemble. At classification time, each "sub-ensemble" provides a vote for its class, whose value is exactly the aforementioned weighted average score.
- Prototype Reduction [65] is also a another significant example that provides a further complex and effective reduce process. First, each Map process works with a different chunk of data by applying any available reduction procedure [77]. Then, selected prototypes are fed to a single reducer that is devoted to eliminate redundant ones. This implies a clear fusion of data in order to obtain a final model, in this case by merging similar examples.
- Fuzzy Rule Based Classification Systems [78] are probably one of the clearest type of models in this category. The pioneer implementation was based on the Chi et al. approach [62,63]. Each Map task comprises a complete fuzzy learning procedure to obtain an independent rule base. To do so, all examples from the input data chunk are iterated deriving a single rule per example, while merging those with the same antecedent and consequent. Then, rule weights are computed based on a fuzzy confidence value from the input examples, also allowing to determine the output class. We must acknowledge that at this stage every single rule base might be used for classification purposes. However, the system can be further enhanced by combining all partial models for every Map, as stated in the beginning of this section. To do so, the Map writes as key-value pair the antecedent and consequent (including class and rule weight) respectively. Reducers then merges those rules with the same antecedent (key) by averaging the values of the rule weights

for the same class consequents, and then the class with the highest rule weight is finally maintained.

There are two main advantages for this type of design. On the one hand, we have stressed its direct synergy with the MR programming model, easing the programmer implementation. On the other hand, we must refer to the degree of diversity of those models obtained from different Maps, being a crucial characteristic for the success of these types of methods [79].

In spite of the previous goodness, there is one significant problem: there is a limit in the scalability degree. It must be acknowledge that there is a strong relationship between the number of Maps and the required number of models. In other words, we cannot decrease the total running time by adding a larger number of Maps, as this may result in "void" functions with a data input but no output.

One final drawback that must be stressed is related to the inner concept of ensemble. Being composed of a "large" set of individual models, two facts must be taken into account. On the one hand, since all of them are considered for the classification step, it is hard to understand the actual reason for the labeling of the queried data. On the other hand, the robustness of the whole ensemble system somehow depends on the individual quality of its components, as well as their diversity. Both properties are not easy to hold, and impose a restriction in the building of the ensemble, for both standard and Big Data applications.

Throughout this section we have presented a short overview on those MR methods that are based on fusion of partial models. As such, they allow to reinforce the abilities of those individual systems, leading to possibly more robust approaches. However, two issues must be taken into account:

1. The key-value representation to link the Map and Reduce processes. It must be very carefully selected as it has a strong influence in the design of the Map function. Although the procedure is usually implemented to be independent among Maps, a common point should be given in order to allow the final combination.
2. The fusion function in the Reducer. It has a clear dependence on the previous item, so that these must be designed as a whole to ensure a robust workflow.

Both points will determine the scalability and exactness of the output system. We refer to the changes in behavior when increasing the number of Maps. Concretely, the efforts must be mostly focused on the development of the partial models, regarding the influence of the lack of data and/or whether the knowledge acquired from different subsets of the problem space are able to be accurately merged.

4.2. Exact fusion for scalable models: distributed data and models' partition

The previous scheme based on approximate models is presented as ideal for distributed learning since submodel creation is naturally parallelizable and applicable to most of standard algorithms. Nevertheless, in approximate fusion, models suffers from several deficiencies, such as: extra parameter configuration (e.g.: number of map tasks), or a subsequent drop on generalization due to a narrower view in submodel creation. Indeed, most of supervised methods demand to access to a large percentage of instances (usually, the whole set) during the training step. In this scenario submodel creation is not possible because of the strong dependency between data partitions. Throughout this section, we analyze those algorithms whose scheme were noted as "exact" in Table 2.

Decision trees (DT), as supervised algorithms, are an example of methods that do not naturally support partial construction in distributed processing. They are top-down algorithms that continuously evaluate splits by using statistics sketches computed on the entire dataset. In this case, all instances (and data partitions) are involved on the statistics computation at every step, except in the rare scenario where all feature values in a partition belongs to leaves nodes. Parallelization here is then considered as a much more complex task.

A possible solution to this problem is to utilize partitioned data to make decisions about how to construct a single exact model. For DTs, statistics for each split considered are collected from the datanodes, and used in the master node to decide which split is the best option for the current state of the model (guided model construction).

This scheme was originally implemented in the Parallel Learner for Assembling Numerous Ensemble Trees (PLANET) method [53]. In **PLANET**, master node controls the complete tree induction process, and launches the MR jobs that construct the nodes. It also maintains the tree model in memory, decides which split predicates will be evaluated, and eventually replicates the updated model across the nodes. Nodes to be evaluated are organized through several queues, one with nodes for MR-based evaluation and another for in-memory evaluation. Depending on the amount of instances involved in computations, nodes are pushed to one queue or the other. Here we focus on MR evaluation of nodes which is more common.

As tree construction proceeds, the master node retrieves nodes from the queue, and schedules MR jobs to evaluate splits. Inside these jobs, statistics for splits are computed in the map side by using the instances in the data partitions. Finally, the reduce side decides which split is more convenient for the tree model. Once a MR job ends, the master nodes updates the model with the nodes and the splits predicates selected, and updates the queues with new nodes at the periphery.

Notice that in this scheme, each MR job fetches the entire dataset in order to avoid determining which records are required by each node, and the extra communication that this step conveys. Instead, it performs a level-wise computation of splits so that the tree is constructed by following a breadth-first strategy. Thanks to this scheme, all nodes at a given level are expanded at the same step, and every instance is part of the input to some analyzed node.

PLANET project does not only offer an elegant solution for single tree induction, but it also extends the original idea to provide powerful exact algorithms based on bagging and boosting ensembles [80]. Concerning bagging, PLANET allows to build multiple trees by maintaining a single queue of nodes for the whole set of trees. By alternating evaluation of nodes from several trees, PLANET can build the random forest in a parallel way. During boosting, PLANET constructs weak learners sequentially as usual. Here training is only parallelized at the tree level. Residuals are easily computed since the model is sent to every MR job in full.

Based on PLANET, MLlib' creators [28] developed two ensemble classifiers for exact fusion: one based on random forest of trees, and another based on gradient boosted trees [66]. Standard DTs are also adopted in MLlib, however, note that they are implemented as an singular case of random forest with a single tree and the full feature set. In general, all aforementioned tree-based algorithms have incorporated several optimizations [81] with respect to PLANET, intended to boost up the tree induction process. Major features introduced are describe below:

- *Efficient bootstrapping*: one of the major drawbacks in PLANET is that its does not allow bootstrapping in bagging. This fact was overcome with MLlib where each record has associated a vector that indicates the number of replicas of each instance in each tree. Note that this strategy reduces the total amount of memory used as replication is not performed.
- *Node tracking*: in order to keep simple its design, PLANET do not track the current position of instances in the tree as it evolves. Instead, every instance needs to be evaluated at every step by traversing the tree from the root. Although simpler, and less memory-consuming, this introduces an unfordable cost in time performance. MLlib solves this problem by assigning to each instance the node where it is currently stacked. It is clearly a better solution since CPU usage is an usual bottleneck for large-scale applications.

- *Enhanced group-wise training*: MLlib has improved the tree-level training scheme by extending the number of nodes to be evaluated to the memory available in the cluster for statistic aggregation. This means that several trees can evaluate complete levels of several trees at the same step, which extremely reduces the number of passes over the dataset.
- *Bin-wise computation*: Instead of directly implementing the best split computation strategy, MLlib proposes to categorize each feature value into a discrete bin. Bins are then exploited to easily compute aggregates for bins, and to calculate information gain.
- *Partition aggregation*: As bin-based categorization is known from the early stages, it is used by the learning process to reduce the number of key-value pairs to be sent across the network (less I/O usage). Each partition provides the aggregates in a single array for all the bins considered, which drastically simplify the scheme proposed by the original instance-wise map function.

Beyond tree learning, other scalable classifiers that generates exact models can be found in MLlib [67]. Non-linear classifiers such as support vector machines, logistic regression or neural networks rely on gradient descent optimization because of its great suitability for distributed computation, and thus these are implemented following a linear approach in MLlib library. All of them share the same objective function based on error minimization: $\min_{w \in \mathbb{R}^d} Q(w)$, which is solved by the aforementioned optimizer.

Without going into further details, gradient descent [80] aims at finding a local minimum of a convex function by iteratively moving towards the steepest descent, which corresponds with the negative of the derivative (gradient) of the function at a given point.

For those optimization problems whose objective function Q can be written as a sum of costs, a stochastic gradient descent (**SGD**) approach can be used. That is the case of supervised learning, where the loss and regularization parts can be decomposed in several single contributions (instance-wise) as shown below:

$$Q(w) := \lambda \, Reg(w) + \frac{1}{n} \sum_{i=1}^{n} Loss(w; x_i, y_i). \tag{1}$$

where λ represents the trade-off factor between regularization and loss, x the input vector, and y the output value.

In stochastic gradient descent, the exact result is approximated by locally computing the gradient at instance-level. In MLlib, SGD implementation computes (sub)gradients by partition (map phase) with an specific sampling fraction (mini-batches). If no sampling is applied, we get an exact gradient descent, otherwise, SGD scheme is performed at different levels. Partial results are then aggregated/sum to obtain the gradient contribution for each iteration (reduce phase).

In recent versions, the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (**L-BFGS**) method [82] was introduced as another alternative for optimization in MLlib. Because of great benefits in performance, it is gaining increasing popularity in MLlib compared to mini-batch gradient descent. The L-BFGS primitive locally approximates the objective function as a quadratic by constructing the Hessian matrix underneath. The Hessian matrix is approximated by previous gradient evaluations, thus solving the vertical scalability issue present in gradient descent.

Beyond classification, some clustering algorithms have been adapted and included in MLlib. For instance, original k-Means is implemented in Spark by replicating and transforming centroids in each Spark stage. Concretely, each Map process is devoted to compute the nearest samples (from its input chunk of data) to each of the k centroids (initially set at random). The output key-value is the centroid “id” and the attributes of the nearest sample respectively. Then each Reducer recalculates the new centroids coordinates by averaging the values of those samples given as “list of values” for each key (centroid). The whole procedure is then iterated in order to refine the centroids, thus improving their robustness. It must be observed the importance of the Reduce step for the global combination of the partial information computed within each Map. k-Nearest Neighbor (kNN) classifier [69] has also some points in common with clustering, as it is based on distances among examples. In this case, the Map stage is devoted to obtain the k nearest training instances to each query input. To carry out an exact computation, the whole test set should be given as input to each Map, allowing to obtain as key-value the index of the test instance and a list of the k closest distances with their corresponding training class labels. Then, the Reducer only needs to find, among all M sets of k-nearest neighbors (with M the number of Maps) the one with lowest value to finally assign the output class. Compared to k-Means (which updates centroids), model guidance is more decoupled in k-NN where the lazy model (case-base) is distributed across workers.

The information-based feature selection framework (IFSF) proposed in [70] is an example of how submodels (importance score by feature) can be fused in data preprocessing. This multivariate algorithm measures redundancy and relevance between the input features and the output class in order to select a final reduce set of features. As a first step in IFSF, input data is vertically split to strictly separate computations between features. Then feature interactions are modeled by replicating the last selected feature and the class across the nodes so that the minimum amount of communication is performed. The map stage calculates the feature scores individually, and the reduce stage selects the best feature from the current set of unselected features. Notice that in this case, model's scope is local since input data is in column-format. This format allows Spark to compute feature scores independently in each partition. The generalization from the perspective of information theory is shown in a recent work in [83].

5. Practical study on scalability

In this section we present a brief experimental study on the scalability of different fusion proposals. We start by presenting Higgs dataset, the large-scale dataset used as reference in our experiments (Section 5.1). Then, the random forest version for Mahout-Hadoop will serve us to illustrate the most elementary fusion process: the approximate strategy based on direct fusion (Section 5.2). Finally, Section 5.3 provides outcomes for exact models, represented by Spark's k-Means and Random Forest.

Experiments have been launched in a 12-node cluster and an extra master node with the following features per node: 6 CPU cores (Intel(R) Core(TM) i7-4930K CPU at 3.40 GHz), 64GB RAM, 2TB HDD, Gigabit Ethernet network connection. Hadoop 2.6.0-cdh5.10.0 and Mahout 0.9 were installed in all the nodes. 4GB was set as maximum for memory in map and reduce processes.

5.1. Higgs boson data

Higgs boson dataset [84] has been elected to measure the performance impact of distinct fusion schemes on standard learning methods. This dataset was uploaded to the UCI repository in 2014 [85] as a result of Higgs's discovery in 2012, and the complex experiments performed at the Large Hadron Collider at CERN.

In this dataset, particle data generated by Monte Carlo simulations aim at distinguishing between a signal process generated by Higgs bosons, and a background process with the identical decay products but distinct kinematic attributes (binary problem). Although experts have confirmed its decay into two tau particles, the signals are rather small and buried in background noise.

From the total of 28 numerical features, the first 21 are kinematic feature measured by the particle sensors in the accelerator. The last features are high-level functions derived by physicists to improve the discriminate power of features. Finally, 11 millions of instances were collected for the entire dataset.

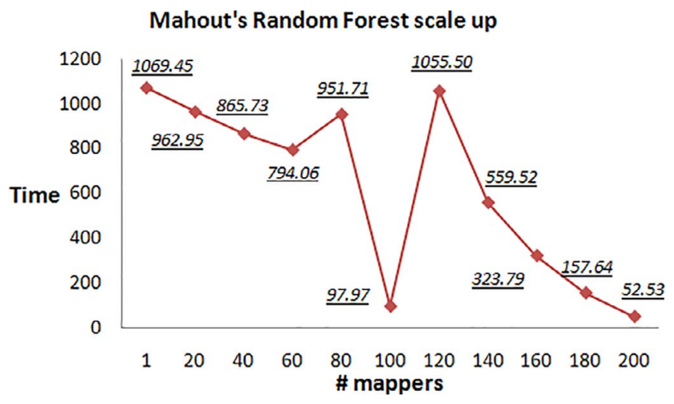


Fig. 4. Average time (3 runnings) obtained by Mahout's Random Forest (200 trees) varying the number of mappers. Label values are displayed close to the points.

5.2. Approximate models: random forest on Mahout-Hadoop

In order to test the scalability behavior of Mahout's Random Forest (200 trees), we have taken runtime measurements for different map configurations, from 1 mapper to 200. Fig. 4 depicts how time performance varies as resources augment.

The plot depicts an expected downtrend as more CPU power is available. However, some rare spikes are present around the 100-cores value. It seems that Mahout's Random Forest prefers an amount of cores divisible between the number of trees, in this case, only 100 and 200. The algorithm shows some rare deficiencies when dealing with values close to 100 (e.g.: 80 and 120) which occur during the tree construction in the map phase. This strange behavior may be explained by some design failures derived from the preliminary design implemented in Mahout's Random Forest.

Concerning accuracy, an increment of map partitions is followed by a drop on this measure as reflected in Fig. 5. This is mainly due to subtrees usually have a narrower view of input space when more map partitions are present or, in other words, the learning stage for the trees suffers from the problem of lack of data [86].

5.3. Approximate and exact fusion: k-Means and random forest on ML-Spark

As introduced in Section 3.4, Spark is mainly formed by exact implementations. Concretely, k-Means and Random Forest are two great illustrative examples in MLlib. Scalability experiments on k-Means and Random Forest have been performed in order to assess their scalability power. For these experiments, Apache Spark 1.6.2 is used as reference platform, and the Higgs dataset is previously partitioned into 216 partitions (= number of cores). Default parameter values are set for

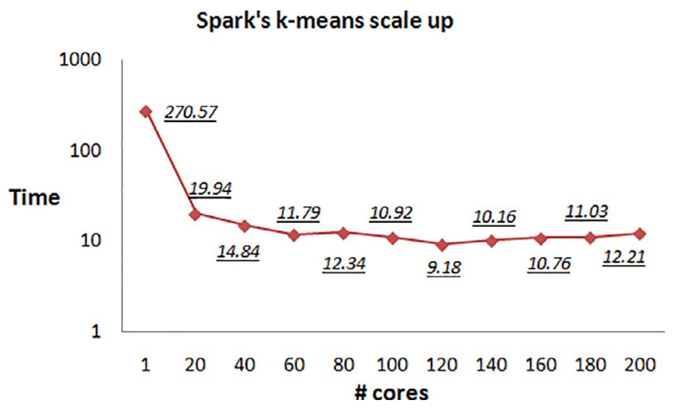


Fig. 6. Average time (3 runnings) obtained by Spark's k-Means varying the number of cores. Label values are displayed close to the points. Y-axis in 10-log scale.

both algorithms.

Notice that scalability evaluation implemented here is slightly different than that of used in Hadoop. In contrast to manual partitioning in Hadoop, Spark allows users to specify the amount of executor cores available for each program running. This feature is much more interesting since it allows to tune resource allocation without altering the original partitioning scheme.

Fig. 6 depicts how the k-Means implementation scales-up by augmenting CPU power. From 1 to 120, the method shows an expected downtrend as more resources are provided, being the minimum stuck at 120 (9.18). The remaining values follow a smooth uptrend until 200 cores.

Fig. 7 illustrates the same study for Random Forest. The results shows a similar trend to that of the case of k-Means, leading to the same conclusions. The global minimum can also be found in 120 cores. The only noticeable difference is the lower time cost held by Random Forest compared with the clustering technique.

In both cases (kNN and Random Forest), we observe a rapid reduce of time from 1 to 20 cores, and a barely stable behavior after 20 cores. A possible explanation for this is the trade-off between the penalty associated with the distributed computation versus the increment due to the parallelization itself. This issue is mainly shown in the case of those datasets with an average size with respect to the computing capacity. Furthermore, we observe that in these experiments the absolute elapsed time is low (about ten seconds), implying a threshold for a larger efficiency improvement.

In addition to the former, we observe that no real scale-up happens from 120 cores. To understand this behavior, we must cite the Spark's authors suggestion [67] for partition tuning, which recommends to prepare 2–4 × partitions per core. A partition rate below 2 × implies that some CPU cores may become idle as the processing granularity

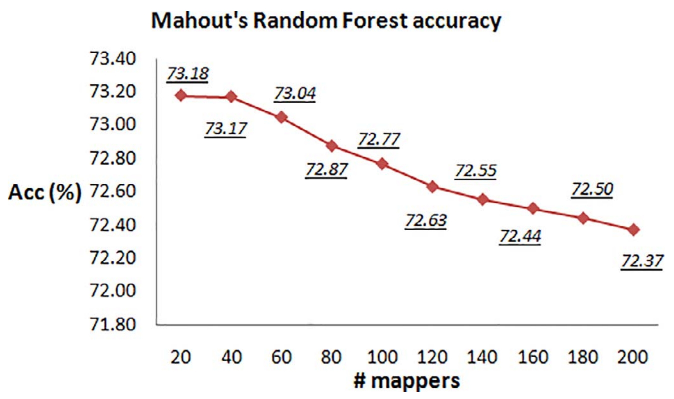


Fig. 5. Average accuracy (5-fold) obtained by Mahout's Random Forest (200 trees) varying the number of mappers. Label values are displayed close to the points.

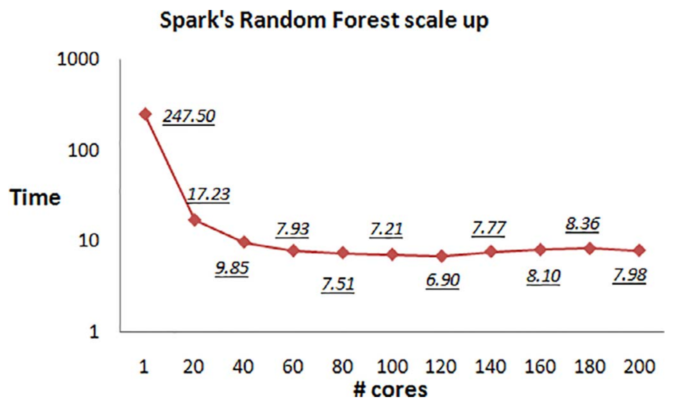


Fig. 7. Average time (3 runnings) obtained by Spark's Random Forest varying the number of cores. Label values are displayed close to the points. Y-axis in 10-log scale.

becomes small. Imagine the 1× case where the most rapid thread must wait the slowest one. A rate above 4× implies a oversized granularity where time usage is mainly dominated by startup overheads and short processes. In our study the closest value to a 2× partition rate (120-cores) is ranked as the best option.

Finally, we must point out that in this case an study on accuracy is not necessary for k-Means and Random Forest since both models provides the same predictive solution regardless of the partition scheme used. For exact fusion models, this claim is always true because all the methods implement the same intrinsic strategy, however, approximate models may vary their output depending on the division scheme followed.

6. Discussion and guidelines

The core objective in this paper was to acknowledge different paradigms and strategies implemented by modern large-scale learning algorithms. Their behavior was also analyzed into detail. To do so, we selected the level of discrepancy between the distributed models and their corresponding single-machine versions. We considered it as the most remarkable aspect to categorize the existing solutions for large-scale Machine Learning.

From this perspective, we identified two unlike groups: (1) approximative fusion of models (one submodel per partition, eventually fused), and (2) exact fusion for scalable models (compounding model with the same output as the sequential version). Other relevant aspects considered in this model categorization were their scope (local vs. global), iteration nature (1-step vs. multistage), or possible guidance by a master thread (guided vs. unguided).

Approximate models present some advantages with respect to approximate fusers such as being usually faster, especially as the number of partitions is increased. Furthermore, any existing model can be embedded into such scheme, focusing the efforts of the design into the Reduce stage. The main drawback of approximate solutions is the loss of accuracy as the number of partitions increases, since there is a clear lack of data for training. On the contrary, exact models are expected to achieve greater accuracy, yet they require more effort in their design (in order to meet the correctness condition).

Regarding these issues, we may provide some tips or suggestions in the development of distributed analytics models for Big Data. These can be considered for researchers in order to go one step further on the topic.

- The main point is to focus on the development and adoption of global and exact parallel techniques in MapReduce, Spark and/or Flink technologies. One clear example is the PLANET methodology, which allowed decision trees to have a global learning scope. This way, a robust an scalable approach that is independent on the number of processes / data partitions can be obtained.
- There is a necessity in developing more theoretical studies to facilitate the migration of current Machine Learning models towards Big Data. This way, a direct connection between a certain learning methodology and its distributed design can be established.
- Approximate models based on the traditional MapReduce scheme may still offer many interesting opportunities for research. In particular, the case of ensemble learning is of extreme importance in this scenario. The coordination among the different submodels must be a priority for a thorough learning of the problem space, and therefore to boost the performance of the final system.
- Finally, a smart use on the distributed operators for Scala is mandatory in order to implement robust and scalable solutions for the fusion process from a practical point of view.

7. Concluding remarks

In this paper, we have focused on the context of Big Data analytics and, in particular, on the design of Machine Learning algorithms following the MR programming model where the processes fusion is the core in the design. This distributed paradigm is based on parallelizing the computation among nodes, each of which is devoted to a subset of the main data. Then, the local learned models must be somehow fused in order to output a single approach.

We have proposed a taxonomy of Big Data distributed models for Machine Learning based on both the fusion tactic and the model scope, distinguishing between two main categories. On the one hand, approximate methods that carry out a direct fusion of models. On the other hand, those that provide an exact fusion of models. To obtain well founded conclusions about these different types of methodologies, we have carried out an experimental study to contrast the scalability of the different schemes. Our results have determined the higher quality of those algorithms based on exact fusion of models. In addition, we have observed the best option to a 2× partition rate for Spark-based Machine Learning implementations.

Finally, we have carried out a discussion on the main findings extracted throughout this research work. Specifically, we have analyzed with relative detail the strong and weak points for both types of the fusion models. This have allowed us to provide several guidelines for future study on the topic, namely to strengthen the development of process fusion searching for a robust design of exact parallel techniques for analytics.

Acknowledgments

This work have been partially supported by FEDER funds; the Spanish Ministry of Science and Technology under projects TIN2014-57251-P and TIN2015-68454-R; and the Foundation BBVA project 75/2016 BigDaPTOOLS.

References

[1] H. Chen, R.H.L. Chiang, V.C. Storey, Business intelligence and analytics: from big data to big impact, MIS Q. 36 (4) (2012) 1165–1188.

[2] M. Minelli, M. Chambers, A. Dhiraj, Big Data, Big Analytics: Emerging Business Intelligence and Analytic Trends for Today's Businesses, 1st ed., Wiley, 2013.

[3] A. Fernández, S. Río, V. López, A. Bawakid, M.J. del Jesus, J. Benítez, F. Herrera, Big data with cloud computing: an insight on the computing environment, mapreduce and programming framework, WIREs Data Min. Knowl. Discovery 4 (5) (2014) 380–409.

[4] C.P. Chen, C.-Y. Zhang, Data-intensive applications, challenges, techniques and technologies: a survey on big data, Inf. Sci. 275 (2014) 314–347.

[5] K. Hwang, M. Chen, Big Data Analytics for Cloud, IoT and Cognitive Computing, 1st ed., Wiley, 2017.

[6] D. Davis, BIG DATA and DATA ANALYTICS: The Beginner's Guide to Understanding the Analytical World. CreateSpace Independent Publishing Platform, 2017.

[7] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of things: a survey on enabling technologies, protocols, and applications, IEEE Commun. Surv. Tutorials 17 (4) (2015) 2347–2376.

[8] G.B. Orgaz, J.J. Jung, D. Camacho, Social big data: recent achievements and new challenges, Inf. Fus. 28 (2016) 45–59.

[9] D. Larson, V. Chang, A review and future direction of agile, business intelligence, analytics and data science, Int. J. Inf. Manage. 36 (5) (2016) 700–710.

[10] T.-M. Choi, H.K. Chan, X. Yue, Recent development in big data analytics for business operations and risk management, IEEE Trans. Cybern. 47 (1) (2017) 81–92.

[11] X. Wu, X. Zhu, G.-Q. Wu, W. Ding, Data mining with big data, Knowl. Data Eng. IEEE Trans. 26 (1) (2014) 97–107.

[12] B. Wixom, T. Ariyachandra, D. Douglas, M. Goul, B. Gupta, L. Iyer, U. Kulkarni, B.J.G. Mooney, G. Phillips-Wren, O. Turetken, The current state of business intelligence in academia: the arrival of big data, Commun. Assoc. Inf. Syst. 34 (1) (2014) 1–13.

[13] A. Abbasi, S. Sarker, R.H.L. Chiang, Big data research in information systems: toward an inclusive research agenda, J. Assoc. Inf. Syst. 17 (2) (2016) 1–32.

[14] M. Chen, Welcome to the new interdisciplinary journal combining big data and cognitive computing, Big Data Cognit. Comput. 1 (1) (2016). 1–1.

[15] A. Gandomi, M. Haider, Beyond the hype: big data concepts, methods, and analytics, Int. J. Inf. Manage. 35 (2) (2015) 137–144.

[16] H. Hu, Y. Wen, T.. Chua, X. Li, Toward scalable systems for big data analytics: a technology tutorial, IEEE Access 2 (2014) 652–687.

[17] M. Chen, S. Mao, Y. Zhang, V.C.M. Leung, Big Data - Related Technologies, Challenges and Future Prospects, Springer briefs in computer science, Springer, 2014.

[18] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters,

Commun. ACM 51 (1) (2008) 107–113.
[19] J. Dean, S. Ghemawat, MapReduce: a flexible data processing tool, Commun. ACM 53 (1) (2010) 72–77.
[20] K.H. Lee, Y.J. Lee, H. Choi, Y.D. Chung, B. Moon, Parallel data processing with mapreduce: a survey, SIGMOD Record 40 (4) (2011) 11–20.
[21] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The hadoop distributed file system, 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST2010, (2010).
[22] B. Krawczyk, L.L. Minku, J. Gama, J. Stefanowski, M. Woniak, Ensemble learning for data stream analysis: a survey, Inf. Fus. 37 (2017) 132–156.
[23] S. Owen, R. Anil, T. Dunning, E. Friedman, Mahout in Action, 1st ed., Manning Publications Co., 2011.
[24] The Apache Software Foundation, Mahout, an open source project which includes scalable machine learning algorithms, (2017).
[25] D. Lyubimov, A. Palumbo, Apache Mahout: Beyond MapReduce, 1st ed., CreateSpace Independent, 2016.
[26] C. Lam, Hadoop in Action, 1st ed., Manning, 2011.
[27] T. White, Hadoop: The Definitive Guide, 4th ed., O'Reilly Media, 2015.
[28] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M.J. Franklin, R. Zadeh, M. Zaharia, A. Talwalkar, Mllib: machine learning in apache spark, J. Mach. Learn. Res. 17 (34) (2016) 1–7.
[29] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: cluster computing with working sets, HotCloud 2010, (2010), pp. 1–7.
[30] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing, Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, NSDI'12, (2012). 2–2.
[31] C.-W. Tsai, C.-F. Lai, M.-C. Chiang, L.T. Yang, Data mining for internet of things: a survey, IEEE Commun. Surv. Tut. 16 (1) (2014) 77–97.
[32] D. Galpert, S. Río, F. Herrera, E. Ancede-Gallardo, A. Antunes, G. Agero-Chapin, An effective big data supervised imbalanced classification approach for ortholog detection in related yeast species, Biomed. Res. Int. 2015 (2015) 748681:1–748681:12.
[33] M. Chen, Y. Hao, K. Hwang, L. Wang, L. Wang, Disease prediction by machine learning over big data from healthcare communities. IEEE Access 5 (2017) 8869–8879.
[34] J.A. Balazs, J.D. Velásquez, Opinion mining and information fusion: a survey, Inf. Fus. 27 (2016) 95–110.
[35] S. Sun, C. Luo, J. Chen, A review of natural language processing techniques for opinion mining systems, Inf. Fus. 36 (2017) 10–25.
[36] Q. Zhang, B. Wu, J. Yang, Parallelization of ontology construction and fusion based on mapreduce, 2014 IEEE 3rd International Conference on Cloud Computing and Intelligence Systems, (2014), pp. 439–443.
[37] J. Meng, R. Li, J. Zhang, Parallel information fusion method for microarray data analysis, 2015 IEEE International Conference on Big Data (Big Data), (2015), pp. 1539–1544.
[38] S. del Río, V. López, J. Benítez, F. Herrera, On the use of mapreduce for imbalanced big data using random forest, Inf. Sci. (Ny) 285 (2014) 112–137.
[39] S. del Río, V. López, J. Benítez, F. Herrera, A mapreduce approach to address big data classification problems based on the fusion of linguistic fuzzy rules, Int. J. Comput. Intell. Syst. 8 (3) (2015) 422–437.
[40] M. Sung, SIMD parallel processing michael sung 6.911: architectures anonymous, 2000.
[41] A.J. Zaman Khan RZ, Use of DAG in distributed parallel computing, Int. J. Appl. Innov. Eng. Manage. 2 (11) (2013) 81–85.
[42] L.G. Valiant, A bridging model for parallel computation, Commun. ACM 33 (8) (1990) 103–111.
[43] J. Dean, S. Ghemawat, Mapreduce: Simplified data processing on large clusters, In Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI), (2004), pp. 137–150.
[44] D. Harris, The history of Hadoop: from 4 nodes to the future of data, 2013, (https://gigaom.com/2013/03/04/the-history-of-hadoop-from-4-nodes-to-the-future-of-data/).
[45] A.S. Foundation, Apache project directory, 2017, (https://projects.apache.org/).
[46] H.D.F. System, Hadoop distributed file system, 2017, (https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html).
[47] A. Tez, Apache tez, 2017, (https://tez.apache.org/).
[48] A. YARN, Apache YARN, 2017, (https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html).
[49] A. Mahout, Apache mahout, 2017, (https://mahout.apache.org/).
[50] A. Spark, Apache spark: lightning-fast cluster computing, 2017, (http://spark.apache.org/).
[51] M. Hamstra, H. Karau, M. Zaharia, A. Konwinski, P. Wendell, Learning Spark: Lightning-Fast Big Data Analytics, O'Reilly Media, Incorporated, 2015.
[52] S. Packages, 3rd party spark packages, 2017, (https://spark-packages.org/).
[53] B. P, J.S. Herbach, S. Basu, R.J. Bayardo, G. Inc, PLANET: massively parallel learning of tree ensembles with mapreduce, PVLDB (2009) 1426–1437.
[54] A. Flink, Apache flink, 2017, (http://flink.apache.org/).
[55] Apache Flink Project, Peeking into Apache flink's engine room, 2017, (https://flink.apache.org/news/2015/03/13/peeking-into-Apache-Flinks-Engine-Room.html).
[56] S. Ewen, K. Tzoumas, M. Kaufmann, V. Markl, Spinning fast iterative data flows, PVLDB 5 (11) (2012) 1268–1279.
[57] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinlnder, M.J. Sax, S. Schelter, M. Hger, K. Tzoumas, D. Warneke, The stratosphere platform for big data analytics, Int. J. Very Large Databases 23 (6) (2014) 939–964.
[58] F. Hueske, M. Peters, M. Sax, A. Rheinlnder, R. Bergmann, A. Krettek, K. Tzoumas, Opening the black boxes in data flow optimization, PVLDB 5 (2012) 1256–1267.
[59] M. Jaggi, V. Smith, M. Takác, J. Terhorst, S. Krishnan, T. Hofmann, M.I. Jordan, Communication-efficient distributed dual coordinate ascent, CoRR abs/1409.1458 (2014).
[60] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, G. Varoquaux, API design for machine learning software: experiences from the scikit-learn project, ECML PKDD Workshop: Languages for Data Mining and Machine Learning, (2013), pp. 108–122.
[61] D. García-Gil, S. Ramírez-Gallego, S. García, F. Herrera, A comparison on scalability for batch big data processing on apache spark and apache flink, Big Data Anal. 2 (1) (2017) 1, http://dx.doi.org/10.1186/s41044-016-0020-2.
[62] S. del Río, V. López, J.M. Benítez, F. Herrera, A mapreduce approach to address big data classification problems based on the fusion of linguistic fuzzy rules. Int. J. Comput. Intell. Syst. 8 (3) (2015) 422–437.
[63] V. López, S. Río, J.M. Benítez, F. Herrera, Cost-sensitive linguistic fuzzy rule based classification systems under the MapReduce framework for imbalanced big data, Fuzzy Sets Syst. 258 (2015) 5–38.
[64] I. Palit, C.K. Reddy, Scalable and parallel boosting with mapreduce, IEEE Trans. Knowl. Data Eng. 24 (10) (2012) 1904–1916.
[65] I. Triguero, D. Peralta, J. Bacardit, S. García, F. Herrera, Mrpr: a mapreduce solution for prototype reduction in big data classification. Neurocomputing 150 (2015) 331–345.
[66] D. Blog, Random forests and boosting in MLlib, 2017, (https://databricks.com/blog/2015/01/21/random-forests-and-boosting-in-mllib.html).
[67] A. Spark, Machine learning library (MLlib) guide, 2017, (https://spark.apache.org/docs/latest/ml-guide.html).
[68] W. Zhao, H. Ma, Q. He, Parallel k-means clustering based on mapreduce, CloudCom 2009, Lecture notes in computer science 5931 (2009), pp. 674–679.
[69] J. Maillo, S. Ramírez-Gallego, I. Triguero, F. Herrera, Knn-is: an iterative spark-based design of the k-nearest neighbors classifier for big data. Knowl. Based Syst. 117 (2017) 3–15.
[70] S. Ramírez-Gallego, I. Lastra, D. Martínez-Rego, V. Bolón-Canedo, J.M. Benítez, F. Herrera, A. Alonso-Betanzos, Fast-mrmr: fast minimum redundancy maximum relevance algorithm for high-dimensional big data. Int. J. Intell. Syst. 32 (2) (2017) 134–152.
[71] R. Polikar, Ensemble based systems in decision making, IEEE Circuits Syst. Mag. 6 (3) (2006) 21–45.
[72] J. Assuncao, P. Fernandes, L. Lopes, S. Normey, Distributed stochastic aware random forests - efficient data mining for big data, Proceedings - 2013 IEEE International Congress on Big Data, BigData 2013, (2013), pp. 425–426.
[73] Y. Wang, W. Goh, L. Wong, G. Montana, Random forests on hadoop for genome-wide association studies of multivariate neuroimaging phenotypes, BMC Bioinfor. 14 (Suppl 16) (2013).
[74] L. Rokach, Decision forest: twenty years of research, Inf. Fus. 27 (2016) 111–125.
[75] R.E. Schapire, A brief introduction to boosting, IJCAI, (1999), pp. 1401–1406.
[76] R.E. Schapire, Y. Singer, Improved boosting algorithms using confidence-rated predictions, Mach. Learn. 37 (3) (1999) 297–336.
[77] I. Triguero, J. Derrac, S. García, F. Herrera, A taxonomy and experimental study on prototype generation for nearest neighbor classification. IEEE Trans. Syst. Man Cybern. Part C 42 (1) (2012) 86–100.
[78] A. Fernandez, C. Carmona, M. del Jesus, F. Herrera, A view on fuzzy systems for big data: progress and opportunities, Int. J. Comput. Intell. Systems 9 (1) (2016) 69–80.
[79] L.I. Kuncheva, Diversity in multiple classifier systems, Inf. Fus. 6 (1) (2005) 3–4.
[80] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition, 2nd ed., Springer series in statistics, Springer, 2011.
[81] D. Blog, Scalable decision trees in MLlib, 2017, (https://databricks.com/blog/2014/09/29/scalable-decision-trees-in-mllib.html).
[82] D.C. Liu, J. Nocedal, On the limited memory BFGS method for large scale optimization, Math. Program. 45 (3) (1989) 503–528.
[83] S. Ramírez-Gallego, H.M. no Talín, D. Martínez-Rego, V. Bolón-Canedo, J. Benítez, A. Alonso-Betanzos, F. Herrera, An information theoretic feature selection framework for big data under apache spark, IEEE Trans. Syst. Man Cybern. in press (2017), http://dx.doi.org/10.1109/TSMC.2017.2670926.
[84] P. Baldi, P. Sadowski, D. Whiteson, Searching for exotic particles in high-Energy physics with deep learning, Nat. Commun. 5 (2014) 4308.
[85] M. Lichman, UCI machine learning repository, 2013.
[86] A. Fernandez, S. Rio, N. Chawla, F. Herrera, An insight into imbalanced big data classification: outcomes and challenges, Complex Intell. Syst. 3 (2) (2017) 105–120.

1.2 Big data preprocessing: methods and prospects

- S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez and F. Herrera, Big data preprocessing: methods and prospects. Big Data Analytics 1 (2016) 1–9, doi: 10.1186/s41044-016-0014-0
  - Status: **Published**.
  - Impact Factor (JCR 2016): Not indexed.

REVIEW

Open Access

Big data preprocessing: methods and prospects

Salvador García*, Sergio Ramírez-Gallego, Julián Luengo, José Manuel Benítez and Francisco Herrera

*Correspondence:
salvagl@decsai.ugr.es
Department of Computer Science and Artificial Intelligence, University of Granada, CITIC-UGR, 18071 Granada, Spain

Abstract

The massive growth in the scale of data has been observed in recent years being a key factor of the Big Data scenario. Big Data can be defined as high volume, velocity and variety of data that require a new high-performance processing. Addressing big data is a challenging and time-demanding task that requires a large computational infrastructure to ensure successful data processing and analysis. The presence of data preprocessing methods for data mining in big data is reviewed in this paper. The definition, characteristics, and categorization of data preprocessing approaches in big data are introduced. The connection between big data and data preprocessing throughout all families of methods and big data technologies are also examined, including a review of the state-of-the-art. In addition, research challenges are discussed, with focus on developments on different big data framework, such as Hadoop, Spark and Flink and the encouragement in devoting substantial research efforts in some families of data preprocessing methods and applications on new big data learning paradigms.

Keywords: Big data, Data mining, Data preprocessing, Hadoop, Spark, Imperfect data, Data transformation, Feature selection, Instance reduction

Background

Vast amounts of raw data is surrounding us in our world, data that cannot be directly treated by humans or manual applications. Technologies as the World Wide Web, engineering and science applications and networks, business services and many more generate data in exponential growth thanks to the development of powerful storage and connection tools. Organized knowledge and information cannot be easily obtained due to this huge data growth and neither it can be easily understood or automatically extracted. These premises have led to the development of data science or data mining [1], a well-known discipline which is more and more present in the current world of the Information Age.

Nowadays, the current volume of data managed by our systems have surpassed the processing capacity of traditional systems [2], and this applies to data mining as well. The arising of new technologies and services (like Cloud computing) as well as the reduction in hardware price are leading to an ever-growing rate of information on the Internet. This phenomenon certainly represents a “Big” challenge for the data analytics community. Big Data can be thus defined as very high volume, velocity and variety of data that require a new high-performance processing [3].

Distributed computing has been widely used by data scientists before the advent of Big Data phenomenon. Many standard and time-consuming algorithms were replaced by their distributed versions with the aim of agilizing the learning process. However, for most of current massive problems, a distributed approach becomes mandatory nowadays since no batch architecture is able to tackle these huge problems.

Many platforms for large-scale processing have tried to face the problematic of Big Data in last years [4]. These platforms try to bring closer the distributed technologies to the standard user (enginners and data scientists) by hiding the technical nuances derived from distributed environments. Complex designs are required to create and maintain these platforms, which generalizes the use of distributed computing. On the other hand, Big Data platforms also requires additional algorithms that give support to relevant tasks, like big data preprocessing and analytics. Standard algorithms for those tasks must be also re-designed (sometimes, entirely) if we want to learn from large-scale datasets. It is not trivial thing and presents a big challenge for researchers.

The first framework that enabled the processing of large-scale datasets was MapReduce [5] (in 2003). This revolutionary tool was intended to process and generate huge datasets in an automatic and distributed way. By implementing two primitives, Map and Reduce, the user is able to use a scalable and distributed tool without worrying about technical nuances, such as: failure recovery, data partitioning or job communication. Apache Hadoop [6, 7] emerged as the most popular open-source implementation of MapReduce, maintaining the aforementioned features. In spite of its great popularity, MapReduce (and Hadoop) is not designed to scale well when dealing with iterative and online processes, typical in machine learning and stream analytics [8].

Apache Spark [9, 10] was designed as an alternative to Hadoop, capable of performing faster distributed computing by using in-memory primitives. Thanks to its ability of loading data into memory and re-using it repeatedly, this tool overcomes the problem of iterative and online processing presented by MapReduce. Additionally, Spark is a general-purpose framework that thanks to its generality allows to implement several distributed programming models on top of it (like Pregel or HaLoop) [11]. Spark is built on top of a new abstraction model called Resilient Distributed Datasets (RDDs). This versatile model allows controlling the persistence and managing the partitioning of data, among other features.

Some competitors to Apache Spark have emerged lastly, especially from the streaming side [12]. Apache Storm [13] is an open-source distributed real-time processing platform, which is capable of processing millions of tuples per second and node in a fault-tolerant way. Apache Flink [14] is a recent top-level Apache project designed for distributed stream and batch data processing. Both alternatives try to fill the "online" gap left by Spark, which employs a mini-batch streaming processing instead of a pure streaming approach.

The performance and quality of the knowledge extracted by a data mining method in any framework does not only depends on the design and performance of the method but is also very dependent on the quality and suitability of such data. Unfortunately, negative factors as noise, missing values, inconsistent and superfluous data and huge sizes in examples and features highly influence the data used to learn and extract knowledge. It is well-known that low quality data will lead to low quality knowledge [15]. Thus data preprocessing [16] is a major and essential stage whose main goal is to obtain final data sets which can be considered correct and useful for further data mining algorithms.

Big Data also suffer of the aforementioned negative factors. Big Data preprocessing constitutes a challenging task, as the previous existent approaches cannot be directly applied as the size of the data sets or data streams make them unfeasible. In this overview we gather the most recent proposals in data preprocessing for Big Data, providing a snapshot of the current state-of-the-art. Besides, we discuss the main challenges on developments in data preprocessing for big data frameworks, as well as technologies and new learning paradigms where they could be successfully applied.

Data preprocessing

The set of techniques used prior to the application of a data mining method is named as data preprocessing for data mining [16] and it is known to be one of the most meaningful issues within the famous Knowledge Discovery from Data process [17, 18] as shown in Fig. 1. Since data will likely be imperfect, containing inconsistencies and redundancies is not directly applicable for a starting a data mining process. We must also mention the fast growing of data generation rates and their size in business, industrial, academic and science applications. The bigger amounts of data collected require more sophisticated mechanisms to analyze it. Data preprocessing is able to adapt the data to the requirements posed by each data mining algorithm, enabling to process data that would be unfeasible otherwise.

Albeit data preprocessing is a powerful tool that can enable the user to treat and process complex data, it may consume large amounts of processing time [15]. It includes a wide range of disciplines, as data preparation and data reduction techniques as can be seen in Fig. 2. The former includes data transformation, integration, cleaning and normalization; while the latter aims to reduce the complexity of the data by feature selection, instance

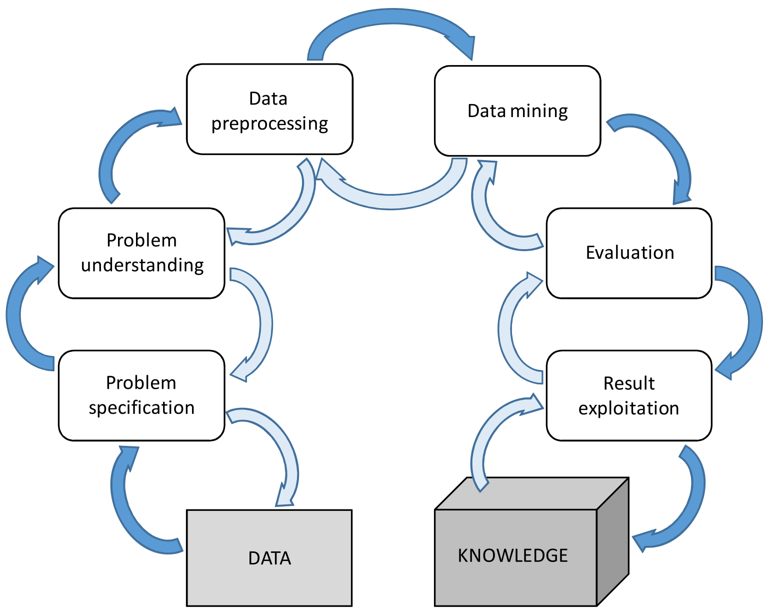


Fig. 1 KDD process

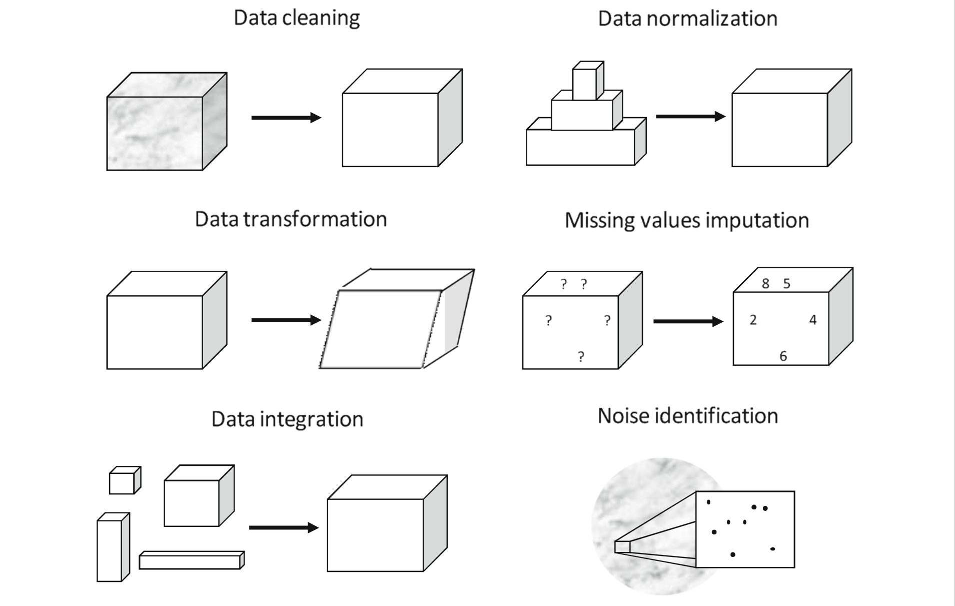


Fig. 2 Data preprocessing tasks

selection or by discretization (see Fig. 3). After the application of a successful data preprocessing stage, the final data set obtained can be regarded as a reliable and suitable source for any data mining algorithm applied afterwards.

Data preprocessing is not only limited to classical data mining tasks, as classification or regression. More and more researchers in novel data mining fields are paying increasingly attention to data data preprocessing as a tool to improve their models. This wider adoption of data preprocessing techniques is resulting in adaptations of known models for related frameworks, or completely novel proposals.

In the following we will present the main fields of data preprocessing, grouping them by their types and showing the current open challenges relative to each one. First, we will tackle the preprocessing techniques to deal with imperfect data, where missing values and noise data are included. Next, data reduction preprocessing approaches will be presented, in which feature selection and space transformation are shown. The following section will deal with instance reduction algorithms, including instance selection and prototype generation. The last three section will be devoted to discretization, resampling for imbalanced problems and data preprocessing in new fields of data mining respectively.

Imperfect data

Most techniques in data mining rely on a data set that is supposedly complete or noise-free. However, real-world data is far from being clean or complete. In data preprocessing it is common to employ techniques to either removing the noisy data or to impute (fill in) the missing data. The following two sections are devoted two missing values imputation and noise filtering.

Missing values imputation

One big assumption made by data mining techniques is that the data set is complete. The presence of missing values is, however, very common in the acquisition processes. A

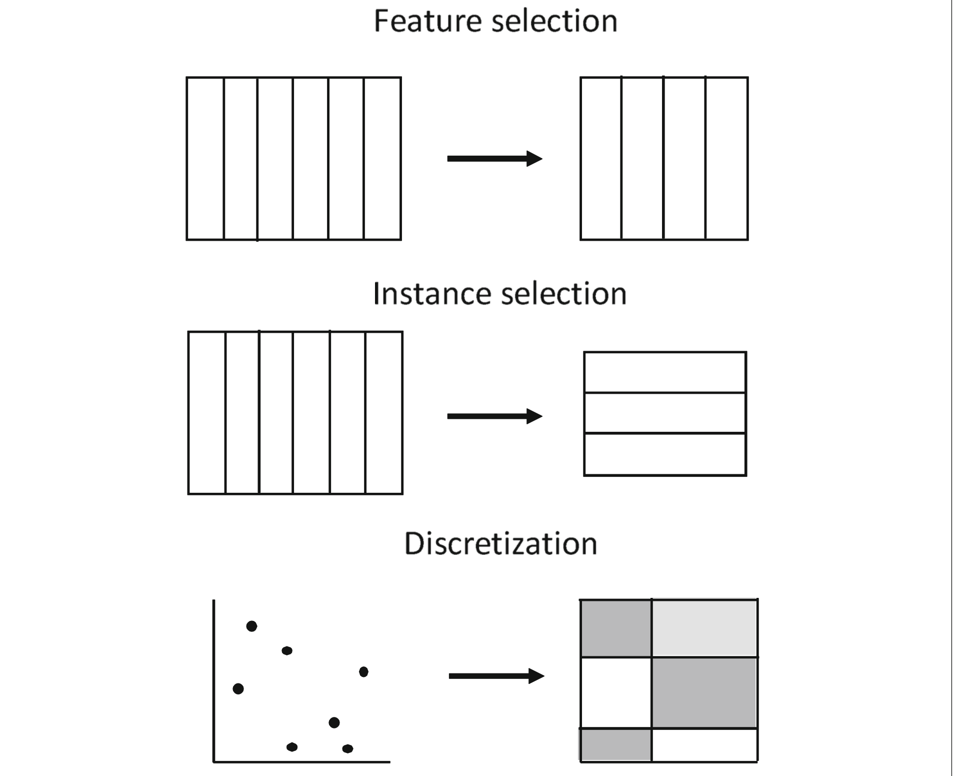


Fig. 3 Data reduction approaches

missing value is a datum that has not been stored or gathered due to a faulty sampling process, cost restrictions or limitations in the acquisition process. Missing values cannot be avoided in data analysis, and they tend to create severe difficulties for practitioners.

Missing values treatment is difficult. Inappropriately handling the missing values will easily lead to poor knowledge extracted and also wrong conclusions [19]. Missing values have been reported to cause loss of efficiency in the knowledge extraction process, strong biases if the missingness introduction mechanism is mishandled and severe complications in data handling.

Many approaches are available to tackle the problematic imposed by the missing values in data preprocessing [20]. The first option is usually to discard those instances that may contain a missing value. However, this approach is rarely beneficial, as eliminating instances may produce a bias in the learning process, and important information can be discarded [21]. The seminal works on data imputation come from statistics. They model the probability functions of the data and take into account the mechanisms that induce missingness. By using maximum likelihood procedures, they sample the approximate probabilistic models to fill the missing values. Since the true probability model for a particular data sets is usually unknown, the usage of machine learning techniques has become very popular nowadays as they can be applied avoiding without providing any prior information.

Noise treatment

Data mining algorithms tend to assume that any data set is a sample of an underlying distribution with no disturbances. As we have seen in the previous section, data gathering is rarely perfect, and corruptions often appear. Since the quality of the results obtained by a data mining technique is dependent on the quality of the data, tackling the problem of noise data is mandatory [22]. In supervised problems, noise can affect the input features, the output values or both. When noise is present in the input attributes, it is usually referred as *attribute noise*. The worse case is when the noise affects the output attribute, as this means that the bias introduced will be greater. As this kind of noise has been deeply studied in classification, it is usually known as *class noise*.

In order to treat noise in data mining, two main approaches are commonly used in the data preprocessing literature. The first one is to correct the noise by using *data polishing methods*, specially if it affects the labeling of an instance. Even partial noise correction is claimed to be beneficial [23], but it is a difficult task and usually limited to small amounts of noise. The second is to use *noise filters*, which identify and remove the noisy instances in the training data and do no require the data mining technique to be modified.

Dimensionality reduction

When data sets become large in the number of predictor variables or the number of instances, data mining algorithms face the *curse of dimensionality* problem [24]. It is a serious problem as it will impede the operation of most data mining algorithms as the computational cost rise. This section will underline the most influential dimensionality reduction algorithms according to the division established into Feature Selection (FS) and space transformation based methods.

Feature selection

Feature selection (FS) is "the process of identifying and removing as much irrelevant and redundant information as possible" [25]. The goal is to obtain a subset of features from the original problem that still appropriately describe it. This subset is commonly used to train a learner, with added benefits reported in the specialized literature [26, 27]. FS can remove irrelevant and redundant features which may induce accidental correlations in learning algorithms, diminishing their generalization abilities. The use of FS is also known to decrease the risk of over-fitting in the algorithms used later. FS will also reduce the search space determined by the features, thus making the learning process faster and also less memory consuming.

The use FS can also help in task not directly related to the data mining algorithm applied to the data. FS can be used in the data collection stage, saving cost in time, sampling, sensing and personnel used to gather the data. Models and visualizations made from data with fewer features will be easier to understand and to interpret.

Space transformations

FS is not the only way to cope with the curse of dimensionality by reducing the number of dimensions. Instead of selecting the most promising features, space transformation techniques generate a whole new set of features by combining the original ones. Such a combination can be made obeying different criteria. The first approaches were based on linear methods, as factor analysis [28] and PCA [29].

More recent techniques try to exploit nonlinear relations among the variables. Some of the most important, both in relevance and usage, space transformation procedures are LLE [30], ISOMAP [31] and derivatives. They focus on transforming the original set of variables into a smaller number of projections, sometimes taking into account the geometrical properties of clusters of instances or patches of the underlying manifolds.

Instance reduction

A popular approach to minimize the impact of very large data sets in data mining algorithms is the use of Instance Reduction (IR) techniques. They reduce the size of the data set without decreasing the quality of the knowledge that can be extracted from it. Instance reduction is a complementary task regarding FS. It reduces the quantity of data by removing instances or by generating new ones. In the following we describe the most important instance reduction and generation algorithms.

Instance selection

Nowadays, instance selection is perceived as necessary [32]. The main problem in instance selection is to identify suitable examples from a very large amount of instances and then prepare them as input for a data mining algorithm. Thus, instance selection is comprised by a series of techniques that must be able to choose a subset of data that can replace the original data set and also being able to fulfill the goal of a data mining application [33, 34]. It must be distinguished between instance selection, which implies a smart operation of instance categorization, from data sampling, which constitutes a more randomized approach [16].

A successful application of instance selection will produce a minimum data subset that it is independent from the data mining algorithm used afterwards, without losing performance. Other added benefits of instance selection is to remove noisy and redundant instances (*cleaning*), to allow data mining algorithms to operate with large data sets (*enabling*) and to focus on the important part of the data (*focusing*).

Instance generation

Instance selection methods concern the identification of an optimal subset of representative objects from the original training data by discarding noisy and redundant examples. Instance generation methods, by contrast, besides selecting data, can generate and replace the original data with new artificial data. This process allows it to fill regions in the domain of the problem, which have no representative examples in original data, or to condensate large amounts of instances in less examples. Instance generation methods are often called prototype generation methods, as the artificial examples created tend to act as a representative of a region or a subset of the original instances [35].

The new prototypes may be generated following diverse criteria. The simplest approach is to relabel some examples, for example those that are suspicious of belonging to a wrong class label. Some prototype generation methods create centroids by merging similar examples, or by first merging the feature space in several regions and then creating a set of prototype for each one. Others adjust the position of the prototypes through the space, by adding or substracting values to the prototype's features.

Discretization

Data mining algorithms require to know the domain and type of the data that will be used as input. The type of such data may vary, from categorical where no order among the values can be established, to numerical data where the order among the values there exist.

Decision trees, for instance, make split based on information or separability measures that require categorical values in most cases. If continuous data is present, the discretization of the numerical features is mandatory, either prior to the tree induction or during its building process.

Discretization is gaining more and more consideration in the scientific community [36] and it is one of the most used data preprocessing techniques. It transforms quantitative data into qualitative data by dividing the numerical features into a limited number of non-overlapped intervals. Using the boundaries generated, each numerical value is mapped to each interval, thus becoming discrete. Any data mining algorithm that needs nominal data can benefit from discretization methods, since many real-world applications usually produce real valued outputs. For example, three of the ten methods considered as the top ten in data mining [37] need an external or embedded discretization of data: C4.5 [38], Apriori [39] and Naïve Bayes [40] In these cases, discretization is a crucial previous stage.

Discretization also produce added benefits. The first is data simplification and reduction, helping to produce a faster and more accurate learning. The second is readability, as discrete attributes are usually easier to understand, use and explain [36]. Nevertheless these benefits come at price: any discretization process is expected to generate a loss of information. Minimizing this information loss is the main goal pursued by the discretizer, but an optimal discretization is a NP-complete process. Thus, a wide range of alternatives are available in the literature as we can see in some published reviews on the topic [36, 41, 42].

Imbalanced learning. Undersampling and oversampling methods

In many supervised learning applications, there is a significant difference between the prior probabilities of different classes, i.e., between the probabilities with which an example belongs to the different classes of the classification problem. This situation is known as the class imbalance problem [43]. The hitch with imbalanced datasets is that standard classification learning algorithms are often biased towards the majority class (known as the "negative" class) and therefore there is a higher misclassification rate for the minority class instances (called the "positive" examples).

While algorithmic modifications are available for imbalanced problems, our interest lies in preprocessing techniques to alleviate the bias produced by standard data mining algorithms. These preprocessing techniques proceed by resampling the data to balance the class distribution. The main advantage is that they are independent of the data mining algorithm applied afterwards.

Two main groups can be distinguished within resampling. The first one is *undersampling methods*, which create a subset of the original dataset by eliminating (majority) instances. The second one is *oversampling methods*, which create a superset of the original dataset by replicating some instances or creating new instances from existing ones.

Non-heuristic techniques, as random-oversampling or random-undersampling were initially proposed, but they tend to discard information or induce over-fitting. Among the more sophisticated, heuristic approaches, "Synthetic Minority Oversampling TEchnique" (SMOTE) [44] has become one of the most renowned approaches in this area. It interpolates several minority class examples that lie together. Since SMOTE can still induce over-fitting in the learner, its combination with a plethora of sampling methods can be found in the specialized literature with excellent results. Under-sampling has the advantage of producing reduced data sets, and thus interesting approaches based on neighborhood methods, clustering and even evolutionary algorithms have been successfully applied to generate quality balanced training sets by discarding majority class examples.

Data preprocessing in new data mining fields

Many data preprocessing methods have been devised to work with supervised data, since the label provides useful information that facilitates data transformation. However, there are also preprocessing approaches for unsupervised problems.

For instance, FS has attracted much attention lately for unsupervised problems [45–47] or missing values imputation [48]. Semisupervised classification, which contains instances both labeled and unlabeled, also shows several works in preprocessing for discretization [49], FS [46], instance selection [50] or missing values imputation [51]. Multi-label classification is a framework prone to gather imbalanced problems. Thus, methods for re-sampling these particular data sets have been proposed [52, 53]. Multi-instance problems are also challenging, and resampling strategies have been also studied for them [54]. Data streams are also a challenging area of data mining, since the information represented may change with time. Nevertheless, data streams are attracting much attention and for instance preprocessing approaches for imputing missing values [55, 56], FS [57] and IR [58] have been recently proposed.

Big data preprocessing

This section aims at detailing a thorough list of contributions on Big Data preprocessing. Table 1 classifies these contributions according to the category of data preprocessing, number of features, number of instances, maximum data size managed by each algorithm and the framework under they have been developed. The size has been computed multiplying the total number features by the number of instances (8 bytes per datum). For sparse methods (like [59] or [60]), only the non-sparse cells have been considered. Figure 4 depicts an histogram of the methods using the size variable. It can be observed as most of methods have only been tested against datasets between zero an five gigabytes, and few approaches have been tested against truly large-scale datasets.

Once seen a snapshot of the current developments in Big Data preprocessing, we will give shorts descriptions of the contributions in the rest of this section. First, we describe one of the most popular machine learning library for Big Data: MLlib; which brings a wide range of data preprocessing techniques to the Spark community. Next the rest of sections will be devoted to enumerate those contributions presented in the literature, and categorized and arranged in the Table 1.

MLlib: a spark machine learning library

MLlib [61] is a powerful machine learning library that enables the use of Spark in the data analytics field. This library is formed by two packages:

- *mllib*: this is the first version of MLlib, which was built on top of RDDs. It contains the majority of the methods proposed up to now.
- *ml*: it comes with the newest features of MLlib for constructing ML pipelines. This higher-level API is built on enhanced DataFrames structures [62].

Table 1 Analyzed methods and the maximum data size managed by each one

Methods	Category	# Features	# Instances	Size (GB)	Framework
[70]	FS	630	65,003,913	305.1196	Hadoop MapReduce
[69]	FS	630	65,003,913	305.1196	Hadoop MapReduce
[74]	FS	1,156	5,670,000	48.8350	MPI
[60]	FS	29,890,095	19,264,097	4.1623	C++/MATLAB
[59]	FS	100,000	10,000,000	1.4901	MapReduce
[76]	FS	100	1,600,000	1.1921	Apache Spark
[80]	FS	127	1,131,571	1.0707	Hadoop MapReduce
[71]	FS	54,675	2,096	0.8538	Hadoop MapReduce
[75]	FS	54	581,012	0.2338	Hadoop MapReduce
[73]	FS	20	1,000,000	0.1490	MapReduce
[77]	FS	–	–	0.0976	Hadoop MapReduce
[79]	FS	256	38,232	0.0729	Hadoop MapReduce
[68]	FS	52	5,253	0.0020	Hadoop MapReduce
[78]	FS	–	–	0.0000	Hadoop MapReduce
[67]	FS	–	–	0.0000	Hadoop MapReduce
[72]	FS	–	–	0.0000	Hadoop MapReduce
[83]	Imbalanced	630	32,000,000	150.2037	Hadoop MapReduce
[84]	Imbalanced	630	32,000,000	150.2037	Hadoop MapReduce
[90]	Imbalanced	630	16,000,000	75.1019	Apache Spark
[89]	Imbalanced	41	4,856,151	1.4834	Hadoop MapReduce
[82]	Imbalanced	41	4,000,000	1.2219	Hadoop MapReduce
[81]	Imbalanced	14	1,432,941	0.1495	Hadoop MapReduce
[86]	Imbalanced	9,731	1,446	0.1048	Hadoop MapReduce
[91]	Imbalanced	14	524,131	0.0547	Hadoop MapReduce
[87]	Imbalanced	36	95,048	0.0255	Hadoop MapReduce
[88]	Imbalanced	8	2,687,280	0.0200	Hadoop MapReduce
[93]	Incomplete	625	4,096,000	19.0735	MapReduce (Twister)
[92]	Incomplete	481	191,779	0.6873	Hadoop MapReduce
[95]	discretization	630	65,003,913	305.1196	Apache Spark
[96]	discretization	630	65,003,913	305.1196	Apache Spark
[94]	discretization	–	–	4.0000	Hadoop MapReduce
[97]	IR	41	4,856,151	1.4834	Hadoop MapReduce

The methods are grouped by preprocessing task, and ordered by maximum data size. Those methods with no information about number of features or instances have been set to zero size

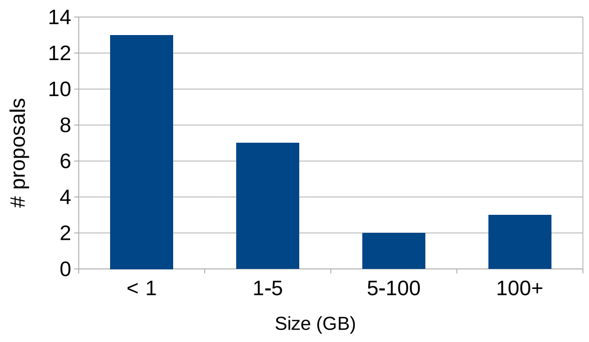


Fig. 4 Maximum data size managed by each preprocessing method (in gigabytes)

Here, we describe and classify all data preprocessing techniques for both versions[1] into five categories: discretization and normalization, feature extraction, feature selection, feature indexers and encoders, and text mining.

Discretization and normalization

Discretization transforms continuous variables using discrete intervals, whereas normalization just performs an adjustment of distributions.

- Binarizer: converts numerical features to binary features. This method makes the assumption that data follows a Bernoulli distribution. If a given feature is greater than a threshold it yields a 1.0, if not, a 0.0.
- Bucketizer: discretizes a set of continuous features by using buckets. The user specifies the number of buckets.
- Discrete Cosine Transform: transforms a real-valued sequence in the time domain into another real-valued sequence (with the same size) in the frequency domain.
- Normalizer: normalizes each row to have unit norm. It uses parameter p, which specifies the p-norm used.
- StandardScaler: normalizes each feature so that it follows a normal distribution.
- MinMaxScaler: normalizes each feature to a specific range, using two parameters: the lower and the upper bound.
- ElementwiseProduct: scales each feature by a scalar multiplier.

Feature extraction

Feature extraction techniques combine the original set of features to obtain a new set of less-redundant variables [63]. For example, by using projections to low-dimensional spaces.

- Polynomial Expansion expands the set of features into a polynomial space. This new space is formed by an n-degree combination of the original dimensions.
- VectorAssambler: combines a set of features into a single vector column.
- Single Value Decomposition (SVD) is matrix factorization method that transform a real/complex matrix M (mxn) into a factorized matrix A.
  The creators expose that for large matrices it is not needed the complete factorization but only to maintain the top-k singular values and vectors. In such way, the dimensions of the implied matrices will be reduced. They also assume that n is much smaller than m (tall-and-skinny matrices) in order to avoid a severe degradation of the algorithm's performance.
- Principal component analysis (PCA) tries to find a rotation such that the set of possibly correlated features transforms into a set of linearly uncorrelated features. The columns used in this orthogonal transformation are called principal components. This method is also designed for matrices with a low number of features.

Feature selection

As explained before, FS tries to select relevant subsets of relevant features without incurring much loss of information [64].

- VectorSlicer: the user selects manually a subset of features.
- RFormula: selects features specified by an R model formula.

- Chi-Squared selector: it orders categorical features using a Chi-Squared test of independence from the class. Then, it selects the most-dependent features. This is a filter method, which needs the number of features to select.

Feature indexers and encoders

These functions convert features from one type to another using indexing or encoding techniques.

- StringIndexer: converts a column of string into a column of numerical indices. The indices are ordered by label frequencies.
- OneHotEncoder: maps a column of strings to a column of unique binary vectors. This encoding allows better representation of categorical features since it removes the numerical order imposed by the previous method.
- VectorIndexer: automatically decides which features are categorical and transform them to category indices.

Other preprocessing methods for text mining

Text mining techniques try to structure the input text, yielding structured patterns of information.

- TF-IDF: this tool is aimed at quantifying how relevant each term is to a document, given a complete set of documents. Term Frequency (TF) measures the number of times that a term appears in a documents, whereas Inverse Document Frequency (IDF) measures how much information is given by a term according to its document frequency. TF is implemented using feature hashing for a better performance, so that each raw feature is mapped into an index. The dimension of the hast table is normally quite high (2^{20}) in order to avoid collisions.
- Word2Vec: it takes as input a text corpus and yields as output the word vectors. It first constructs a vocabulary from the text, and then learns vector representation of words.
- CountVectorizer: transforms a corpus into a set of vectors of token counts. It extracts the vocabulary using an estimator and counts the number of occurrences for each term.
- Tokenizer: breaks some text into individual terms using simple or regular expressions.
- StopWordsRemover: removes irrelevant words from the input text. The list of stop words is specified as parameter.
- n-gram: generates sequences of n-grams terms, where each one is formed by a space-delimited string of n consecutive words.

Feature selection

As mentioned before, FS has a key role to play in dealing with large-scale datasets, especially those that present an ultra-high dimensionality. However, FS methods, like many other learning methods, suffers from the "curse of dimensionality" [65], and consequently, are not expected to scale well. New paradigms and tools have emerged to solve this problematic [66]. Most of them are centered in the use of parallel processing to distribute the massive complexity burden across several nodes. Here, a list of the contributions for FS is presented:

- [59]: Singh et al. proposed a new approximate heuristic, optimized for logistic regression in MapReduce, which employs a greedy search to select features increasingly.
- [67]: Meena et al. designed an evolutionary approach based on Ant Colony Optimization (ACO) with the aim of finding the optimal subset of features. It parallelizes on Hadoop MapReduce some parts of the algorithm, such as: tokenization, the computation of association degrees, and the evaluation of solutions.
- [68]: Tanupabrungsun et al. proposed a Genetic Algorithm (GA) approach with a wrapper fitness function. In this work, the Hadoop master process is in charge of the management of the population whereas the fitness evaluation is parallelized.
- [69]: Triguero et al. proposed an evolutionary feature weighting model to learn the feature weights per map. They introduced a Reduce phase adding the weights and using a threshold to select the most relevant instance. This model was the winner solution in the ECBDL'14 competition.
- [70]: Peralta et al. proposed a different approach based on independent GA processes (executed on each partition). A voting scheme is employed to aggregate the partial solutions.
- [71]: Kumar et al. implemented three feature selectors (ANOVA, Kruskal–Wallis, and Friedman test) based on statistical test. All of them were parallelized on Hadoop MapReduce so as each feature is evaluated independently.
- [72]: Hodge et al. proposed an unified framework which uses binary Correlation Matrix Memories (CMMs) to store and retrieve patterns using matrix calculus. They propose to compute sequentially the CMMs, and them, to distribute them on Hadoop to obtain the final coefficients.
- [73]: A feature selection method based on differential privacy (Laplacian Noise) and a Gini-index measure was designed by Chen et al. This technique was implemented using a general MapReduce model.
- [74]: Zhao et al. proposed a FS framework for both unsupervised and supervised learning, which includes several measures, such as: the Akaike information criterion (AIC), the Bayesian information criterion (BIC), and the corrected Hannan–Quinn information criterion (HQC). This framework has been implemented on MPI.
- [75]: Sun et al. designed a method that computes the total combinatory mutual information, and the contribution degree between all feature variables and class variable. It uses a iterative process (implemented on Hadoop) to select the most relevant features.
- [76]: A filter method based on column subset selection was implemented by Ordozgoiti et al. However, as stated by the authors, this Spark algorithm is not designed to tackle high-dimensional problems.
- [77]: A simple version of TF-IDF (for Hadoop MapReduce) was designed by Chao et al. to deal with text mining problem on Big Data.
- [78]: Dalavi et al. proposed a novel weighting scheme based on supervised learning (using SVMs) for Hadoop MapReduce.
- [79]: He et al. implemented on Hadoop a FS method using positive approximation as an accelerator for traditional rough sets.

- [80]: Wang et al. designed a family of feature selection algorithms for online learning. The algorithm selects those features with bigger weights, according to a linear classifier based on L1-norm.
- [60]: Tan et al. reformulated the FS problem as a convex semi-infinite programming problem. They also proposed to speed up the training phase through several cache techniques and a modified accelerated proximal gradient method. This sequential approach (written in C++ and MatLab) has been included on this list because of its relevance and promising results on Big Data (see Table 1).

Imbalanced data

Classification problems are typically formed by a small set of classes. Some of them come with a tiny percentage of instances compared with the other classes. This highly-imbalanced problems are more noteworthy in Big Data environments where millions of instances are present. Some contributions to this topic have been implemented on Hadoop MapReduce:

- [81]: The first approach in dealing with imbalanced large-scale datasets was proposed by Park et al. In this work, a simple over-sampling technique was employed using Apache Hadoop and Hive on traffic data with a 14 % of positive instances.
- [82]: Hu et al. proposed an enhanced version of Synthetic Minority Over-sampling Technique (SMOTE) algorithm on MapReduce. This method focused on replicating those minority cases that only belong to the boundary region to solve the problem of original SMOTE, which omits the distribution of the original data while yields new samples.
- [83]: Rio et al. adapted some over-sampling, under-sampling and cost-sensitive methods for MapReduce. All these distributed algorithms apply a sampling technique on each data partition, and reduce the partial results by randomly selecting a fixed amount of instances. It was extended for extremely imbalanced data in [84] using a high over-sampling rate to highlight the presence of the minority class. This proposal has been tested against several bio-informatics problems (like contact map prediction and orthogonal detection) with accurate results [69, 85].
- [86]: Wang et al. proposed an algorithm that weighs the penalties associated to each instance in order to reduce the effect of less important points. This weighted boosting method aims at adjusting the weights of each instance in each iteration. The weighted instances are finally classified by a SVM classifier.
- [87]: Bhagat et al. proposed an extension to this work where a combination of SMOTE and a One-vs-All (OVA) approach is tested.
- [88]: Zhai et al. designed a oversampling technique based on nearest neighbors for ensemble learning. This technique yields several over-sampled sets by alternating the process between positive and negative instances. In this work, only the neighbors computation is reported to be distributed using MapReduce.
- [89]: Triguero et al. designed an evolutionary undersampling method for Big Data classification. It is based on two MapReduce stages: the first one builds a decision tree in each map after performing undersampling; and the second one, classifies the test set using the set of trees. The building phase is accelerated by a windowing technique. In [90], an iterative model was designed on Apache Spark aiming at

solving extremely imbalance problems [90]. Through Spark in-memory operations, this model is able to make an efficient use of data.

- [91]: Park et al. developed a distributed version of SMOTE algorithm for traffic detection. This version consists of two MapReduce jobs: the first one calculates distances among the input examples; and the second one sorts the results by distance. The latter step caches the original data into a distributed cache, which can be consider as a serious problem for scalability.

Incomplete data

In most of current real-life problems, there is a potential for incomplete data (also called missing data). Because of either human or machine failure, input data can present some gaps or errors. This problem needs to be faced early with some additional techniques (like imputation methods) that prevent the learning process from its negative effect. Although Big Data systems are more prone to incompleteness, just a couple of contributions have been proposed in the literature to solve this:

- [92]: Chen et al. designed a data cleansing method based on the combination of set-valued decision information system, and a deep analysis of missing information. The algorithm implements on Hadoop MapReduce the computation of the equivalent set-valued decision information system and the boolean equivalence matrix. By using this information, they purge duplicate and inconsistent objects from the input data.
- [93]: Zhang et al. run an investigation about the effect of rough sets over incomplete information systems. Its Twister MapReduce implementation aims at accelerating the computation of the relation matrix, one of the main structures in rough sets theory. One of its main advantages is the Sub-Merge operation implemented, which accelerates the process of joining the relation matrices and saves some space.

Discretization

Discretization task is frequently used to improve the performance and effectiveness of classifiers. It is also used to simplify, and therefore, to reduce continuous-valued datasets. For this reason, data discretization has become one of the most important task in the knowledge discovery process. Nevertheless, standard discretization are not prepared to deal with big datasets. Here, we present the unique contributions in this field:

- [94]: Zhang et al. implemented on Hadoop a parallel version of Chi-Squared discretization method. However, the main drawback of this method is its poor scalability due to the merging process is bounded by the number of input features.
- [95]: Ramírez et al. designed an efficient implementation of Fayyad's discretizer on Apache Spark. This entropy minimization proposal was re-adapted to completely distribute the computation burden associated to the most-consuming operations in this method: feature sorting and boundary points generation. In [96], an updated taxonomy of the most relevant discretization methods is presented along with the Big Data challenge that supposes the application of discretization techniques in large-scale scenarios. A distributed entropy minization discretizer is presented and evaluated on several big datasets.

Instance reduction

Instance selection is a type of preprocessing technique, which aims at reducing the number of samples to be considered in the learning phase. In spite of its promising results with small and medium datasets, this task is normally undermined when coping with large-scale datasets (from tens of thousands of instances onwards).

Just one contribution of Triguero et al. [97] has been able to address this problem from a distributed perspective up to now. In this work, the authors apply an advanced IR technique (called SSMA-SFLSDE) over each data partition (map phase) using Hadoop. The reduce phase offers several ways to aggregate the partial instance sets, either by: concatenating all partial results (baseline), filtering noisy prototypes, or merging redundant samples. An extension to this method was proposed in [98]. A second phase of parallelization based on windowing is included in this extension on the mappers side.

In this section, we have reviewed the most important contributions on large-scale preprocessing. Regarding MLlib, it offers a wide set of preprocessing algorithms, however, almost all these methods looks quite simple. Indeed, focusing on FS, only a simple statistical filter (Chi-squared) has been implemented. On the other hand, a list of more complex and diverse contributions have been presented in the literature. Nevertheless, just a few of these methods have been tested against really huge datasets (greater than 5 GBs). Accordingly, more scalable proposals are required to tackle the actual size of incoming data, and to cover neglected fields of preprocessing (like IR).

Challenges and new possibilities in big data preprocessing

This last section of the paper will be devoted to point out all the existing lines in which the efforts on Big Data preprocessing should be made in the next years. The new possibilities on this topic will be centered onto three main key points: new technologies, to scale the data preprocessing techniques and new learning paradigms on which they can be applied.

New technologies

As we can see in previous content of this paper, new technologies for Big Data are emerging in the last years and few attempts of data preprocessing proposals can be found adapted to take advantage of them. It is clear that Spark [10] is offering better performance results than Hadoop [7] in processing. But also, Spark is a newer technology and there has been little time to develop ideas until now. Thus, the near future will offer new methods developed in Spark under the library MLlib [61] which is growing increasingly.

It is worth mentioning that other emerging platform, such as Flink [14], are bridging the gap of stream and batch processing that Spark currently has. Flink is a streaming engine that can also do batches whereas Spark is a batch engine that emulates streaming by micro-batches. This results in that Flink is more efficient in terms of low latency, especially when dealing with real time analytical processing.

In the particular case of data preprocessing in Spark, excepting basic data preprocessing, we can find some developments in FS and discretization for Big Data. Spark is a more mature technology and implements MLlib with tens of already available learning algorithms. This will make easy and encourage the integration of novel data preprocessing methods in a near future. However, it is desirable to start the development of data preprocessing techniques on Flink, in particular with streaming and real-time applications.

Scaling data preprocessing techniques to deal with big data

Another remarkable outcome derived from the previous analysis of existing Big Data preprocessing techniques is that most of the effort has been devoted to the development of FS methods, and even there are some data preprocessing families in which nothing or almost nothing has been done.

- Instance reduction: these techniques will allow us to arrange a subset of data to carry out the same learning tasks that we could do with original data, but with a low decrease of performance. It is very desirable to have a complete set of instance reduction techniques to obtain subsets of data from big databases for certain purposes and paradigms. The key problem is that these techniques have to be re-adjusted to deal with large scale data, they require high computation capabilities and they are assumed to follow an iterative procedure.
- Missing values imputation: it is a hard problem in which many relationships among data have to be analyzed to estimate the best possible value to replace a missing value.
- Noise treatment: it is again a complex problem in which decisions depend on two perspectives: the computation of similarities among data points and the run and fusion of several decisions coming from ensembles to enable the noise identification approach.

Additionally, there is an open issue related to the arrangement and combination of several data preprocessing techniques to achieve the optimal outcome for a data mining process. This is discussed in [99], where the most influential data preprocessing techniques are presented and some instructive experimental studies emphasize the effects caused by different arrangement of data preprocessing techniques. This is an original complex challenge, but it will be more complex according to the data scales in Big Data scenarios. This complexity may also be influenced by other factors that mainly depends of the data preprocessing technique in question; such as its dependency of intermediate results, its capacity of treating different volumes of data, its possibility of parallelization and iterative processing, or even the input it requires or the output it provides.

New big data learning paradigms

Data mining is not a static field and new problems are continuously arising. In consequence data preprocessing techniques are evolving along with data mining and with the appearance of new challenges and problems that data mining tries to tackle, new proposals of data preprocessing methods have been proposed.

These problems are becoming a part of the Big Data universe and they are being currently addressed by some of the mentioned technologies [100]. In addition, they will require data preprocessing techniques to ensure high quality solutions and good performance in the results obtained. This is another major challenge for Big Data preprocessing and it will concern different learning paradigms, besides classification and regression, such as:

- Unsupervised learning: Clustering [101] and rule association mining [102] have been addressed in Big Data. Developments on real-time applications can be also found in the literature [103]. It is well-known that the success of these problems depends heavily on the quality of data, being the data cleaning, transformation and discretization the techniques with the most important role for this.

- Semi-supervised learning: A significant growth of applications and solutions on this paradigm is expected in the near future. Due to the fact of generating and storing more and more data, the labeling of examples cannot be done for all and the predictive or descriptive task will be supported by a subset of labelled examples [104]. Data preprocessing, especially at the instance level [105], would be useful to improve the quality of this kind of data.
- Data streams and real-time processing: processing large data offering real-time responses is one of the most popular and demanding paradigms in business [106]. Currently, there are some specific approaches in Big Data streams [107–109] and even software development [110]. Data preprocessing techniques, such as noise editing [58], should be able to tackle Big Data scenarios in upcoming applications.
- Non-standard supervised problems: there are some other popular supervised paradigms in which Big Data solutions will be necessary soon. This is the case of ordinal classification/regression [111], multi-label classification [52, 53, 112] or multi-instance learning [54]. All the possible data preprocessing approaches will also be required to enable and improve these solutions.

Conclusions

At the present, the size, variety and velocity of data is huge and continues to increase every day. The use of Big Data frameworks to store, process, and analyze data has changed the context of the knowledge discovery from data, especially the processes of data mining and data preprocessing. In this paper, we presented a review on the rise of data preprocessing in cloud computing. We presented a updated categorization of data preprocessing contributions under the big data framework. The review covered different families of data preprocessing techniques, such as feature selection, imperfect data, imbalanced learning and instance reduction as well as the maximum size supported and the frameworks in which they have been developed. Furthermore, the key issues in big data preprocessing were highlighted.

In the future, significant challenges and topics must be addressed by the industry and academia, especially those related to the use of new platforms such as Apache Spark/Flink, the enhancement of scaling capabilities of existing techniques and the approach to new big data learning paradigms. Researchers, practitioners, and data scientists should collaborate to guarantee the long-term success of big data preprocessing and to collectively explore new domains.

Endnote

[1]*mllib* and *ml* documentation: http://spark.apache.org/docs/latest/mllib-guide.html.

Acknowledgements
This work was partially supported by the Spanish Ministry of Science and Technology under project TIN2014-57251-P and the Andalusian Research Plan P11-TIC-7765.

Authors' contributions
All authors have contributed equally to finish this paper. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Received: 6 January 2016 Accepted: 15 September 2016
Published online: 01 November 2016

References

1. Aggarwal CC. Data Mining: The Textbook. Berlin, Germany: Springer; 2015.
2. Wu X, Zhu X, Wu GQ, Ding W. Data mining with big data. IEEE Trans Knowl Data Eng. 2014;26(1):97–107.
3. Laney D. 3D Data Management: Controlling Data Volume, Velocity and Variety. 2001. http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf. Accessed July 2015.
4. Fernández A, del Río S, López V, Bawakid A, del Jesús MJ, Benítez JM, et al. Big data with cloud computing: an insight on the computing environment, mapreduce, and programming frameworks. Wiley Interdisc Rew Data Min Knowl Discov. 2014;4(5):380–409.
5. Dean J, Ghemawat S. Mapreduce: Simplified data processing on large clusters. In: OSDI 2004. San Francisco, CA; 2004. p. 137–50.
6. White T. Hadoop, The Definitive Guide. Sebastopol: O'Reilly Media, Inc; 2012.
7. Apache Hadoop Project. Apache Hadoop. 2015. http://hadoop.apache.org/. Accessed December 2015.
8. Lin J. Mapreduce is good enough? if all you have is a hammer, throw away everything that's not a nail! Big Data. 2012;1(1):28–37.
9. Karau H, Konwinski A, Wendell P, Zaharia M. Learning Spark: Lightning-Fast Big Data Analytics. Sebastopol: O'Reilly Media; 2015.
10. Spark A. Apache Spark: Lightning-fast cluster computing. https://spark.apache.org/. Accessed December 2015.
11. Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. *NSDI'12*. San Jose; 2012. p. 15–28.
12. InfoWorld. Apache Flink: New Hadoop contender squares off against Spark. 2015. http://www.infoworld.com/article/2919602/hadoop/flink-hadoops-new-contender-for-mapreduce-spark.html. Accessed December 2015.
13. Storm. Apache Storm. 2015. http://storm-project.net/. Accessed December 2015.
14. Flink. Apache Flink. 2015. https://flink.apache.org/. Accessed December 2015.
15. Pyle D. Data Preparation for Data Mining. San Francisco: Morgan Kaufmann Publishers Inc.; 1999.
16. García S, Luengo J, Herrera F. Data Preprocessing in Data Mining. Berlin: Springer; 2015.
17. Han J, Kamber M, Pei J. Data Mining: Concepts and Techniques, 3rd ed. Burlington: Morgan Kaufmann Publishers Inc; 2011.
18. Zaki MJ, Meira W. Data Mining and Analysis: Fundamental Concepts and Algorithms. New York: Cambridge University Press; 2014.
19. Wang H, Wang S. Mining incomplete survey data through classification. Knowl Inf Syst. 2010;24(2):221–33.
20. Luengo J, García S, Herrera F. On the choice of the best imputation methods for missing values considering three groups of classification methods. Knowl Inf Syst. 2012;32(1):77–108.
21. Little RJA, Rubin DB. Statistical Analysis with Missing Data. Wiley Series in Probability and Statistics, 1st ed. New York: Wiley; 1987.
22. Frénay B, Verleysen M. Classification in the presence of label noise: A survey. IEEE Trans Neural Netw Learn Syst. 2014;25(5):845–69.
23. Zhu X, Wu X. Class Noise vs. Attribute Noise: A Quantitative Study. Artif Intell Rev. 2004;22:177–210.
24. Bellman RE. Adaptive Control Processes - A Guided Tour. Princeton, NJ: Princeton University Press; 1961.
25. Hall MA. Correlation-based feature selection for machine learning. Waikato University, Department of Computer Science. 1999.
26. Guyon I, Elisseeff A. An introduction to variable and feature selection. J Mach Learn Res. 2003;3:1157–82.
27. Chandrashekar G, Sahin F. A survey on feature selection methods. Comput Electr Eng. 2014;40(1):16–28.
28. Kim JO, Mueller CW. Factor Analysis: Statistical Methods and Practical Issues (Quantitative Applications in the Social Sciences). New York: Sage Publications, Inc; 1978.
29. Dunteman GH. Principal Components Analysis. A Sage Publications. Thousand Oaks: SAGE Publications; 1989.
30. Roweis S, Saul L. Nonlinear dimensionality reduction by locally linear embedding. Science. 2000;290(5500): 2323–326.
31. Tenenbaum JB, Silva V, Langford JC. A global geometric framework for nonlinear dimensionality reduction. Science. 2000;290(5500):2319–323.
32. Liu H, Motoda H. On issues of instance selection. Data Min Knowl Disc. 2002;6(2):115–30.
33. Olvera-López JA, Carrasco-Ochoa JA, Martínez-Trinidad JF, Kittler J. A review of instance selection methods. Artif Intell Rev. 2010;34(2):133–43.
34. García S, Derrac J, Cano JR, Herrera F. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. IEEE Trans Pattern Anal Mach Intell. 2012;34(3):417–35.
35. Triguero I, Derrac J, García S, Herrera F. A taxonomy and experimental study on prototype generation for nearest neighbor classification. IEEE Trans Syst Man Cybern Part C. 2012;42(1):86–100.
36. Liu H, Hussain F, Tan CL, Dash M. Discretization: An enabling technique. Data Min Knowl Discov. 2002;6(4):393–423.
37. Wu X, Kumar V, (eds). The Top Ten Algorithms in Data Mining. Boca Ratón, Florida: CRC Press; 2009.
38. Quinlan JR. C4.5: Programs for Machine Learning. San Francisco, CA: Morgan Kaufmann Publishers Inc.; 1993.
39. Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Proceedings of the 20th Very Large Data Bases Conference (VLDB); 1994. p. 487–99.
40. Yang Y, Webb GI. Discretization for naive-bayes learning: managing discretization bias and variance. Mach Learn. 2009;74(1):39–74.
41. Yang Y, Webb GI, Wu X. Discretization methods. In: Data Mining and Knowledge Discovery Handbook. Germany: Springer; 2010. p. 101–16.
42. García S, Luengo J, Sáez JA, López V, Herrera F. A Survey of Discretization Techniques: Taxonomy and Empirical Analysis in Supervised Learning. IEEE Trans Knowl Data Eng. 2013;25(4):734–50.
43. López V, Fernández A, García S, Palade V, Herrera F. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. Inf Sci. 2013;250:113–41.

44. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: synthetic minority over-sampling technique. J Artif Intell Res. 2002;16(1):321–57.
45. Li Z, Tang J. Unsupervised feature selection via nonnegative spectral analysis and redundancy control. IEEE Trans Image Process. 2015;24(12):5343–355.
46. Han J, Sun Z, Hao H. Selecting feature subset with sparsity and low redundancy for unsupervised learning. Knowl-Based Syst. 2015;86:210–23.
47. Wang S, Pedrycz W, Zhu Q, Zhu W. Unsupervised feature selection via maximum projection and minimum redundancy. Knowl-Based Syst. 2015;75:19–29.
48. Ishioka T. Imputation of missing values for unsupervised data using the proximity in random forests. In: Nternational Conference on Mobile, Hybrid, and On-line Learning. Nice; 2013. p. 30–6.
49. Bondu A, Boullé M, Lemaire V. A non-parametric semi-supervised discretization method. Knowl Inf Syst. 2010;24(1):35–57.
50. Impedovo S, Barbuzzi D. Instance selection for semi-supervised learning in multi-expert systems: A comparative analysis. Neurocomputing. 2015;5:61–70.
51. Williams D, Liao X, Xue Y, Carin L, Krishnapuram B. On classification with incomplete data. IEEE Trans Pattern Anal Mach Intell. 2007;29(3):427–36.
52. Charte F, Rivera AJ, del Jesus MJ, Herrera F. MLSMOTE: Approaching imbalanced multilabel learning through synthetic instance generation. Knowl-Based Syst. 2015;89:385–97.
53. Charte F, Rivera AJ, del Jesús MJ, Herrera F. Addressing imbalance in multilabel classification: Measures and random resampling algorithms. Neurocomputing. 2015;163:3–16.
54. Xiaoguang W, Xuan L, Nathalie J, Stan M. Resampling and cost-sensitive methods for imbalanced multi-instance learning. In: 13th IEEE International Conference on Data Mining Workshops, ICDM Workshops, TX, USA, December 7-10, 2013. USA: IEEE; 2013. p. 808–16.
55. Jiang N, Gruenwald L. Estimating missing data in data streams. In: 12th International Conference on Database Systems for Advanced Applications, DASFAA 2007; Bangkok; Thailand; 9 April 2007 Through 12 April 2007. Bangkok; 2007.
p. 981–7.
56. Zhang P, Zhu X, Tan J, Guo L. SKIF: a data imputation framework for concept drifting data streams In: Huang J, Koudas N, Jones GJF, Wu X, Collins-Thompson K, An A, editors. CIKM. Toronto; 2010. p. 1869–1872.
57. Kogan J. Feature selection over distributed data streams through convex optimization. In: Proceedings of the 2012 SIAM International Conference on Data Mining. Anaheim; 2012. p. 475–84.
58. Lu N, Lu J, Zhang G, de Mántaras RL. A concept drift-tolerant case-base editing technique. Artif Intell. 2016;230: 108–33.
59. Singh S, Kubica J, Larsen SE, Sorokina D. Parallel large scale feature selection for logistic regression. In: SIAM International Conference on Data Mining (SDM). Sparks, Nevada; 2009. p. 1172–1183.
60. Tan M, Tsang IW, Wang L. Towards ultrahigh dimensional feature selection for big data. J Mach Learn Res. 2014;15: 1371–1429.
61. Meng X, Bradley JK, Yavuz B, Sparks ER, Venkataraman S, Liu D, Freeman J, Tsai DB, Amde M, Owen S, Xin D, Xin R, Franklin MJ, Zadeh R, Zaharia M, Talwalkar A. MLlib: Machine learning in apache spark. CoRR. J Machine Learning Res. 2015;17(2016):1–7. abs/1505.06807.
62. Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Franklin MJ, Ghodsi A, Zaharia M. Spark SQL: Relational data processing in spark. In: ACM SIGMOD International Conference on Management of Data. SIGMOD '15. Melbourne; 2015. p. 1383–1394.
63. Guyon I, Gunn S, Nikravesh M, Zadeh LA. Feature Extraction: Foundations and Applications (Studies in Fuzziness and Soft Computing). Germany: Springer; 2006.
64. Blum AL, Langley P. Selection of relevant features and examples in machine learning. Artif Intell. 1997;97(1-2): 245–71.
65. Zhai Y, Ong Y, Tsang IW. The emerging "big dimensionality". IEEE Comput Intell Mag. 2014;9(3):14–26.
66. Bolón-Canedo V, Sánchez-Marono N, Alonso-Betanzos A. Recent advances and emerging challenges of feature selection in the context of big data. Knowl-Based Syst. 2015;86:33–45.
67. Meena MJ, Chandran KR, Karthik A, Samuel AV. An enhanced ACO algorithm to select features for text categorization and its parallelization. Expert Syst Appl. 2012;39(5):5861–871.
68. Tanupabrungsun S, Achalakul T. Feature reduction for anomaly detection in manufacturing with mapreduce GA/kNN. In: 19th IEEE International Conference on Parallel and Distributed Systems (ICPADS). Seoul; 2013. p. 639–44.
69. Triguero I, del Río S, López V, Bacardit J, Benítez JM, Herrera F. ROSEFW-RF: The winner algorithm for the ECBDL'14 big data competition: An extremely imbalanced big data bioinformatics problem. Knowl-Based Syst. 2015;87:69–79.
70. Peralta D, Río S, Ramírez S, Triguero I, Benítez JM, Herrera F. Evolutionary feature selection for big data classification: A mapreduce approach. Math Probl Eng. 2015. Article ID 246139.
71. Kumar M, Rath SK. Classification of microarray using mapreduce based proximal support vector machine classifier. Knowl-Based Syst. 2015;89:584–602.
72. Hodge VJ, O'Keefe S, Austin J. Hadoop neural network for parallel and distributed feature selection. Neural Netw. 2016. doi:10.1016/j.neunet.2015.08.011.
73. Chen K, Wan W-q, Li Y. Differentially private feature selection under mapreduce framework. J China Univ Posts Telecommun. 2013;20(5):85–103.
74. Zhao Z, Zhang R, Cox J, Duling D, Sarle W. Massively parallel feature selection: an approach based on variance preservation. Mach Learn. 2013;92(1):195–220.
75. Sun Z, Li Z. Data intensive parallel feature selection method study. In: International Joint Conference on Neural Networks (IJCNN). USA: IEEE; 2014. p. 2256–262.
76. Ordozgoiti B, Gómez-Canaval S, Mozo A. Massively parallel unsupervised feature selection on spark. In: New Trends in Databases and Information Systems. Communications in Computer and Information Science. Germany: Springer; 2015. p. 186–96.

77. Chao P, Bin W, Chao D. Design and implementation of parallel term contribution algorithm based on mapreduce model. In: 7th Open Cirrus Summit. USA: IEEE; 2012. p. 43–7.
78. Dalavi M, Cheke S. Hadoop mapreduce implementation of a novel scheme for term weighting in text categorization. In: International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT). USA: IEEE; 2014. p. 994–9.
79. He Q, Cheng X, Zhuang F, Shi Z. Parallel feature selection using positive approximation based on mapreduce. In: 11th International Conference on Fuzzy Systems and Knowledge Discovery FSKD. USA: IEEE; 2014. p. 397–402.
80. Wang J, Zhao P, Hoi SCH, Jin R. Online feature selection and its applications. IEEE Trans Knowl Data Eng. 2014;26(3):698–710.
81. Park SH, Ha YG. Large imbalance data classification based on mapreduce for traffic accident prediction. In: 8th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS). Birmingham; 2014. p. 45–9.
82. Hu F, Li H, Lou H, Dai J. A parallel oversampling algorithm based on NRSBoundary-SMOTE. J Inf Comput Sci. 2014;11(13):4655–665.
83. del Río S, López V, Benítez JM, Herrera F. On the use of mapreduce for imbalanced big data using random forest. Inf Sci. 2014;285:112–37.
84. del Río S, Benítez JM, Herrera F. Analysis of data preprocessing increasing the oversampling ratio for extremely imbalanced big data classification. In: IEEE TrustCom/BigDataSE/ISPA, Volume 2. USA: IEEE; 2015. p. 180–5.
85. Galpert D, Del Río S, Herrera F, Ancede-Gallardo E, Antunes A, Aguero-Chapin G. An effective big data supervised imbalanced classification approach for ortholog detection in related yeast species. BioMed Res Int. 2015. article 748681.
86. Wang X, Liu X, Matwin S. A distributed instance-weighted SVM algorithm on large-scale imbalanced datasets. In: IEEE International Conference on Big Data. USA: IEEE; 2014. p. 45–51.
87. Bhagat RC, Patil SS. Enhanced SMOTE algorithm for classification of imbalanced big-data using random forest. In: IEEE International Advance Computing Conference (IACC). USA: IEEE; 2015. p. 403–8.
88. Zhai J, Zhang S, Wang C. The classification of imbalanced large data sets based on mapreduce and ensemble of elm classifiers. Int J Mach Learn Cybern. 2016. doi:10.1007/s13042-015-0478-7.
89. Triguero I, Galar M, Vluymans S, Cornelis C, Bustince H, Herrera F, Saeys Y. Evolutionary undersampling for imbalanced big data classification. In: IEEE Congress on Evolutionary Computation, CEC. USA: IEEE; 2015. p. 715–22.
90. Triguero I, Galar M, Merino D, Maillo J, Bustince H, Herrera F. Evolutionary undersampling for extremely imbalanced big data classification under apache spark. In: IEEE Congress on Evolutionary Computation, CEC, In Press. USA: IEEE; 2016.
91. Park S-h, Kim S-m, Ha Y-g. Highway traffic accident prediction usin vds big data analysis. J Supercomput. 2016. doi:10.1007/s11227-016-1624-z.
92. Chen F, Jiang L. A parallel algorithm for datacleansing in incomplete information systems using mapreduce. In: 10th International Conference on Computational Intelligence and Security (CIS). Kunmina, China; 2014. p. 273–7.
93. Zhang J, Wong JS, Pan Y, Li T. A parallel matrix-based method for computing approximations in incomplete information systems. IEEE Trans Knowl Data Eng. 2015;27(2):326–39.
94. Zhang Y, Yu J, Wang J. Parallel implementation of chi2 algorithm in mapreduce framework. In: Human Centered Computing - First International Conference, HCC. Germany: Springer; 2014. p. 890–9.
95. Ramírez-Gallego S, García S, Mourino-Talin H, Martínez-Rego D, Bolon-Canedo V, Alonso-Betanzos A, Benitez JM, Herrera F. Distributed entropy minimization discretizer for big data analysis under apache spark. In: IEEE TrustCom/BigDataSE/ISPA, Volume 2. USA: IEEE; 2015. p. 33–40.
96. Ramírez-Gallego S, García S, Mouriño-Talín H, Martínez-Rego D, Bolón-Canedo V, Alonso-Betanzos A, Benítez JM, Herrera F. Data discretization: taxonomy and big data challenge. Wiley Interdiscip Rev Data Min Knowl Disc. 2016;6(1):5–21.
97. Triguero I, Peralta D, Bacardit J, García S, Herrera F. MRPR: A mapreduce solution for prototype reduction in big data classification. Neurocomputing. 2015;150 Part A:331–45.
98. Triguero I, Peralta D, Bacardit J, García S, Herrera F. A combined mapreduce-windowing two-level parallel scheme for evolutionary prototype generation. In: IEEE Congress on Evolutionary Computation (CEC); 2014. p. 3036–043.
99. García S, Luengo J, Herrera F. Tutorial on practical tips of the most influential data preprocessing algorithms in data mining. Knowl-Based Syst. 2016. doi:10.1016/j.knosys.2015.12.006.
100. Hashem IAT, Yaqoob I, Anuar NB, Mokhtar S, Gani A, Khan SU. The rise of "big data" on cloud computing: Review and open research issues. Inf Syst. 2015;47:98–115.
101. Tsapanos N, Tefas A, Nikolaidis N, Pitas I. A distributed framework for trimmed kernel k-means clustering. Pattern Recogn. 2015;48(8):2685–698.
102. Chen Y, Li F, Fan J. Mining association rules in big data with ngep. Clust Comput. 2015;18(2):577–85.
103. Aghabozorgi S, Seyed Shirkhorshidi A, Ying Wah T. Time-series clustering - a decade review. Inf Syst. 2015;53(C): 16–38.
104. Zhu Q, Zhang H, Yang Q. Semi-supervised affinity propagation clustering based on subtractive clustering for large-scale data sets. In: Intelligent Computation in Big Data Era. Germany: Springer; 2015. p. 258–265.
105. Triguero I, García S, Herrera F. SEG-SSC: a framework based on synthetic examples generation for self-labeled semi-supervised classification. IEEE Trans Cybern. 2015;45(4):622–34.
106. Gupta S. Learning Real-time Processing with Spark Streaming. Birmingham: PACKT Publishing; 2015.
107. Works K, Rundensteiner EA. Practical identification of dynamic precedence criteria to produce critical results from big data streams. Big Data Res. 2015;2(4):127–44.
108. Luts J. Real-time semiparametric regression for distributed data sets. IEEE Trans Knowl Data Eng. 2015;27(2):545–57.

109. Sun D, Zhang G, Yang S, Zheng W, Khan SU, Li K. Re-stream: Real-time and energy-efficient resource scheduling in big data stream computing environments. Inf Sci. 2015;319:92–112.
110. De Francisci Morales G, Bifet A. Samoa: Scalable advanced massive online analysis. J Mach Learn Res. 2015;16(1): 149–53.
111. Gutiérrez PA, Pérez-Ortiz M, Sánchez-Monedero J, Fernandez-Navarro F, Hervás-Martínez C. Ordinal regression methods: survey and experimental study. IEEE Trans Knowl Data Eng. 2015;28(1):127–46.
112. Gibaja E, Ventura S. A tutorial on multilabel learning. ACM Comput Surv. 2015;47(3):52–15238.

1.3 Multivariate Discretization Based on Evolutionary Cut Points Selection for Classification

- S. Ramírez-Gallego, S. García, J. M. Benítez and F. Herrera, Multivariate Discretization Based on Evolutionary Cut Points Selection for Classification. IEEE Transactions on Cybernetics 46 (3) (2016) 595–608, doi: 10.1109/TCYB.2015.2410143
  - Status: **Published**.
  - Impact Factor (JCR 2016): 7.384
  - Subject Category: Computer Science, Artificial Intelligence. Ranking 5 / 133 (**Q1**).
  - Subject Category: Computer Science, Cybernetics. Ranking 1 / 22 (**Q1**).

Multivariate Discretization Based on Evolutionary Cut Points Selection for Classification

Sergio Ramírez-Gallego, Salvador García, José Manuel Benítez, *Member, IEEE*, and Francisco Herrera

***Abstract*—Discretization is one of the most relevant techniques for data preprocessing. The main goal of discretization is to transform numerical attributes into discrete ones to help the experts to understand the data more easily, and it also provides the possibility to use some learning algorithms which require discrete data as input, such as Bayesian or rule learning. We focus our attention on handling multivariate classification problems, where high interactions among multiple attributes exist. In this paper, we propose the use of evolutionary algorithms to select a subset of cut points that defines the best possible discretization scheme of a data set using a wrapper fitness function. We also incorporate a reduction mechanism to successfully manage the multivariate approach on large data sets. Our method has been compared with the best state-of-the-art discretizers on 45 real datasets. The experiments show that our proposed algorithm overcomes the rest of the methods producing competitive discretization schemes in terms of accuracy, for C4.5, Naive Bayes, PART, and PrUning and BuiLding Integrated in Classification classifiers; and obtained far simpler solutions.**

***Index Terms*—Classification, data mining (DM), data preprocessing, discretization, evolutionary algorithms (EAs), numerical attributes.**

I. Introduction

DATA preprocessing [1] is a crucial research topic in data mining (DM) since most real-world databases are highly influenced by negative elements such as the presence of noise, missing values, inconsistent, and superfluous data. The reduction of data is also an essential task especially when dealing with large data sets, focusing on the selection or extraction of the most informative features [2] or instances [3] in the data.

Discretization, as one of the basic reduction techniques, has received increasing research attention in recent years [4], [5] and has become one of the most broadly used techniques in DM. The objective of a discretization process is to transform numerical attributes into discrete ones by producing a finite number of intervals, and by associating a discrete, numerical value to each interval [6], [7]. Although real-word DM tasks often involve numerical attributes, many algorithms can only handle categorical attributes, such as CN2 [8], algorithm quasioptimal-learning family [9], or Naive Bayes [10], [11], whereas others can deal with numerical attributes but would perform better on discrete-valued features[12].[1]

Among the most important advantages of using discretized data is that it is simpler and more reduced than numerical data. For example, some kind of decision trees yield more compact, shorter, and more accurate results than the derived ones using numerical values [6], [13]. Discretization of data besides has the effect of improving the speed and accuracy of DM algorithms.

Cut points selection problem for discretization is formed by all the singleton values present in each input attribute (candidate points). As this space can become very complex, specially when the data grow (both instances and features); we resorted to the use of a subset of attribute values or cut points, considering only those points that fall in the class borders [boundary points (BPs)]. Since this problem can be considered as an optimization problem with a binary search space, evolutionary algorithms (EAs) can be applied.

EAs have been used for data preparation with promising results [14]. In discretization, few evolutionary approaches can be found in the literature. In [15], a multivariate proposal is presented based on finding hidden association patterns using genetic algorithms (GAs) and clustering. Some discretization approaches based on rough sets have been proposed in [16] and [17], which also address this problem by using GAs. An important development in this area was done in [18], where an estimation of distribution algorithm is used for optimizing a Naive Bayes wrapper-based discretizer. A multivariate discretization EA for optimal cut points selection was proposed in [19]. The objective of this algorithm was to maximize the accuracy of the subsequent classification and to simplify the solution by using an inconsistency-based fitness function [20], [21].

In this paper, we present an evolutionary-based discretization algorithm with binary representation called evolutionary multivariate discretizer (EMD), which selects the most adequate combination of boundary cut points to create discrete intervals. Our proposal is inspired by the work presented

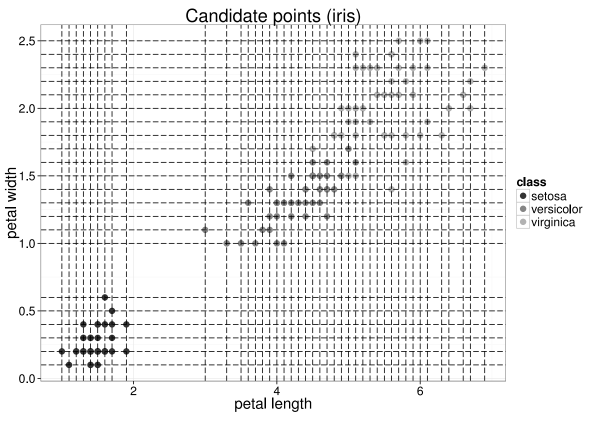
Manuscript received October 22, 2014; revised January 14, 2015; accepted February 27, 2015. Date of publication March 18, 2015; date of current version February 12, 2016. This work was supported in part by the Spanish Ministry of Science and Technology under Project TIN2011-28488, and in part by the Andalusian Research Plans under Grant P11-TIC-7765 and Grant P10-TIC-6858. This paper was recommended by Associate Editor Q. Zhao.

S. Ramírez-Gallego, J. M. Benítez, and F. Herrera are with the Department of Computer Science and Artificial Intelligence, CITIC-UGR (Research Center on Information and Communications Technology), University of Granada, Granada 18071, Spain (e-mail: sramirez@decsai.ugr.es).

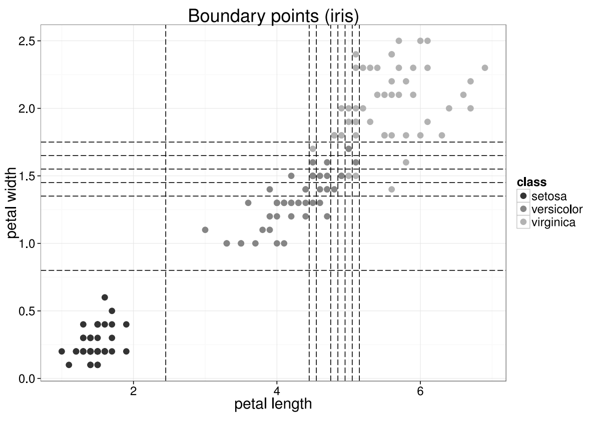
S. García is with the Department of Computer Science, University of Jaén, Jaén 23071, Spain, and also with the Department of Information Systems, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 22254, Saudi Arabia.

Digital Object Identifier 10.1109/TCYB.2015.2410143

[1]Although there are subsequent versions of these classic algorithms that are able to manage continuous attributes (e.g., for Naive Bayes [11]), in this paper, we will focus on the standard version of these classifiers.



(a)



(b)

Fig. 1. Cut points selection: problem. (a) Candidate points. (b) Boundary points.

in [19], but we have introduced some new enhancements, such as a wrapper-based fitness function to yield more competitive discretization solutions and a chromosome reduction mechanism to agilize the discretization process. Besides, our algorithm is a multivariate approach as it is able to take advantage of the existing dependencies among the complete set of attributes. These enhancements allow us to obtain better outcomes than in the previous approach.

We compare our approach with the discretizers emphasized as the best performing according to [5]. We consider two of the most influential classifiers that benefit from discretization: C4.5 and Naive Bayes [22]. We also extend the experiments using two additional and relevant classifiers: PART and PrUning and BuiLding Integrated in Classification (PUBLIC). The empirical study consists of 45 datasets, seven discretizers for comparison, and an analysis based on nonparametric statistical testing [23]–[25].

The rest of this paper is organized as follows. Section II defines some basic concepts used in this paper and categorizes our algorithm according to a discretization taxonomy. In Section III, our proposal is explained. In Section IV, we provide the experimental framework, the results obtained, and an analysis based on nonparametric statistical testing. Finally, Section V concludes this paper.

II. Background

In this section, we formally define the problem of discretization and we propose an approximation to reduce the algorithm's complexity through a reduction of the set of cut points. After that, we enumerate the typical features used to categorize the discretizers [5] and consequently, we classify our algorithm.

Finally, as our algorithm is based on the evolutionary computation idea and particularly, on the subclass of GAs; we explain briefly them and present the CHC model used in our proposal.

A. Definitions

Considering a classification problem with C target classes, a set with N instances, and M attributes, we can define the discretization as follows. A discretization algorithm would partition a continuous attribute A into k_A discrete and disjoint intervals

$$D_A = \{[d_0, d_1], (d_1, d_2], \ldots, (d_{k_A-1}, d_{k_A}]\} \tag{1}$$

where d_0 and d_{k_A}, respectively, are the minimum and maximal value, the values in D_A are arranged in ascending order. Such set of discrete intervals D_A is called a discretization scheme on attribute A and $P_A = \{d_1, d_2, \ldots, d_{k_A-1}\}$ is the set of cut points of attribute A. Finally, P denotes the complete set of cut points for all the continuous attributes in M. Therefore, the search space is formed by all the candidate cut points for each attribute, which is basically all the different existing values in the training set, considering each attribute separately.

In order to reduce the complexity of the initial search space, we resorted to the use of a reduced subset formed by the BPs of each attribute. Partitioning of a numerical attribute A begins by sorting its values into ascending order. Let Dom(A) denote the domain of the attribute A and $\text{val}_A(s)$ denote the value of the attribute A in a sample $s \in S$. If a pair of samples $u, v \in S$ exists, having different classes, such that $\text{val}_A(u) < \text{val}_A(v)$, and another sample $w \in S$ does not exist such that $\text{val}_A(u) < \text{val}_A(w) < \text{val}_A(v)$. We define a BP $\text{bp} \in \text{Dom}(A)$ as the midpoint value between $\text{val}_A(u)$ and $\text{val}_A(v)$.

Thus, the set of BPs for attribute A is denoted as BP_A, and BP denotes the complete set. BPs are known to form the optimal intervals for most of the evaluation measures used, as inconsistency or information gain [26]. Hence, using only the BPs in the search space, we can obtain significant savings in complexity and time consumption. Additionally, the above definition of BPs allow us to achieve the maximum separability between classes, thus obtaining better discretization results. A representation of the candidate and BPs for the iris dataset is showed in Fig. 1.

B. Categorization

Discretization methods may be categorized according to many features. Typically, discretization algorithms are classified into two main categories: 1) top-down (splitting) and 2) bottom-up (merging). Top-down methods [27] start from

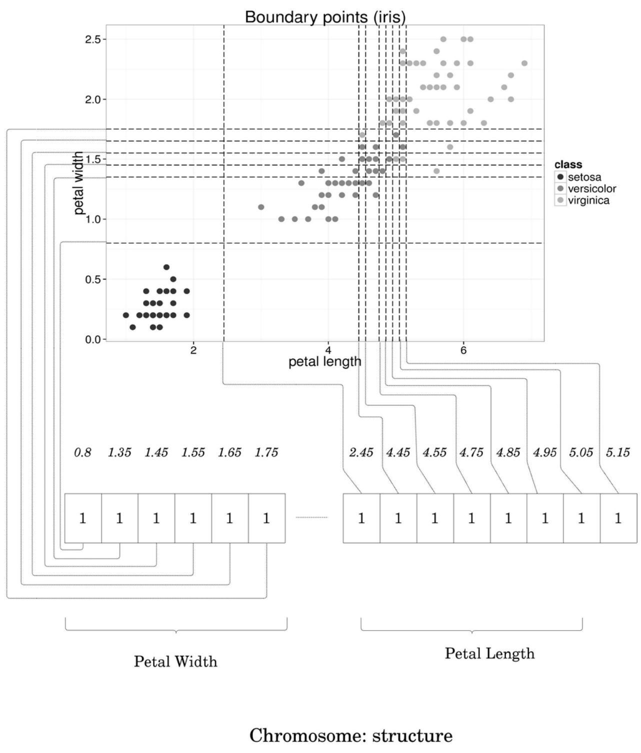


Fig. 2. Chromosome representation.

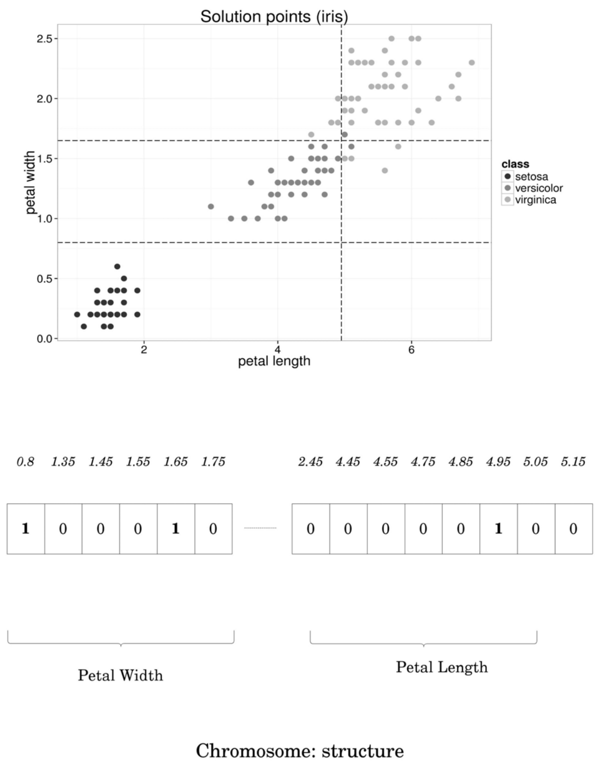


Fig. 3. Chromosome solution.

the initial interval and recursively split it into smaller intervals, while bottom-up algorithms begin with the set of all distinct values and iteratively merge adjacent intervals [28]. Apart from that, discretization methods may be grouped as below.

1) *Supervised Versus Unsupervised:* Unsupervised discretizers do not consider the class label whereas supervised ones do.
2) *Local Versus Global:* Global discretizers require all available data in the attribute whereas local ones do not.
3) *Dynamic Versus Static:* Dynamic discretizers act when the learner is constructing the models whereas the static ones proceed before the learning stage.
4) *Splitting Versus Merging:* This refers to the procedure used to yield new intervals, using either splittings or joinings.
5) *Univariate or Multivariate:* Univariate algorithms discretize each attribute separately whereas multivariate ones take into account the relationships among all attributes when attempting to find the best complete set P. Due to the high complexity of the complete search space, multivariate approach has been less exploited than univariate.
6) *Direct Versus Incremental:* Incremental discretizers begin with a simple discretization and pass through an improvement process until an stopping criterion is reached whereas direct ones divide the range into k intervals simultaneously.
7) *Evaluation Measure:* Indicates which kind of metric uses the algorithm to compare solutions (information, statistical, rough sets, wrapper, binning, etc.).

A complete and more exhaustive taxonomy of these methods can be found in [5].

According to this categorization, we can classify our proposal as a static, multivariate, supervised, global, direct, and hybrid discretizer. Furthermore, as it uses an evaluation measure (explained in the next section) based on the classification error, we can also categorize it as a wrapper discretizer.

C. Genetic Algorithms

GAs have proved to perform well in many optimization problems [29]. They have been used specially in engineering, biology, and health care [30]–[33]. GAs are search heuristic methods that mimic the process of natural selection, being mainly inspired by evolutionary techniques such as inheritance, mutation, selection, and crossover [34].

Here, we describe the basic scheme of a GA algorithm and we present the GA chosen for solving cut points selection problem in discretization: the CHC model.

1) GA Basic Scheme: A GA starts with a population of I candidate solutions, called individuals or chromosomes. Let each individual be a $L-$dimensional vector $X_{i,G} = \{x_{i,G}^1, \ldots, x_{i,G}^N\}$, being the subsequent generations in GA denoted by $G = 0, \ldots, G_{\max}$. The initial population should cover the complete search space as much as possible, as it is randomly chosen. Afterwards, we evaluate this initial population using a fitness measure.

To improve the quality of the solutions, the GA process enters in a generational loop until the stopping criteria is

Algorithm 1 GA Algorithm Basic Scheme

```
Initialize the population
Evaluate initial population
while Stopping criteria is not reached do
    Select the best-fit individuals
    Apply the crossover
    Perform mutations
    Evaluate new population
    Replace least-fit individuals
end while
```

Algorithm 2 Reduction Process

```
Require: $n_r = 0, M_s = 1000, n_e > 0, M_e > 0$
if $C_s * (1 - R_{perc}) > M_s$ and $n_e/(M_e * R_{rate}) > n_r$ then
    $newPop \leftarrow REDUCTION(oldPop, C_p, points)$
    function REDUCTION($oldPop, C_p, points$)
        $C_s = |bc| * (1 - R_{perc})$
        $bc = getBestIndividual(oldPop)$
        $T = \emptyset$
        for $i = 0$ in $|C_p|$ do
            if $bc[i] == TRUE$ then
                $T = addPoint(T, points[i])$
            else
                $RP = addPoint(T, points[i])$
            end if
        end for
        $rem_s = C_s - |T|$
        if $rem_s > 0$ then
            if $|RP| > rem_s$ then
                $OP = orderByRanking(C_p, RP)$
                for $i = 0$ in $rem_s$ do
                    $T = addPoint(T, OP[i])$
                end for
            else
                $T = addAllPoints(T, RP)$
            end if
        end if
        $T = orderByValueAndAttribute(T)$
        $newPop = reduceChromosomes(oldPop, T)$
        $C_p = initializeCounter(T)$
    end function
    $newPop = restartPopulation(newPop)$
    $newPop = evaluatePopulation(newPop)$
    $n_r \leftarrow n_r + 1$
end if
```

reached. In each generation the fitness for each individual is evaluated. The best chromosomes of this generation are selected according to a selection probability (proportional to the fitness), forming the parent population. The selected parents are reproduced through the crossover operator to create a subsequent population, called the offspring population. To create diversity, some mutations are performed (according to a probability) on the offsprings, altering one or more gene values in a chromosome from its initial state. After evaluating these new individuals, the GA algorithms replace the least-fit solutions with the new best-fit ones. A basic scheme of a GA algorithm is represented in Algorithm 1.

2) CHC Algorithm: As a backbone of our EA, we have used the CHC model [35]. CHC is a classical evolutionary model thought for binary coding that tries to get a suitable tradeoff between a deep exploration of search space (diversity) and the ability of exploiting the local properties of the search to avoid a premature convergence (exploitation). CHC introduces several features to achieve this tradeoff, such as incest prevention or reinitialization.

During each generation the CHC develops the following steps.

a) It uses a parent population of the same size as the original to generate an intermediate population using

TABLE I
SUMMARY DESCRIPTION FOR CLASSIFICATION DATASETS

Data Set	#Ex.	#Atts.	#Num.	#Nom.	#Cl.
abalone	4,174	8	7	1	28
appendicitis	106	7	7	0	2
australian	690	14	8	6	2
autos	205	25	15	10	6
balance	625	4	4	0	3
banana	5,300	2	2	0	2
bands	539	19	19	0	2
banknote	1,372	5	5	0	2
bupa	345	6	6	0	2
cleveland	303	13	13	0	5
climate	540	18	18	0	2
contraceptive	1,473	9	9	0	3
crx	690	15	6	9	2
dermatology	366	34	34	0	6
ecoli	336	7	7	0	8
flare-solar	1,066	9	9	0	2
glass	214	9	9	0	7
haberman	306	3	3	0	2
hayes	160	4	4	0	3
heart	270	13	13	0	2
hepatitis	155	19	19	0	2
iris	150	4	4	0	3
mammographic	961	5	5	0	2
movement	360	90	90	0	15
newthyroid	215	5	5	0	3
pageblocks	5,472	10	10	0	5
penbased	10,992	16	16	0	10
phoneme	5,404	5	5	0	2
pima	768	8	8	0	2
saheart	462	9	8	1	2
satimage	6,435	36	36	0	7
segment	2,310	19	19	0	7
seismic	2,584	19	15	4	2
sonar	208	60	60	0	2
spambase	4,597	57	57	0	2
specfheart	267	44	44	0	2
tae	151	5	5	0	3
thoracic	470	17	3	14	2
titanic	2,201	3	3	0	2
transfusion	748	5	5	0	2
vehicle	846	18	18	0	4
vowel	990	13	13	0	11
wine	178	13	13	0	3
wisconsin	699	9	9	0	2
yeast	1,484	8	8	0	10

TABLE II
PARAMETERS OF THE DISCRETIZERS AND CLASSIFIERS

Method	Parameters
C4.5	pruned tree, confidence = 0.25, 2 examples per leaf
PART	confidence = 0.25, items per leaf = 2
PUBLIC	nodes between prune = 25
ChiMerge	confidence threshold = 0.05
FUSINTER	$\alpha = 0.975$, $\lambda = 1$
ECPSD	population = 50, M_e = 10,000, $\alpha = 0.5$
EMD	population = 50, M_e = 10,000, $\alpha = 0.7$, R_{rate} = 0.1, R_{perc} = 0.5

TABLE III
AVERAGE ACCURACY OBTAINED FOR C4.5

	Ameva		CAIM		ChiMerge		FUSINTER		MDLP		Modified Chi2		PKID		EMD	
	Avg	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*
abalone	0.2149	0.0213	0.2432	0.0179	0.2245	0.0106	0.2530	0.0176	**0.2537**	0.0225	0.1749	0.0255	0.2307	0.0216	0.2343	0.0215
appendicitis	0.8336	0.1255	0.8336	0.1255	0.8336	0.1099	0.8236	0.1159	0.8336	0.1099	0.7855	0.1264	0.8018	0.0277	**0.8700**	0.0715
australian	0.8667	0.0347	**0.8725**	0.0409	0.8565	0.0395	0.8681	0.0316	0.8638	0.0336	0.8580	0.0420	0.8493	0.0336	0.8536	0.0427
autos	0.7549	0.0566	0.7263	0.0864	0.7474	0.0778	**0.7937**	0.0598	0.7691	0.1029	0.7897	0.0992	0.7670	0.1052	0.7644	0.0739
balance	0.7456	0.0455	0.7472	0.0458	0.7777	0.0471	0.7282	0.0606	0.6992	0.0552	0.6640	0.0575	0.6482	0.0535	**0.8047**	0.0395
banana	0.7249	0.0206	0.6387	0.0158	0.6253	0.0189	**0.8791**	0.0113	0.7485	0.0220	0.6392	0.0180	0.7043	0.0184	0.8730	0.0155
bands	**0.6700**	0.0562	0.6458	0.0489	0.6346	0.0470	0.5975	0.0399	0.5378	0.0916	0.6642	0.0423	0.6197	0.0701	0.6494	0.0661
banknote	0.9096	0.0209	0.8892	0.0204	0.8863	0.0280	0.9584	0.0261	0.9446	0.0188	0.9526	0.0181	0.8484	0.0233	**0.9657**	0.0146
bupa	0.6807	0.0537	0.6065	0.0917	0.6361	0.0879	0.6198	0.0667	0.5715	0.0740	0.5704	0.0645	0.5789	0.0333	**0.6814**	0.0784
cleveland	0.5574	0.0741	0.5484	0.0707	0.5482	0.0724	**0.5640**	0.0541	0.5377	0.0533	0.5471	0.0970	0.5345	0.0535	0.5480	0.0652
climate	0.9278	0.0268	0.9278	0.0255	0.9000	0.0416	**0.9370**	0.0222	0.9333	0.0264	0.9167	0.0149	0.9148	0.0091	0.9333	0.0206
contraceptive	0.4909	0.0362	0.5105	0.0316	**0.5506**	0.0350	0.5221	0.0456	0.5045	0.0269	0.5045	0.0473	0.4875	0.0299	0.5261	0.0370
crx	0.8551	0.0520	0.8739	0.0387	0.8667	0.0379	**0.8797**	0.0474	0.8681	0.0389	0.8768	0.0470	0.8522	0.0492	0.8638	0.0344
dermatology	0.9535	0.0386	0.9318	0.0558	0.9424	0.0378	0.9508	0.0358	**0.9589**	0.0297	**0.9589**	0.0297	0.9454	0.0576	0.9482	0.0330
ecoli	0.7082	0.0558	0.7469	0.0494	0.7708	0.0400	0.7503	0.0688	**0.7771**	0.0586	0.7381	0.0414	0.6603	0.0690	0.7472	0.0772
flare	**0.6782**	0.0428	**0.6782**	0.0428	**0.6782**	0.0428	0.6735	0.0462	0.6754	0.0380	0.6754	0.0380	0.6754	0.0380	0.6726	0.0409
glass	0.5305	0.0513	0.6761	0.1190	0.6814	0.1015	0.6679	0.1201	**0.7579**	0.1053	0.6280	0.1303	0.5788	0.1159	0.7370	0.0988
haberman	0.7413	0.0607	**0.7512**	0.0601	0.7347	0.0506	0.7251	0.0353	0.7253	0.0338	0.7353	0.0100	0.7353	0.0100	0.7415	0.0365
hayes	**0.8020**	0.0716	**0.8020**	0.0716	**0.8020**	0.0716	0.7338	0.1154	0.5202	0.0812	0.7201	0.1325	0.7107	0.1298	0.7426	0.0954
heart	0.7889	0.0957	0.7852	0.1129	0.8037	0.0973	0.8074	0.1015	0.7963	0.0944	0.7889	0.0957	0.7926	0.0745	**0.8222**	0.0718
hepatitis	0.7821	0.1180	0.8271	0.0827	0.7883	0.0763	**0.8517**	0.0604	0.8325	0.0763	0.8008	0.0897	0.8258	0.0682	0.8071	0.0695
iris	0.9333	0.0314	0.9333	0.0314	0.9333	0.0314	0.9400	0.0378	0.9333	0.0314	0.9333	0.0444	0.9267	0.0663	**0.9533**	0.0427
mammographic	0.8159	0.0478	0.8284	0.0510	**0.8325**	0.0579	0.8263	0.0551	0.8315	0.0529	0.8200	0.0441	0.8117	0.0525	0.8315	0.0536
movement	0.4333	0.0830	0.4750	0.0769	0.3972	0.0945	0.6194	0.0869	0.6056	0.1029	0.6306	0.0818	0.3222	0.1033	**0.6361**	0.0519
newthyroid	0.9253	0.0508	0.9351	0.0502	0.9444	0.0481	0.9165	0.0478	0.9444	0.0421	0.9398	0.0443	0.9398	0.0309	**0.9494**	0.0433
pageblocks	0.9656	0.0052	0.9618	0.0050	0.9666	0.0045	0.9618	0.0060	0.9684	0.0060	0.9474	0.0107	0.9496	0.0066	**0.9693**	0.0065
penbased	0.9293	0.0058	0.8877	0.0135	0.8882	0.0074	0.8947	0.0105	0.8866	0.0129	0.8904	0.0119	0.6700	0.0061	**0.9491**	0.0064
phoneme	0.7877	0.0196	0.7913	0.0181	0.7820	0.0205	0.8205	0.0198	0.8124	0.0225	0.7533	0.0140	0.7681	0.0184	**0.8447**	0.0174
pima	**0.7448**	0.0327	0.7345	0.0528	0.7292	0.0256	0.7318	0.0495	0.7344	0.0425	0.7203	0.0468	0.7307	0.0589	0.7435	0.0175
saheart	0.6991	0.0499	0.7055	0.0394	**0.7080**	0.0504	0.6450	0.0404	0.6817	0.0555	0.6971	0.0462	0.6580	0.0136	0.6839	0.0513
satimage	0.2507	0.0055	0.8541	0.0130	**0.8637**	0.0123	0.8449	0.0160	0.8454	0.0155	0.8387	0.0139	0.8020	0.0091	0.8483	0.0125
segment	0.9537	0.0112	0.9468	0.0131	0.9524	0.0114	0.9502	0.0143	0.9385	0.0145	0.8831	0.0223	0.8476	0.0179	**0.9606**	0.0076
seismic	**0.9342**	0.0001	**0.9342**	0.0001	**0.9342**	0.0001	**0.9342**	0.0001	**0.9342**	0.0001	**0.9342**	0.0001	**0.9342**	0.0001	0.9334	0.0024
sonar	**0.7969**	0.1188	0.7400	0.0663	0.7455	0.0897	0.6967	0.1139	0.7638	0.1194	0.7398	0.1105	0.6962	0.1068	0.7731	0.0965
spambase	0.9354	0.0120	0.9356	0.0128	**0.9384**	0.0072	0.9206	0.0172	0.9273	0.0130	0.8764	0.0162	0.8869	0.0138	0.9252	0.0113
specfheart	0.8050	0.0680	0.7785	0.0691	0.7829	0.0514	0.7496	0.0502	0.7268	0.0979	0.7752	0.0667	0.7942	0.0175	**0.8204**	0.0211
tae	0.4454	0.1982	0.4583	0.1639	**0.5433**	0.1406	0.5113	0.1691	0.3442	0.0236	0.5296	0.1240	0.4708	0.1293	0.5317	0.1230
thoracic	0.8468	0.0085	0.8468	0.0085	0.8468	0.0085	0.8468	0.0085	0.8468	0.0085	**0.8511**	0.0000	**0.8511**	0.0000	0.8468	0.0085
titanic	0.7733	0.0304	0.7774	0.0315	0.7733	0.0306	0.7760	0.0296	0.7715	0.0290	0.7760	0.0296	0.7892	0.0231	**0.7906**	0.0221
transfusion	**0.7927**	0.0467	0.7727	0.0248	0.7580	0.0072	0.7619	0.0502	0.7621	0.0041	0.7621	0.0041	0.7621	0.0041	0.7728	0.0454
vehicle	0.6773	0.0417	0.6739	0.0415	0.6667	0.0525	0.6832	0.0437	**0.6832**	0.0509	0.6831	0.0559	0.6454	0.0447	0.6831	0.0367
vowel	0.7283	0.0568	0.6960	0.0335	0.6677	0.0578	0.7253	0.0538	**0.7323**	0.0643	0.7121	0.0552	0.4848	0.0429	0.7071	0.0503
wine	**0.9382**	0.0486	0.9101	0.0553	0.9042	0.0593	0.9379	0.0419	0.8984	0.0798	0.9268	0.0648	0.7974	0.1109	0.9212	0.0724
wisconsin	0.9371	0.0204	0.9385	0.0192	**0.9514**	0.0215	0.9456	0.0349	0.9442	0.0281	0.9471	0.0309	0.9384	0.0272	0.9499	0.0148
yeast	0.5499	0.0330	0.5304	0.0420	0.5021	0.0213	0.5674	0.0361	**0.5722**	0.0316	0.4434	0.0306	0.3862	0.0341	0.5243	0.0349
MEAN	*0.7515*	*0.0486*	*0.7624*	*0.0486*	*0.7622*	*0.0463*	*0.7732*	*0.0492*	*0.7600*	*0.0476*	*0.7556*	*0.0497*	*0.7250*	*0.0451*	***0.7852***	*0.0434*

crossover operator. Then, this is randomly paired to generate a new offspring population.

b) After that, a survival competition is held where the best chromosomes from the parent and offspring populations are selected to obtain a new population with the original fixed size.

CHC also implements half uniform crossover (HUX), a special and heterogeneous crossover operator for chromosomes. HUX recombination produces two offspring which are maximally distant from their two parent vectors by exchanging half of the bits that differ in the two parents, using the Hamming distance. To prevent incest in the chromosomes, only those parents who differ from each other by some number of bits (mating threshold) are mated. The initial threshold T is set at $L/4$, where L is the length of the chromosomes. If no offspring are inserted into the new population then the threshold is reduced by one.

Instead of the mutation operation of GAs, CHC includes a reseeding process to unblock the search process when it is stuck. CHC applies the reinitialization mechanism if the population remains unchanged during an specified number of evaluations (i.e., the difference threshold has dropped to zero and no new offspring generated is better than any chromosome from parent population). Reseeding of the population is accomplished by randomly changing 35% of the genes in the template chromosome (the best previous solution) to form each new chromosome in the population; resuming the search after that. In this manner, the algorithm introduces new diversity although maintaining some of exploitation, conserving the best solution found over the course of the previous search as a template.

III. Evolutionary Multivariate Discretizer

In this section, we describe EMD, an evolutionary-based algorithm with binary representation for discretization. EMD uses a wrapper fitness function based on a tradeoff between the classification error provided by the straightforward application of two classifiers, and the number of intervals produced.

Despite it being well-known that multivariate approaches improve the discretization process on supervised learning, they have been less exploited in the literature. The proposed algorithm follows a multivariate approach, taking advantage of the existing interactions and dependencies among the attributes to improve the underlying discretization process. Furthermore, to tackle larger problems and, in general, to speed-up the discretization process on all datasets; we include a chromosome reduction mechanism.

A. Representation

Classically, an individual is represented in a binary array of 0's and 1's. Let us define the search space of the problem

TABLE IV
AVERAGE ACCURACY OBTAINED FOR NAIVE BAYES

	Ameva		**CAIM**		**ChiMerge**		**FUSINTER**		**MDLP**		**Modified Chi2**		**PKID**		**EMD**	
	Avg	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*
abalone	0.2127	0.0312	0.2585	0.0152	0.2518	0.0203	0.2482	0.0229	0.2496	0.0172	0.2429	0.0188	**0.2611**	0.0225	0.2578	0.0239
appendicitis	**0.8800**	0.0958	0.8709	0.1061	0.8518	0.1153	0.8418	0.1167	0.8709	0.0970	0.8518	0.1071	0.8609	0.0989	0.8709	0.1006
australian	0.8507	0.0399	**0.8638**	0.0464	0.8507	0.0483	0.8580	0.0414	0.8449	0.0432	0.8449	0.0416	0.8551	0.0368	0.8565	0.0346
aut	0.6729	0.1146	0.6497	0.0873	0.6603	0.1121	0.6580	0.0960	0.6732	0.1180	0.6488	0.1186	**0.7269**	0.1112	0.6606	0.0671
bal	0.7968	0.0407	0.8031	0.0423	0.8273	0.0451	0.8671	0.0380	0.7266	0.0653	0.9088	0.0150	**0.9120**	0.0133	0.8544	0.0381
banana	0.7049	0.0234	0.6049	0.0272	0.5836	0.0308	0.7162	0.0192	0.7247	0.0222	0.6300	0.0199	0.7147	0.0206	**0.7357**	0.0199
bands	**0.7255**	0.0431	0.6643	0.0394	0.6715	0.0453	0.7013	0.0555	0.5045	0.0907	0.7235	0.0618	0.6865	0.0704	0.6550	0.0552
banknote	0.8943	0.0224	0.8907	0.0231	0.8746	0.0287	0.9103	0.0191	0.9205	0.0155	0.9140	0.0233	0.9220	0.0227	**0.9424**	0.0143
bupa	**0.6599**	0.1229	0.6169	0.0891	0.6130	0.0898	0.6427	0.0931	0.5715	0.0740	0.6376	0.0460	0.6277	0.0986	0.6548	0.0894
cleveland	**0.5778**	0.0453	0.5612	0.0758	0.5449	0.0899	0.5580	0.0585	0.5545	0.0407	0.5482	0.0724	0.5544	0.0772	0.5709	0.0717
climate	0.9111	0.0161	0.9167	0.0171	0.9111	0.0111	0.9185	0.0251	**0.9352**	0.0252	0.9130	0.0287	0.9019	0.0249	**0.9352**	0.0238
contraceptive	0.5064	0.0473	0.4963	0.0289	**0.5255**	0.0291	0.5132	0.0291	0.5051	0.0484	0.5024	0.0315	0.5092	0.0283	0.5200	0.0344
crx	0.8551	0.0478	0.8609	0.0428	**0.8638**	0.0433	0.8449	0.0438	0.8565	0.0480	0.8420	0.0309	0.8522	0.0360	0.8536	0.0589
dermatology	0.9810	0.0223	0.9755	0.0299	0.9755	0.0299	0.9783	0.0279	0.9782	0.0214	**0.9865**	0.0191	0.9782	0.0251	0.9482	0.0330
ecoli	0.8127	0.0706	0.8094	0.0468	0.8213	0.0423	**0.8332**	0.0596	0.8216	0.0616	0.7948	0.0642	0.8038	0.0602	0.7889	0.0563
flare-solar	0.6557	0.0494	0.6557	0.0494	0.6539	0.0506	0.6576	0.0473	**0.6754**	0.0380	0.6529	0.0494	0.6548	0.0493	0.6726	0.0409
glass	0.4647	0.0661	0.7034	0.1411	0.6736	0.1033	0.6878	0.1183	0.7206	0.0838	0.7199	0.0824	**0.7246**	0.0983	0.7124	0.1234
haberman	**0.7478**	0.0674	0.7352	0.0456	0.7251	0.0355	0.7382	0.0404	0.7285	0.0370	0.7220	0.0451	0.7282	0.0545	0.7449	0.0585
hayes-roth	0.7437	0.0973	0.7437	0.0973	0.7586	0.1340	0.7273	0.1269	0.5202	0.0812	0.7937	0.1429	0.7932	0.1268	**0.8157**	0.1115
heart	0.8370	0.0859	0.8407	0.0721	0.8370	0.0927	**0.8444**	0.0887	0.8407	0.0891	0.8296	0.0943	**0.8444**	0.0716	0.8296	0.0831
hepatitis	0.8196	0.0964	0.8388	0.1037	0.8383	0.1441	**0.8646**	0.0902	0.8383	0.1162	0.8263	0.1240	0.8071	0.1169	0.8254	0.1021
iris	0.9333	0.0444	0.9400	0.0492	0.9400	0.0378	0.9467	0.0526	0.9267	0.0378	0.9333	0.0444	0.9200	0.0422	**0.9533**	0.0427
mammographic	0.8148	0.0438	0.8262	0.0404	0.8086	0.0424	0.8315	0.0560	0.8221	0.0446	0.8263	0.0519	0.8325	0.0548	**0.8357**	0.0534
movement_libras	0.6500	0.0657	0.6528	0.0575	0.6611	0.0876	0.6333	0.0738	0.6056	0.0568	0.6250	0.0799	**0.6667**	0.0490	0.5583	0.0891
newthyroid	0.9535	0.0318	0.9582	0.0416	0.9673	0.0450	**0.9725**	0.0319	0.9489	0.0413	0.9582	0.0468	0.9675	0.0317	0.9494	0.0433
page-blocks	0.9401	0.0077	0.9342	0.0065	0.9344	0.0096	0.9317	0.0080	0.9311	0.0094	0.9375	0.0083	0.9158	0.0096	**0.9406**	0.0087
penbased	0.8608	0.0091	0.8712	0.0077	0.8763	0.0074	0.8739	0.0099	0.8766	0.0097	**0.8771**	0.0087	0.8722	0.0084	0.8708	0.0088
phoneme	0.7889	0.0195	0.7894	0.0182	0.7794	0.0222	0.7737	0.0190	0.7689	0.0214	0.7705	0.0133	0.7742	0.0218	**0.7935**	0.0232
pima	0.7280	0.0434	0.7320	0.0604	0.7372	0.0441	0.7475	0.0529	0.7526	0.0377	0.7397	0.0470	0.7410	0.0504	**0.7722**	0.0340
saheart	0.6582	0.0554	0.7035	0.0480	**0.7249**	0.0541	0.6451	0.0627	0.6624	0.0578	0.6777	0.0794	0.6756	0.0578	0.7079	0.0325
satimage	0.2528	0.0065	0.8169	0.0160	0.8183	0.0130	0.8194	0.0130	0.8210	0.0131	**0.8222**	0.0148	0.8211	0.0142	0.8199	0.0156
segment	0.9126	0.0108	0.9039	0.0119	0.9104	0.0096	0.9030	0.0139	0.9104	0.0159	0.8987	0.0220	0.8909	0.0274	**0.9355**	0.0126
seismic	0.8224	0.0276	0.8196	0.0287	0.8293	0.0305	0.8196	0.0236	0.8200	0.0224	0.8580	0.0162	0.8247	0.0168	**0.9334**	0.0024
sonar	0.7788	0.0910	0.7745	0.0830	0.7600	0.1116	0.7407	0.1088	0.7688	0.1268	**0.7836**	0.0825	0.7452	0.1403	0.7393	0.1033
spambase	0.8995	0.0160	0.8938	0.0118	0.8962	0.0155	0.8971	0.0140	0.8989	0.0140	0.9021	0.0143	0.8945	0.0159	**0.9228**	0.0152
spectfheart	0.7644	0.0865	0.7682	0.0962	0.7345	0.0895	0.7756	0.0885	0.7305	0.0895	0.7496	0.0917	0.7752	0.0840	**0.8128**	0.0372
tae	0.5113	0.1538	0.4904	0.1735	0.5046	0.1573	0.4771	0.1539	0.3442	0.0236	**0.5571**	0.1084	0.4904	0.1763	0.5438	0.1092
thoracic	0.8191	0.0395	0.8213	0.0272	0.8191	0.0372	0.8255	0.0340	0.8213	0.0304	**0.8298**	0.0381	0.8043	0.0528	0.8213	0.0304
titanic	0.7810	0.0302	0.7783	0.0297	**0.7833**	0.0324	0.7788	0.0302	0.7760	0.0322	0.7788	0.0302	0.7788	0.0302	0.7833	0.0307
transfusion	**0.7687**	0.0665	0.7633	0.0220	0.7446	0.0445	0.7566	0.0653	0.7500	0.0479	0.7500	0.0298	0.7499	0.0528	0.7406	0.0434
vehicle	0.6122	0.0464	0.6064	0.0367	0.5934	0.0371	0.6265	0.0362	0.5910	0.0350	0.6241	0.0361	0.6205	0.0276	**0.6419**	0.0462
vowel	0.6364	0.0343	0.6222	0.0486	**0.6677**	0.0445	0.6313	0.0313	0.6030	0.0513	0.6515	0.0432	0.5788	0.0340	0.6343	0.0451
wine	0.9830	0.0274	0.9775	0.0392	0.9830	0.0274	0.9663	0.0391	**0.9886**	0.0241	0.9598	0.0779	0.9663	0.0470	0.9212	0.0724
wisconsin	0.9671	0.0191	0.9671	0.0191	0.9700	0.0196	**0.9742**	0.0221	0.9728	0.0218	0.9714	0.0213	0.9728	0.0218	0.9513	0.0216
yeast	0.5695	0.0305	0.5796	0.0419	**0.5944**	0.0259	0.5789	0.0312	0.5695	0.0325	0.5620	0.0397	0.5519	0.0387	0.5735	0.0396
MEAN	*0.7493*	*0.0501*	*0.7634*	*0.0498*	*0.7634*	*0.0531*	*0.7675*	*0.0518*	*0.7494*	*0.0474*	*0.7684*	*0.0507*	*0.7678*	*0.0526*	***0.7747***	*0.0489*

as all the possible discretization schemes offered by their set of BP (presented in Section II-A). In this way, the common binary encoding is suitable to represent these schemes so that each BP corresponds to a gene. A chromosome thus consists of |BP| genes with two possible values: 0, if the associated cut point is not included in P; and 1, if it is. The chromosome structure associated with the BPs approach is presented in Fig. 2 and an example of a possible solution along with its spatial representation are depicted in Fig. 3.

B. Fitness Function

To select the most appropriate discretization scheme from the population, it is necessary to define a suitable fitness function to evaluate the solutions. Let Q be a subset of cut points selected from BP and be coded by a chromosome. We define a fitness function as the aggregation of two objectives, namely the classification error of the discretized data and the minimization of the number of cut points

$$\text{Fitness}(Q) = \alpha \cdot \frac{|Q|}{|\text{BP}|} + (1 - \alpha) \cdot \Delta \qquad (2)$$

where $|Q|$ is the number of boundary cut points currently selected in the chromosome, |BP| is the total number of cut points, Δ is the classification error of the discretized data, and α is the weight factor specified as input parameter. With this fitness design, we can obtain more effective discretization schemes (through the classifier evaluation), while maintaining solutions as simple as possible.

The classification error is a supervised measure used to compute the number of misclassified instances. In our case, the classification error is computed as the aggregated error obtained using two classifiers [22]: 1) an unpruned version of C4.5 [36] and 2) Naive Bayes [37]. In this manner, the algorithm has several criteria to evaluate the solutions. This part is defined as follows:

$$\Delta = \frac{\delta_{C45} + \delta_{\text{NB}}}{2} \qquad (3)$$

where δ_{C45} is the total number of instances misclassified by C4.5 divided by the total number of instances and δ_{NB} represents the same computation for Naive Bayes. This measure is basically the arithmetic mean of the classification error committed by these two classifiers with the domain [0, 1].

We have decided to use a wrapper fitness function because of its promising accuracy results which outperform the results in [19] (with an inconsistency-based function). Moreover, this measure guarantees solutions that are simpler than those based on inconsistency.

Regarding the time complexity of our approach, this is mainly conditioned by the number of cut points (L) and the number of instances (I) of the data set, as well as, the maximum number of evaluations (E) until the stopping criterion is reached. According to these variables, we can state that

TABLE V
AVERAGE CUT POINTS

	Ameva		**CAIM**		**ChiMerge**		**FUSINTER**		**MDLP**		**Modified Chi2**		**PKID**		**EMD**	
	Avg	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*
abalone	190.00	0.00	190.00	0.00	148.30	3.37	**26.30**	0.95	38.10	1.10	2,642.20	570.94	384.60	0.52	104.70	16.26
appendicitis	8.30	0.48	8.00	0.00	8.00	0.00	16.70	2.41	7.00	0.47	35.00	6.27	64.00	0.00	**2.80**	0.40
australian	19.10	1.52	15.00	0.00	13.80	0.92	56.30	1.83	11.20	0.92	98.60	43.88	141.00	0.94	**6.40**	1.28
autos	79.30	1.89	76.20	0.42	75.80	0.42	42.40	3.37	30.80	3.22	27.20	4.54	183.80	3.33	**5.70**	1.35
balance	9.00	0.00	9.00	0.00	9.00	0.00	12.50	0.53	**5.20**	0.42	15.60	0.52	17.00	0.00	9.00	1.00
banana	4.10	0.32	**3.00**	0.00	**3.00**	0.00	25.30	2.71	11.10	0.32	2,088.40	21.37	137.00	0.00	15.00	3.13
bands	34.80	2.82	20.00	0.00	19.80	0.42	78.70	4.22	**3.30**	0.82	53.40	7.82	235.40	2.17	16.60	1.69
banknote	8.00	0.89	**4.00**	0.00	**4.00**	0.00	12.20	1.47	42.70	4.56	10.30	5.98	136.00	0.00	7.40	2.06
bupa	10.70	1.16	7.00	0.00	7.00	0.00	33.10	3.03	**1.80**	0.42	111.80	41.02	91.10	0.32	10.50	0.81
cleveland	53.10	0.32	37.90	0.32	33.10	0.88	16.40	0.70	**8.10**	0.32	34.70	13.90	90.00	2.79	11.00	0.89
climate	41.30	3.77	18.00	0.00	13.80	1.99	3.80	0.40	13.10	1.70	**1.00**	1.00	378.00	0.00	4.10	1.14
contraceptive	19.00	0.00	16.00	0.00	16.00	0.00	23.80	1.14	**9.10**	0.88	41.20	2.35	53.00	0.00	22.50	2.16
crx	9.40	1.07	7.00	0.00	6.50	0.53	44.80	3.33	5.50	0.71	92.20	73.68	114.20	0.63	**2.80**	1.33
dermatology	171.00	0.00	101.90	0.32	72.90	1.20	46.40	0.97	33.80	1.48	37.40	3.10	107.80	0.42	**8.10**	0.54
ecoli	49.30	2.21	37.90	0.32	37.20	0.63	11.70	0.48	13.00	0.82	62.50	16.41	82.00	0.00	**9.00**	1.10
flare	10.20	0.42	10.00	0.00	10.00	0.00	7.50	0.85	3.90	0.32	13.90	0.99	13.00	0.00	**2.10**	0.30
glass	55.00	0.00	55.00	0.00	52.80	1.03	14.10	1.45	14.40	0.97	51.10	12.84	96.40	1.07	**10.70**	0.78
haberman	6.60	1.07	4.00	0.00	4.00	0.00	11.20	1.62	**2.00**	0.00	46.60	1.78	36.00	0.00	3.80	1.25
hayes	9.00	0.00	9.00	0.00	7.00	0.00	7.30	0.48	**4.00**	0.00	8.30	0.67	12.00	0.00	6.90	0.30
heart	15.20	0.42	14.00	0.00	13.00	0.00	35.80	2.94	9.90	0.32	42.20	17.18	86.60	0.70	**5.00**	1.26
hepatitis	22.90	1.20	20.00	0.00	19.00	0.82	22.40	2.55	8.90	1.10	15.40	4.86	72.30	0.48	**3.90**	0.94
iris	9.00	0.00	9.00	0.00	9.00	0.00	10.10	0.99	8.20	1.03	27.30	8.86	44.60	0.52	**2.90**	0.30
mammographic	6.10	0.32	6.00	0.00	6.00	0.00	12.60	0.84	7.70	0.67	47.50	2.68	43.00	0.00	**4.40**	0.66
movement	1,263.00	1.56	1,264.60	1.35	1,247.10	4.36	167.70	4.06	131.20	5.92	157.80	16.25	1,531.00	0.00	**22.40**	2.25
newthyroid	11.00	0.00	11.00	0.00	11.00	0.00	14.60	0.97	13.20	0.92	16.30	6.13	65.70	0.48	**3.50**	0.81
pageblocks	41.00	0.00	41.00	0.00	41.00	0.00	31.10	1.29	57.60	2.32	715.80	138.09	629.00	0.00	**18.60**	1.80
penbased	145.00	0.00	145.10	0.32	145.00	0.00	132.80	1.99	148.00	2.40	144.90	1.85	1,073.40	0.97	**43.10**	4.85
phoneme	7.00	0.00	**6.00**	0.00	**6.00**	0.00	53.50	1.96	27.40	1.26	669.60	208.26	336.90	0.32	26.00	3.97
pima	11.70	0.67	**9.00**	0.00	**9.00**	0.00	38.10	1.97	9.90	0.57	91.20	23.94	161.50	0.71	11.30	2.00
saheart	12.70	1.57	9.00	0.00	9.00	0.00	50.80	4.61	**6.10**	0.88	54.10	13.04	145.70	0.95	7.40	1.36
satimage	217.00	0.00	217.00	0.00	217.00	0.00	320.00	4.81	397.10	4.89	323.00	4.00	1,746.20	3.65	**38.30**	4.22
segment	109.00	0.00	101.90	0.32	100.90	0.32	137.30	2.00	151.70	2.41	803.70	378.68	723.00	0.00	**16.50**	1.91
seismic	15.90	1.92	11.00	0.00	9.80	0.40	7.80	0.60	13.40	0.80	3.70	0.90	242.30	0.46	**0.30**	0.90
sonar	103.70	2.95	61.00	0.00	60.80	0.42	356.50	6.49	21.90	2.56	20.30	2.50	772.70	1.06	**8.50**	1.12
spambase	62.00	0.00	58.00	0.00	58.00	0.00	189.30	4.40	100.70	2.91	5,039.40	1,596.54	797.30	2.83	**20.20**	1.54
specfheart	63.40	3.41	45.00	0.00	44.90	0.32	124.30	6.31	21.90	2.02	30.90	5.22	616.90	16.66	**4.90**	1.30
tae	15.70	1.42	9.00	0.00	9.00	0.00	15.20	1.03	**1.00**	0.00	70.40	12.38	33.10	0.32	9.40	1.20
thoracic	5.70	1.62	3.00	0.00	3.00	0.00	**0.00**	0.00	7.00	1.00	6.00	2.32	60.00	0.00	0.60	0.66
titanic	**4.00**	0.00	**4.00**	0.00	**4.00**	0.00	5.00	0.00	**4.00**	0.00	5.00	0.00	6.00	0.00	**4.00**	0.00
transfusion	5.90	0.70	4.00	0.00	3.60	0.49	**3.00**	0.00	9.70	1.85	41.90	2.47	61.60	0.80	6.60	1.11
vehicle	55.30	0.48	55.00	0.00	54.50	0.53	90.60	2.63	52.50	2.59	152.90	40.10	398.70	0.67	**18.30**	2.10
vowel	131.00	0.00	113.00	0.00	101.00	0.00	38.00	1.56	**30.50**	2.64	66.20	14.01	307.00	0.00	40.10	5.50
wine	27.00	0.00	27.00	0.00	26.90	0.32	44.20	2.04	22.80	0.92	13.80	0.42	157.00	0.00	**3.80**	0.98
wisconsin	10.00	0.00	10.00	0.00	10.00	0.00	22.60	0.84	20.00	1.15	20.40	1.71	71.20	0.79	**5.00**	0.89
yeast	73.10	0.32	58.00	0.00	56.20	0.63	12.40	0.70	**11.90**	1.20	155.80	34.43	161.10	0.57	25.00	3.16
MEAN	*71.79*	*0.81*	*65.34*	*0.07*	*62.59*	*0.44*	*53.92*	*1.99*	*34.50*	*1.42*	*315.71*	*74.80*	*282.58*	*1.00*	***13.54***	*1.88*

the time complexity of the whole algorithm strongly depends on the evaluation phase, which in turn depends on the time complexity of the two classifiers used in the wrapper function.

Naive Bayes classifier provides an efficient time complexity $O(LI)$ similar to the consistency measures presented in [21]. For C4.5, it is known that the tree pruning introduces a time complexity of $O(LIlog(I)^2)$ [36]. To mitigate it, we use an unpruned C4.5 version that improves the time complexity $(O(LIlog(I)))$ and becomes it in a evaluation measure more efficient and sensitive to the error. Therefore, EMD obtains a time complexity $O(ELIlog(I))$.

Despite our algorithm is less efficient than other measures [20], we can assure that the wrapper function runs in an acceptable time complexity. Additionally, through the chromosome reduction mechanism L is continuously reduced, thus decreasing the time complexity aforementioned.

The objective of the GA is to minimize the fitness function defined; therefore, to obtain consistent and simple discretization schemes with the minimum possible number of cut points, but always keeping a fair classification accuracy through the error counterpart.

C. Reduction

Whenever an EA faces problems with a significant size, its application can become unsatisfying due to the growth of the chromosomes. A great number of attributes and examples cause an increment in complexity because the number of BPs represented increases.

As in any optimization problem, the cut points selection problem offers multiple solutions (local optima) which can be considered as valid, although not the best ones (global optima) [38]. Agreeing that the size of the problem becomes troublesome, it is desirable to reach some local optima to avoid a long delay or even inability in obtaining the global optima. In our case, we propose a chromosome reduction process to speed-up the convergence by reducing the number of boundary cut points to be considered. For that, in each generation, we only preserve the most selected points in previous evaluations (for each reduction stage).

The pseudo-code of the reduction process is described in Algorithm 2 and applied at the beginning of each generation. Given a counter of the times each point is selected C_p, the current best chromosome bc, the current size of the chromosomes C_s (initially equal to |BP|), the number of reductions accomplished n_r, the maximum size for chromosomes M_s, the number of evaluations accomplished up to now n_e, the maximum number of evaluations M_e, the reduction rate R_{rate}, and the reduction percentage R_{perc}. EMD applies a distinct reduction process each $M_e * R_{\text{rate}}$ evaluations, if and only if the maximum size allowed M_s is not exceeded.

The following steps are described as follows.

1) Calculate the new reduced size for the chromosomes: $|\text{bc}| * (1 - R_{\text{perc}})$.

TABLE VI
WILCOXON TEST RESULTS FOR ACCURACY (C45 AND NAIVE BAYES) AND NUMBER OF CUT POINTS

Algorithms	C4.5		Naive Bayes		# Cut Points	
	+	±	+	±	+	±
Ameva	1	6	0	6	2	3
CAIM	1	6	0	6	3	4
ChiMerge	1	6	0	7	4	6
FUSINTER	2	6	0	7	2	5
MDLP	1	6	0	6	5	6
Modified Chi2	1	5	0	7	1	1
PKID	0	0	0	6	0	0
EMD	7	7	4	7	7	7

TABLE VII
AVERAGE RANKINGS OF THE ALGORITHMS (FRIEDMAN PROCEDURE + ADJUSTED p-VALUE WITH HOLM'S TEST) FOR ACCURACY (C4.5 AND NAIVE BAYES)

C4.5			Naive Bayes		
Algorithms	Ranking	p_{Holm}	Algorithms	Ranking	p_{Holm}
EMD	3.04	—	EMD	3.61	—
FUSINTER	3.98	0.0707	FUSINTER	4.11	0.3329
Ameva	4.32	0.0445	Ameva	4.53	0.1725
MDLP	4.33	0.0445	Mod. Chi2	4.62	0.1725
CAIM	4.36	0.0445	PKID	4.66	0.1725
ChiMerge	4.43	0.0358	ChiMerge	4.76	0.1354
Modified Chi2	5.13	3.14e-4	CAIM	4.79	0.1354
PKID	6.4	~ 0	MDLP	4.92	0.0778

2) Initialize the set of new points T with all the points marked as 1 in the best previous chromosome bc. The remaining points are ordered (by ranking) and stored in *RP*.
3) According to the counter C_p, add the most selected points from the rest of the individuals to the new structure until completing the new chromosome size C_s.
4) Once completed T, we reorder the points by value also considering the original order of the attributes, form the new chromosome structure (as in Fig. 2).
5) Apply the new structure to each individual, remove all nonselected points.
6) Reinitialize the counter C_p according to the new structure.
7) Restart the new population newPop using the best previous solution bc as reference.
8) Evaluate the new population.

IV. EXPERIMENTAL FRAMEWORK AND RESULTS

Next, we describe the methodology followed in the experimental study which compares the proposed technique with the best discretization algorithms in the experimental review [5] using a wide range of classifiers. Here, our method is denoted by EMD.

This section is divided in three parts: firstly, we test EMDs performance in terms of accuracy and number of cut points using the classifiers included in the wrapper function: C4.5 [36] and Naive Bayes [10], [37]. Secondly, we prove EMDs classification ability using two classifiers (PART [39] and PUBLIC [40]) not included in its fitness function in order to test the quality and generalization capabilities of the discretized schemes. Finally, we compare our algorithm with the approach in which it is inspired [19] (denoted by evolutionary cut points selection discretizer (ECPSD)) in terms of accuracy (using C4.5 and Naive Bayes) and time complexity.

TABLE VIII
AVERAGE RANKINGS OF THE ALGORITHMS (FRIEDMAN PROCEDURE + ADJUSTED p-VALUE WITH HOLM'S TEST) FOR NUMBER OF CUT POINTS

Algorithms	Ranking	p_{Holm}
EMD	1.99	—
MDLP	3.13	0.0267
ChiMerge	3.40	0.0130
CAIM	4.08	1.57e-4
FUSINTER	4.72	~ 0
Ameva	5.09	~ 0
Modified Chi2	5.86	~ 0
PKID	7.73	~ 0

A. Experimental Framework

The discretizers involved in the comparison for the first two parts are: Ameva [41], class-atribute interdependence maximization (CAIM) [27], ChiMerge [28], FUSINTER [42], minimum description length principle (MDLP) [43], Mod-Chi2 [44], and proportional k-interval discretizatio (PKID) [10]. Implementations of the discretizers as well as the classifiers used in these experiments can be found under the KEEL DM tool [45].

Performance of the algorithms is analyzed by using 45 datasets taken from the UCI machine learning database repository [46]. The datasets considered are partitioned using the ten fold cross-validation procedure. Table I gives a summary of datasets used in our experiments. For each data set, the number of examples (#Ex.), the total number of attributes (#Atts.), some of which could be numerical (#Num.) or nominal (#Nom.), and the number classes (#Cl) are shown.

The recommended parameters of the discretizers and the classifiers, according to their authors' specification [5], [19], [39], are specified in Table II; only for those methods that require them.

For our method, GA parameters (population size and number of evaluations) has been established to standard values that have been demonstrated to perform well in most of cases where GAs have been applied on data preprocessing tasks [1]. The alpha weight factor is set to 0.7 in favor to the removal of points for two reasons: firstly, the more points removed during the cycle, the faster the algorithm converges. On the other hand, the selection of candidate points is a multimodal problem in which many subset of points achieve similar performances, making more difficult the reduction subobjective.

B. Analysis and Empirical Results

1) Wrapper Classifiers Comparison: To compare the performance of these methods, test classification accuracy

TABLE IX
AVERAGE ACCURACY OBTAINED FOR PART

	Ameva		**CAIM**		**ChiMerge**		**FUSINTER**		**MDLP**		**Modified Chi2**		**PKID**		**EMD**	
	Avg	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*
abalone	**0.2130**	0.0253	0.1689	0.0041	0.1739	0.0111	0.1761	0.0121	0.1679	0.0042	0.1670	0.0042	0.1658	0.0019	0.1694	0.0053
appendicitis	0.8236	0.1172	0.7664	0.0894	0.8427	0.1290	0.7764	0.1445	0.8527	0.1291	0.7918	0.0727	0.8018	0.0263	**0.8709**	0.1085
australian	0.6116	0.1133	**0.6217**	0.1085	0.5551	0.0066	0.5797	0.0449	0.5551	0.0066	0.5623	0.0303	0.5551	0.0066	0.5551	0.0066
autos	0.3865	0.0839	0.4913	0.1070	0.5166	0.1176	0.3891	0.0951	0.4998	0.1317	**0.5336**	0.1460	0.4536	0.0766	0.5251	0.0973
balance	0.6893	0.0684	0.6877	0.0665	0.7134	0.0705	0.5842	0.1029	0.6493	0.0519	0.5797	0.0934	0.5505	0.0465	**0.7298**	0.0809
banana	0.7234	0.0199	0.6387	0.0150	0.6253	0.0179	0.5615	0.0286	**0.7451**	0.0207	0.5917	0.0265	0.5574	0.0081	0.6694	0.1101
bands	0.5770	0.0103	0.5788	0.0295	**0.5977**	0.0592	0.5807	0.0084	0.5323	0.0825	0.5789	0.0076	0.5789	0.0162	0.5770	0.0289
banknote	0.8070	0.0961	0.6363	0.0268	0.6573	0.1318	0.7005	0.1130	0.7392	0.0671	0.7798	0.0602	0.5554	0.0013	**0.8368**	0.0554
bupa	0.5589	0.0613	**0.6122**	0.0795	0.5735	0.0809	0.5761	0.0334	0.5715	0.0702	0.5789	0.0407	0.5675	0.0323	0.5930	0.0816
cleveland	0.5412	0.0317	**0.5444**	0.0399	0.5412	0.0381	0.5412	0.0381	0.5412	0.0381	0.5378	0.0372	0.5412	0.0381	0.5412	0.0381
climate	0.9093	0.0268	**0.9148**	0.0091	**0.9148**	0.0091	**0.9148**	0.0091	**0.9148**	0.0091	**0.9148**	0.0091	**0.9148**	0.0091	**0.9148**	0.0091
contraceptive	0.4284	0.0051	0.4270	0.0022	0.4270	0.0022	0.4223	0.0139	**0.4297**	0.0108	0.4270	0.0022	0.4230	0.0128	0.4270	0.0022
crx	0.5551	0.0093	0.5594	0.0096	0.5580	0.0097	0.5710	0.0305	0.5696	0.0425	0.5812	0.0563	**0.6130**	0.0835	0.6058	0.0624
dermatology	**0.9154**	0.0868	0.6469	0.0671	0.6010	0.1415	0.7935	0.1235	0.7158	0.0883	0.8143	0.0995	0.3817	0.1596	0.4537	0.1823
ecoli	0.4997	0.0715	0.5565	0.0884	0.5271	0.1002	0.4256	0.0114	**0.5824**	0.1182	0.4789	0.0572	0.4256	0.0114	0.4344	0.0333
flare	0.6304	0.0587	0.6304	0.0587	0.6510	0.0437	0.6481	0.0533	**0.6754**	0.0361	0.6416	0.0517	0.6201	0.0527	0.6726	0.0409
glass	0.3986	0.0626	0.5254	0.1355	0.5053	0.0730	0.4310	0.0620	**0.5530**	0.1608	0.4918	0.1028	0.3860	0.0612	0.4452	0.1081
haberman	0.7186	0.0363	0.7353	0.0094	0.7353	0.0094	0.7353	0.0094	0.7353	0.0094	0.7252	0.0360	**0.7482**	0.0419	0.7353	0.0094
hayes	**0.4859**	0.1675	**0.4859**	0.1675	0.2959	0.0820	0.3596	0.1311	0.2936	0.0777	0.4041	0.1531	0.4035	0.1693	0.4557	0.1558
heart	0.5556	0.0000	0.5556	0.0000	0.5556	0.0000	0.5926	0.0811	0.5556	0.0000	0.5889	0.0584	**0.6074**	0.0579	0.5556	0.0000
hepatitis	0.7938	0.0225	0.7875	0.0250	0.7817	0.0778	0.8125	0.0451	0.8192	0.0265	0.7812	0.0783	0.7938	0.0225	**0.8329**	0.0700
iris	0.4333	0.1528	0.4000	0.1333	**0.4933**	0.1611	0.4000	0.1333	0.4600	0.1562	0.3333	0.0000	0.3667	0.1000	0.4000	0.1333
mammographic	0.7182	0.1359	0.5369	0.0049	0.7255	0.1344	**0.7711**	0.0616	0.7234	0.1331	0.6192	0.0541	0.6753	0.0607	0.5724	0.1079
movement	0.0639	0.0127	0.0806	0.0231	**0.1222**	0.0530	0.0833	0.0569	0.1000	0.0451	**0.1222**	0.0889	0.0861	0.0382	0.0750	0.0279
newthyroid	0.7251	0.0644	0.7342	0.0702	0.7481	0.0817	**0.8377**	0.0964	0.7807	0.0933	0.7805	0.0936	0.7305	0.0565	0.7805	0.0792
pageblocks	**0.9119**	0.0155	0.9004	0.0051	0.9066	0.0104	0.9011	0.0098	0.9041	0.0121	0.8993	0.0023	0.8980	0.0007	0.9059	0.0137
penbased	**0.1310**	0.0247	0.1265	0.0148	0.1027	0.0019	0.1088	0.0187	0.1095	0.0071	0.1027	0.0019	0.1049	0.0027	0.1244	0.0435
phoneme	**0.7119**	0.0158	0.7065	0.0008	0.7065	0.0008	0.7065	0.0008	0.7065	0.0008	0.7091	0.0048	0.7065	0.0008	0.7084	0.0049
pima	0.6511	0.0051	0.6614	0.0182	**0.6875**	0.0447	0.6511	0.0051	0.6511	0.0051	0.6537	0.0126	0.6511	0.0051	0.6537	0.0088
saheart	0.6537	0.0030	**0.6623**	0.0195	0.6537	0.0030	0.6537	0.0030	0.6537	0.0030	0.6558	0.0067	0.6537	0.0030	0.6537	0.0030
satimage	0.2508	0.0054	0.4063	0.1059	0.3375	0.1491	0.3831	0.0947	0.2382	0.0031	0.2833	0.0958	0.2513	0.0207	**0.4128**	0.0909
segment	0.5377	0.1268	0.5515	0.0693	0.5351	0.1308	**0.5948**	0.0755	0.5074	0.0674	0.5048	0.0580	0.4913	0.0208	0.4593	0.1255
seismic	**0.9342**	0.0001	**0.9342**	0.0001	**0.9342**	0.0001	**0.9342**	0.0001	**0.9342**	0.0001	0.9230	0.0119	**0.9342**	0.0001	**0.9342**	0.0001
sonar	**0.6719**	0.1459	0.5624	0.0988	0.5338	0.0162	0.6014	0.0842	0.5338	0.0162	0.5576	0.0699	0.4910	0.0671	0.5529	0.0561
spambase	**0.7907**	0.0510	**0.7907**	0.0366	0.7786	0.0418	0.7114	0.0585	0.7494	0.0667	0.7318	0.0738	0.6167	0.0179	0.7089	0.0811
specfheart	**0.7942**	0.0166	**0.7942**	0.0166	**0.7942**	0.0166	**0.7942**	0.0166	**0.7942**	0.0166	**0.7942**	0.0166	0.7868	0.0318	**0.7942**	0.0166
tae	0.3842	0.0652	**0.4513**	0.1644	0.3842	0.1222	0.3642	0.0738	0.3442	0.0224	0.3913	0.1066	0.3508	0.0585	0.3983	0.1212
thoracic	0.8511	0.0000	0.8489	0.0064	0.8426	0.0255	0.8511	0.0000	**0.8553**	0.0085	0.8319	0.0260	0.8511	0.0095	**0.8553**	0.0085
titanic	0.6770	0.0009	0.6770	0.0009	0.6770	0.0009	0.6770	0.0009	**0.7678**	0.0289	0.6770	0.0009	0.6770	0.0009	0.6770	0.0009
transfusion	**0.7621**	0.0041	**0.7621**	0.0041	**0.7621**	0.0041	**0.7621**	0.0041	0.7514	0.0329	**0.7621**	0.0041	**0.7621**	0.0041	**0.7621**	0.0041
vehicle	0.2659	0.0201	0.2624	0.0183	0.2897	0.0380	0.2671	0.0213	0.2718	0.0289	**0.3676**	0.0917	0.2932	0.0166	0.3236	0.0713
vowel	0.0909	0.0000	0.0939	0.0091	0.0909	0.0000	0.1253	0.0268	0.0909	0.0000	0.0909	0.0000	0.0909	0.0000	**0.1374**	0.0399
wine	0.4680	0.1439	0.5324	0.1848	**0.6725**	0.1221	0.4435	0.0818	0.4503	0.0852	0.5879	0.1964	0.4493	0.0645	0.6621	0.1363
wisconsin	0.7327	0.0991	0.7512	0.0952	0.7497	0.1052	0.6552	0.0093	0.6552	0.0093	0.6552	0.0093	**0.7567**	0.1281	0.7169	0.0989
yeast	0.3127	0.0048	0.3133	0.0046	0.3255	0.0150	0.3167	0.0134	**0.3322**	0.0177	0.3174	0.0126	0.3167	0.0047	0.3167	0.0102
MEAN	***0.5855***	*0.0509*	*0.5803*	*0.0499*	*0.5823*	*0.0553*	*0.5726*	*0.0475*	*0.5792*	*0.0454*	*0.5756*	*0.0503*	*0.5475*	*0.0366*	*0.5819*	*0.0571*

obtained by C4.5 and Naive Bayes using all discretizers is presented in Tables III and IV, respectively. The best case in each data set is highlighted in bold. Likewise, Table V shows the average cut points yielded by each discretizer. Observing the results, the following analysis can be stated.

a) According to the classification accuracy, EMD is the best alternative when using C4.5 as classifier on average (0.7852). Our proposal outperforms 14/45, being the most promising algorithm out of all those used. Regarding to Naive Bayes, EMD again represents the best option with the most promising mean (0.7747), outperforming on 14/45 datasets.
b) The lowest number of cut points on average (26/45) is held by our method. EMD yields simpler discretization schemes in almost half of the cases.
c) A remarkable case in point is observed in the simple iris data set. EMD only requires three cut points to offer the best accuracy with C4.5 and Naive Bayes. There are other datasets where our method also outperforms the others (i.e., page-blocks and segment), in both number of points and classification accuracy (for C4.5 and Naive Bayes).
d) For larger datasets (with many attributes and/or many examples), our algorithm generally obtains simpler solutions with accurate schemes due to the reduction process included.

Statistical analysis will be carried out by means of nonparametric statistical tests, such as: Wilcoxon signed-ranks test and the Friedman procedure [23]–[25].

The nonparametric Wilcoxon signed-ranks test is used for conducting pairwise comparison between our proposal and the rest of the techniques. Table VI collects the results offered by the Wilcoxon test considering a level of significance equal to $\alpha = 0.05$. This table is divided into three parts, each one associated with columns: in the first and second parts, the measure of accuracy classification in the test is used for C4.5 and Naive Bayes, respectively. In the third part, we accomplish the Wilcoxon test by using as a performance measure the number of cut points produced by the discretizers. The table indicates, for each method in the rows, the number of discretizers outperformed by using the Wilcoxon test under the column represented by the "+" symbol. The column with the "±" symbol indicates the number of wins and ties obtained by the method in the row. The maximum value for each column is highlighted by a shaded cell.

Tables VII and VIII present the statistical analysis conducted by the Friedman procedure for C4.5 and Naive Bayes accuracy, and for the number of cut points, respectively. This test generates a ranking according to the effectiveness associated with

TABLE X
AVERAGE ACCURACY OBTAINED FOR PUBLIC

	Ameva		CAIM		ChiMerge		FUSINTER		MDLP		Modified Chi2		PKID		EMD	
	Avg	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*
abalone	0.2199	0.0250	0.2549	0.0163	0.2540	0.0180	0.2568	0.0202	0.2585	0.0191	0.2554	0.0198	0.2252	0.0084	**0.2595**	0.0149
appendicitis	0.8427	0.1275	0.8427	0.1359	0.8327	0.1109	0.8145	0.1303	0.8609	0.1059	0.8336	0.1270	0.7927	0.0557	**0.8700**	0.0715
australian	0.8377	0.0436	0.8377	0.0385	0.8377	0.0414	**0.8551**	0.0306	0.8478	0.0322	**0.8551**	0.0306	0.8522	0.0319	0.8478	0.0306
autos	**0.6398**	0.0986	0.5956	0.0749	0.6275	0.1040	0.6045	0.1032	0.6353	0.1278	0.6347	0.0919	0.5462	0.0548	0.5759	0.0800
balance	0.7104	0.0574	0.7073	0.0540	0.7619	0.0550	0.7456	0.0284	0.6959	0.0544	0.7489	0.0324	0.7265	0.0300	**0.7697**	0.0381
banana	0.7249	0.0206	0.6387	0.0158	0.6253	0.0189	0.8674	0.0183	0.7492	0.0213	0.5517	0.0010	0.5517	0.0010	**0.8704**	0.0145
bands	0.6495	0.0667	**0.6901**	0.0618	0.6790	0.0512	0.6864	0.0576	0.5529	0.0433	0.6568	0.0502	0.6753	0.0513	0.6734	0.0488
banknote	0.9082	0.0213	0.8899	0.0192	0.8833	0.0250	0.9613	0.0231	0.9424	0.0186	0.9476	0.0199	0.6472	0.0292	**0.9694**	0.0130
bupa	0.6382	0.0879	0.5980	0.0953	**0.6395**	0.0813	0.5789	0.0333	0.5715	0.0740	0.5789	0.0333	0.5789	0.0333	0.6382	0.0730
cleveland	0.5375	0.0616	0.5444	0.0687	0.5410	0.0773	0.5280	0.0526	**0.5609**	0.0612	0.5409	0.0623	0.5444	0.0472	0.5577	0.0553
climate	0.9111	0.0246	0.9222	0.0284	0.9148	0.0091	0.9148	0.0189	**0.9333**	0.0264	0.9037	0.0319	0.9148	0.0091	0.9222	0.0395
contraceptive	0.4820	0.0428	0.5384	0.0347	**0.5425**	0.0379	0.4942	0.0430	0.4915	0.0350	0.4650	0.0329	0.4691	0.0331	0.5092	0.0535
crx	0.8406	0.0534	0.8348	0.0526	0.8435	0.0583	0.8449	0.0568	0.8377	0.0467	0.8551	0.0478	0.8551	0.0478	**0.8551**	0.0454
dermatology	0.9563	0.0428	**0.9619**	0.0446	0.9536	0.0495	0.9592	0.0515	0.9536	0.0495	0.9592	0.0515	0.9563	0.0428	0.9509	0.0316
ecoli	0.7565	0.0613	**0.8063**	0.0585	0.7534	0.0417	0.7949	0.0390	0.8038	0.0330	0.7059	0.0663	0.6966	0.0646	0.7623	0.0646
flare	**0.6754**	0.0380	**0.6754**	0.0380	**0.6754**	0.0380	**0.6754**	0.0380	**0.6754**	0.0380	**0.6754**	0.0380	**0.6754**	0.0380	**0.6754**	0.0361
glass	0.4842	0.0731	0.6782	0.1566	0.6335	0.1362	0.6471	0.0711	0.6744	0.1024	0.6092	0.0936	0.4701	0.0798	**0.7184**	0.1220
haberman	0.7284	0.0408	**0.7512**	0.0601	0.7186	0.0349	0.7482	0.0542	0.7285	0.0370	0.7353	0.0100	0.7353	0.0100	0.7447	0.0577
hayes	**0.8300**	0.0973	**0.8300**	0.0973	**0.8300**	0.0973	0.7113	0.1242	0.5202	0.0812	0.7415	0.1332	0.6604	0.0961	0.7476	0.1242
heart	0.7556	0.0823	0.7556	0.0823	0.7556	0.0765	0.7704	0.0904	0.7593	0.0824	0.7667	0.0874	0.7593	0.1051	**0.8074**	0.0857
hepatitis	0.7608	0.1330	0.7554	0.0767	0.7808	0.0309	0.7683	0.0581	0.7742	0.0556	0.7808	0.0309	0.7804	0.0539	**0.8008**	0.0569
iris	0.9333	0.0314	0.9333	0.0314	0.9333	0.0314	0.9400	0.0378	0.9333	0.0314	0.9467	0.0422	0.8333	0.0956	**0.9533**	0.0427
mammographic	0.8221	0.0595	**0.8315**	0.0414	0.8169	0.0572	0.8315	0.0426	0.8294	0.0441	0.8242	0.0551	0.8315	0.0434	0.8294	0.0514
movement	0.3556	0.0366	0.3500	0.0743	0.3722	0.0766	**0.5750**	0.0525	0.5500	0.0611	0.5528	0.0443	0.1917	0.0515	0.5417	0.0672
newthyroid	0.9255	0.0399	0.9255	0.0453	0.9398	0.0541	0.9255	0.0501	0.9299	0.0465	0.9442	0.0486	0.9028	0.0555	**0.9494**	0.0433
pageblocks	**0.9669**	0.0035	0.9627	0.0041	0.9642	0.0064	0.9611	0.0050	0.9638	0.0074	0.9476	0.0124	0.9472	0.0073	0.9653	0.0076
penbased	0.9282	0.0116	0.9171	0.0097	0.9131	0.0110	0.9218	0.0077	0.9124	0.0097	0.9142	0.0117	0.7865	0.0162	**0.9519**	0.0072
phoneme	0.7907	0.0178	0.7877	0.0155	0.7794	0.0221	0.7889	0.0241	0.7950	0.0216	0.7065	0.0008	0.7065	0.0008	**0.8151**	0.0187
pima	0.7344	0.0443	0.7346	0.0544	0.7266	0.0337	0.7003	0.0530	0.7305	0.0394	0.7019	0.0447	0.6550	0.0095	**0.7499**	0.0377
saheart	0.6904	0.0535	**0.7078**	0.0324	0.6949	0.0369	0.6797	0.0374	0.6710	0.0426	0.6816	0.0625	0.6386	0.0699	0.6818	0.0469
satimage	0.2446	0.0034	**0.8493**	0.0160	0.8421	0.0162	0.8448	0.0106	0.8252	0.0127	0.8413	0.0103	0.7277	0.0307	0.8449	0.0126
segment	0.9455	0.0166	0.9329	0.0149	0.9459	0.0146	0.9398	0.0263	0.9390	0.0105	0.8502	0.0485	0.7519	0.0314	**0.9468**	0.0131
seismic	**0.9342**	0.0001	**0.9342**	0.0001	**0.9342**	0.0001	**0.9342**	0.0001	**0.9342**	0.0001	**0.9342**	0.0001	**0.9342**	0.0001	0.9334	0.0024
sonar	0.7055	0.0943	0.7243	0.1330	**0.7643**	0.0951	0.7162	0.1375	0.7398	0.0904	0.7400	0.0997	0.5338	0.0171	0.7014	0.0669
spambase	0.9226	0.0139	0.9232	0.0147	**0.9278**	0.0064	0.9102	0.0143	0.9147	0.0150	0.8601	0.0215	0.9030	0.0174	0.9147	0.0165
specfheart	0.7793	0.0389	0.7793	0.0389	0.7942	0.0247	0.7942	0.0175	0.7792	0.0257	0.7979	0.0175	0.7942	0.0175	**0.8017**	0.0276
tae	0.4117	0.1383	**0.5838**	0.1074	0.4379	0.1324	0.3379	0.0228	0.3442	0.0236	0.3313	0.0321	0.3379	0.0228	0.4504	0.1147
thoracic	**0.8511**	0.0000	**0.8511**	0.0000	**0.8511**	0.0000	**0.8511**	0.0000	**0.8511**	0.0000	**0.8511**	0.0000	**0.8511**	0.0000	**0.8511**	0.0000
titanic	0.7751	0.0299	0.7783	0.0297	0.7751	0.0301	0.7751	0.0301	0.7678	0.0305	0.7751	0.0301	0.7906	0.0233	**0.7906**	0.0221
transfusion	**0.7794**	0.0376	0.7700	0.0246	0.7487	0.0324	0.7554	0.0133	0.7514	0.0329	0.7621	0.0041	0.7621	0.0041	0.7554	0.0355
vehicle	**0.6833**	0.0468	0.6820	0.0603	0.6668	0.0550	0.6726	0.0328	0.6679	0.0470	0.6572	0.0329	0.5899	0.0675	0.6832	0.0237
vowel	0.6010	0.0765	0.6121	0.0460	0.5818	0.0713	**0.6717**	0.0413	0.6182	0.0523	0.6576	0.0568	0.3667	0.0393	0.6515	0.0345
wine	0.9265	0.0757	0.8925	0.0740	0.9271	0.0525	0.9438	0.0586	**0.9608**	0.0377	0.9490	0.0561	0.8082	0.1425	0.9157	0.0682
wisconsin	0.9428	0.0213	0.9385	0.0234	0.9471	0.0233	**0.9528**	0.0252	0.9471	0.0337	0.9427	0.0405	0.9399	0.0315	0.9456	0.0155
yeast	0.5176	0.0509	0.5358	0.0381	0.5249	0.0432	0.5567	0.0313	**0.5661**	0.0358	0.4266	0.0524	0.4455	0.0470	0.5580	0.0286
MEAN	*0.7346*	*0.0498*	*0.7566*	*0.0498*	*0.7532*	*0.0471*	*0.7602*	*0.0425*	*0.7500*	*0.0422*	*0.7422*	*0.0426*	*0.6921*	*0.0392*	***0.7713***	*0.0436*

TABLE XI
WILCOXON TEST RESULTS FOR ACCURACY
(PART AND PUBLIC)

Algorithms	PART		PUBLIC	
	+	±	+	±
Ameva	1	7	1	6
CAIM	1	7	1	6
ChiMerge	1	7	1	6
FUSINTER	1	6	1	6
MDLP	1	7	1	6
Modified Chi2	1	7	1	6
PKID	0	0	0	0
EMD	2	7	7	7

TABLE XII
AVERAGE RANKINGS OF THE ALGORITHMS (FRIEDMAN PROCEDURE + ADJUSTED p-VALUE WITH HOLM'S TEST) FOR ACCURACY
(PART AND PUBLIC)

PART			PUBLIC		
Algorithms	Ranking	p_{Holm}	Algorithms	Ranking	p_{Holm}
EMD	3.84	—	EMD	2.97	—
CAIM	4.16	1.0937	FUSINTER	4.03	0.0505
ChiMerge	4.16	1.0937	CAIM	4.12	0.0505
MDLP	4.33	1.0313	ChiMerge	4.54	0.0090
Ameva	4.52	0.7779	MDLP	4.54	0.0090
FUSINTER	4.58	0.7779	Mod. Chi2	4.78	0.0023
Mod. Chi2	4.69	0.6120	Ameva	4.87	0.0014
PKID	5.72	0.0019	PKID	6.14	~ 0

each discretizer (second column), ordering the tables from the best to the worst Friedman ranking. The third column shows the adjusted p-value with the *post hoc* Holm's test. Note that EMD is established as the control algorithm because it has obtained the best position in all the rankings. By using a level of significance $\alpha = 0.1$, EMD is significantly better than the rest of the methods, considering both C4.5 accuracy and number of cut points. Although EMD obtains the best Friedman rank for Naive Bayes, the second column shows that EMD is not significantly different to the rest of discretizers.

According to the statistical analysis, we can assert the following.

a) Statistically, no method can be considered to be better than our proposal in any of the measures used.
b) The nonparametric tests confirms that EMD is better than all the other discretizers for all measures. Only for Naive Bayes, our proposal only outperforms 4/7 methods.
c) Considering the tradeoff accuracy/simplicity, we can establish EMD as the best option. It needs to make a

TABLE XIII
ECPSD RESULTS

	C4.5 Accuracy		**Naive Bayes Accuracy**		**Cut points**	
	Avg	*Std-Dev*	*Avg*	*Std-Dev*	*Avg*	*Std-Dev*
abalone	0.1996	0.0200	0.2599	0.0172	1,527.00	32.80
appendicitis	0.8327	0.1199	0.8127	0.1153	9.40	1.02
australian	0.7681	0.0531	0.7942	0.0522	34.70	4.80
autos	0.7951	0.0632	0.6006	0.1011	22.00	14.66
balance	0.5556	0.0610	0.5556	0.0610	1.00	0.00
banana	0.6902	0.0210	0.7019	0.0151	429.20	25.48
bands	0.6121	0.0731	0.6346	0.0361	25.60	1.96
banknote	0.8783	0.0407	0.9038	0.0138	170.70	54.04
bupa	0.6475	0.0787	0.6434	0.0644	18.40	1.20
cleveland	0.5119	0.0639	0.5710	0.0755	15.90	1.04
climate	0.9185	0.0170	0.9167	0.0149	113.40	43.34
contraceptive	0.5173	0.0408	0.5173	0.0291	17.10	0.70
crx	0.8565	0.0402	0.8464	0.0494	15.80	10.35
dermatology	0.9398	0.0422	0.9535	0.0348	6.90	0.70
ecoli	0.7414	0.0440	0.8065	0.0666	17.50	1.12
flare	0.6754	0.0361	0.6754	0.0361	2.00	0.00
glass	0.6666	0.0970	0.6468	0.1200	25.40	4.48
haberman	0.7252	0.0356	0.7217	0.0425	2.40	1.02
hayes	0.3234	0.1250	0.3234	0.1250	2.00	0.45
heart	0.8074	0.0544	0.8333	0.0504	12.00	0.63
hepatitis	0.8183	0.0917	0.8379	0.1079	9.40	0.66
iris	0.9467	0.0400	0.9467	0.0400	2.60	0.49
mammographic	0.8023	0.0471	0.8023	0.0471	1.50	0.67
movement	0.2444	0.0324	0.5972	0.0469	5,917.50	95.05
newthyroid	0.9398	0.0420	0.9489	0.0442	5.20	1.17
pageblocks	0.9375	0.0187	0.9271	0.0101	592.20	53.74
penbased	0.8941	0.0282	0.8663	0.0104	182.40	49.93
phoneme	0.7406	0.0204	0.7816	0.0138	1,409.00	28.80
pima	0.7436	0.0432	0.7593	0.0451	43.90	2.84
saheart	0.6905	0.0424	0.6861	0.0404	57.20	12.24
satimage	0.8491	0.0110	0.8096	0.0167	322.60	62.11
segment	0.7450	0.0281	0.8788	0.0197	2,976.00	38.11
seismic	0.9342	0.0001	0.8665	0.0159	64.30	23.58
sonar	0.7057	0.1088	0.7452	0.1051	1,001.80	44.84
spambase	0.8830	0.0148	0.8973	0.0146	2,184.90	60.44
specfheart	0.7792	0.0478	0.8019	0.0565	35.50	23.00
tae	0.5229	0.0665	0.5229	0.1605	14.20	1.17
thoracic	0.8468	0.0085	0.8213	0.0332	6.00	0.89
titanic	0.6770	0.0009	0.6770	0.0009	0.00	0.00
transfusion	0.7848	0.0383	0.7674	0.0274	3.80	0.40
vehicle	0.6702	0.0494	0.5591	0.0536	107.90	16.66
vowel	0.3364	0.0356	0.4394	0.0706	1,950.50	35.90
wine	0.8650	0.0622	0.9154	0.0680	19.20	8.00
wisconsin	0.9442	0.0323	0.9442	0.0323	2.20	0.40
yeast	0.5081	0.0371	0.5755	0.0310	31.20	0.98
MEAN	*0.7216*	*0.0461*	*0.7354*	*0.0496*	*431.3200*	*16.9301*

TABLE XIV
WILCOXON TEST RESULTS FOR ACCURACY (C4.5 AND NAIVE BAYES) AND NUMBER OF CUT POINTS (ECPSD)

Algorithms	C4.5		Naive Bayes		# Cut Points	
	+	±	+	±	+	±
ECPSD	0	0	0	0	0	0
EMD	1	1	1	1	1	1

lower number of cut points than the discretizers which are similar in accuracy, specially for the largest datasets. In fact, in many cases it is capable of outperforming in accuracy the discretizers with similar performance in simplicity.

d) EMD obtains the best results considering both classifiers. No discretizer has performed well on both classifiers simultaneously up to now.

2) Other Classifiers Comparison: To compare the performance of our method using other classifiers not included in its fitness function, we compare the accuracy obtained in test data by the classifiers PART and PUBLIC using all previous discretizers. This information is presented in Tables IX and X,

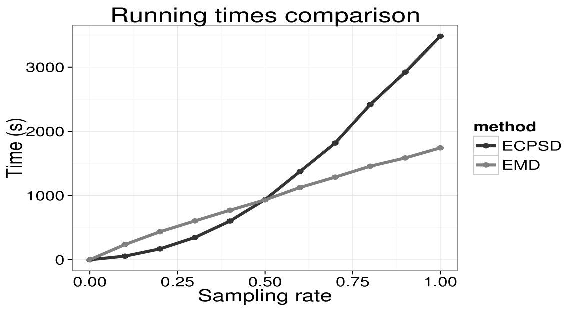


Fig. 4. Running times comparison (pen-based).

respectively. Observing the results, the following analysis can be stated.

a) According to the classification accuracy, EMD is near to the best alternative on average (Ameva, 0.5855) when using PART as classifier (0.5819). Our proposal outperforms 11/45, being one of the most promising algorithm out of all those used. Better results are obtained using PUBLIC, where EMD represents the best alternative with the most promising mean (0.7713), outperforming on 19/45 datasets.

b) EMD demonstrates to perform even better than the previous classifiers with PUBLIC and in similar way to the others discretizers with PART. It is again one of the most of promising algorithms with classifiers not included in its fitness function.

Statistical analysis will be again carried out by means of nonparametric statistical tests, as we presented before in Section IV-B1. Table XI collects the results offered by the Wilcoxon test considering a level of significance equal to $\alpha = 0.05$. This table is divided into two columns: both of them measure the classification accuracy on test for PART and PUBLIC, respectively.

Table XII presents the statistical analysis conducted by the Friedman procedure for PART and PUBLIC accuracy. EMD is established as the control algorithm as it has obtained the best position in all the rankings. By using a level of significance $\alpha = 0.1$, EMD is significantly better than the rest of the methods, considering PUBLIC accuracy. Although EMD obtains the best Friedman rank for PART, the second column shows that EMD is not significantly different to the rest of discretizers.

According to the statistical analysis, we can assert the following.

a) Statistically, no method can be considered to be better than our proposal in any of the measures used.

b) The nonparametric tests confirms that EMD is better than PUBLIC in test accuracy. Only for PART, our proposal only outperforms 2/7 methods. However, it can be established again as the best alternative.

c) EMD obtains the best results considering both classifiers, as in the previous experiment.

3) ECPSD Comparison: ECPSD results are presented in Table XIII in terms of C4.5 and Naive Bayes test accuracy and number of cut points. We compare these results with those yielded by our approach (Tables III–V). Additionally,

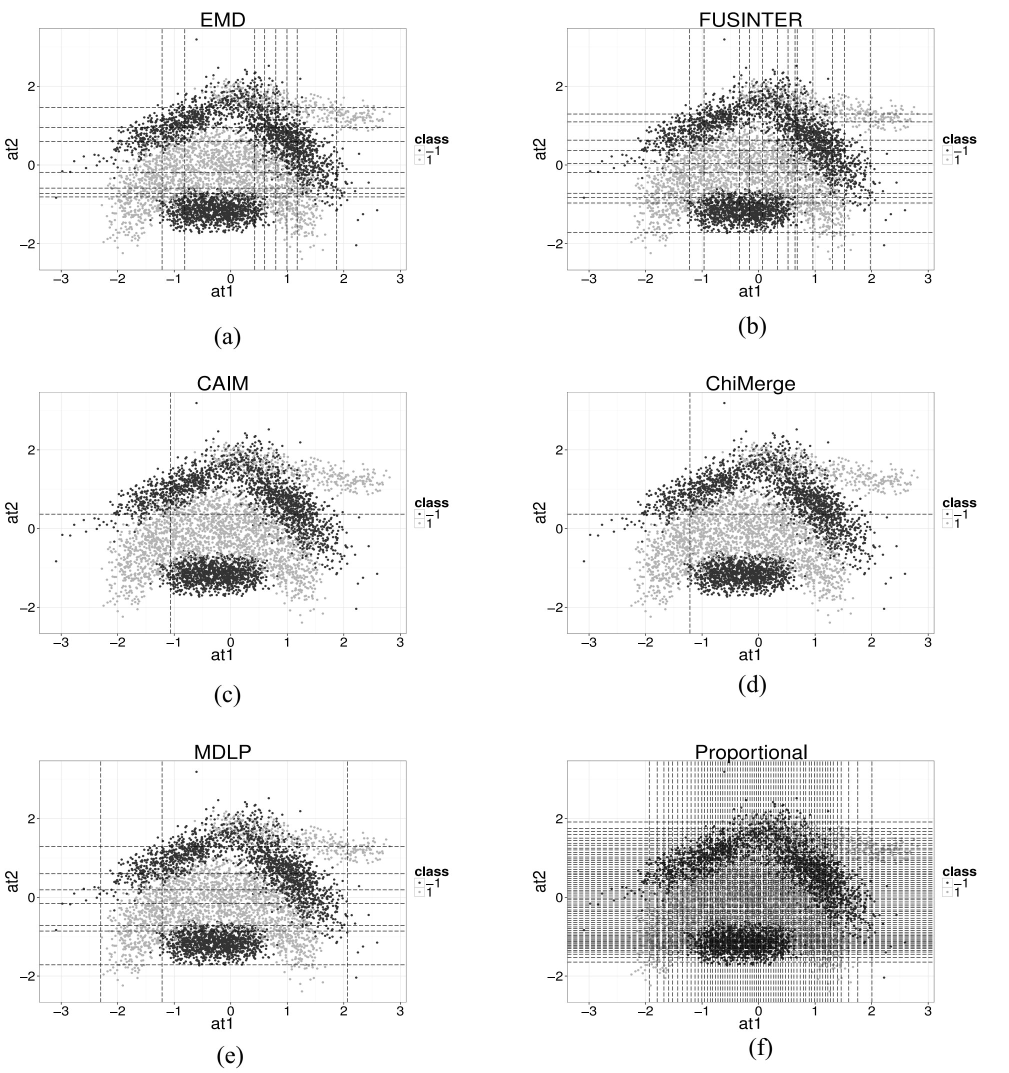


Fig. 5. Case study: banana. (a) EMD: % test acc. C45 (0.8730), % test acc. NB (0.7357), # intervals (15.00). (b) FUSINTER: % test acc. C45 (0.8791), % test acc. NB (0.7162), # intervals (25.30). (c) CAIM: % test acc. C45 (0.6387), % test acc. NB (0.6049), # intervals (3.00). (d) ChiMerge: % test acc. C45 (0.6253), % test acc. NB (0.5836), # intervals (3.00). (e) MDLP: % test acc. C45 (0.7485), % test acc. NB (0.7247), # intervals (11.10). (f) PKID: % test acc. C45 (0.7043), % test acc. NB (0.7147), # intervals (137.00).

Wilcoxon test will be used to compare both algorithms as presented in Section IV-B1. Table XIV collects the results offered by this test considering a level of significance equal to $\alpha = 0.05$. This table is divided into three parts, each one associated with columns: in the first and second parts, the measure of accuracy classification in the test is used for C4.5 and Naive Bayes, respectively. In the third part, we accomplish the Wilcoxon test by using as a performance measure the number of cut points produced by the discretizers.

According to this information, we can assert the following.

a) EMD outperforms 32/45 for C4.5 test accuracy, 34/45 for Naive Bayes test accuracy, and 34/45 for the number of cut points. Besides, EMD always performs better than ECPSD for all measures considered, on average.
b) The nonparametric tests confirms that EMD is better than ECPSD in all measures used in these experiments.

Regarding to the time complexity of both algorithms, Fig. 4 shows the running times derived from applying these algorithms on different versions of penbased dataset (described in Table I). This dataset is formed by a pen-based database with more than 11 k isolated handwritten characters. We have chosen this for being the set with the largest number of instances in our experiments.

In this figure, we apply different sampling percentages over this dataset to show the differences between both methods when they try to scale-up large datasets. We can observe as in the smallest cases (<50%) ECPSD is faster than EMD due to the fastness of its inconsistency-based fitness function. However, as the sample size grows the chromosome reduction mechanism starts to reduce the complexity of the solutions to evaluate, and then the convergence becomes faster. It demonstrates EMDs ability to scale-up and the effectiveness of its chromosome reduction mechanism.

Even though our algorithm presents a higher time complexity than other discretizers, it obtains better results in accuracy and produces simpler solutions than its competitors. Time is not really determinant in this case since discretization is a offline DM task that is only performed once before the classification task, which indeed will perform faster due to the simpler discretization schemes derived.

C. Case Study

To illustrate the discretization schemes generated by each different discretizer, we propose a case study depicted in Fig. 5. This figure represents the different discretization intervals generated on banana dataset by some of the algorithms presented above. Banana is a very noisy artificially created dataset with two dimensions (described in Table I). This was created using a mixture of overlapping Gaussians with a banana-shaped distribution. The data is uniformly distributed along the clusters and superimposed with a normal distribution with standard deviation in all directions.

In this figure, we can see how EMD and FUSINTER clearly show good performance in the banana dataset. Both algorithms generate a correct number of cut points that properly separate examples from different classes. There is no clear difference between both of them on accuracy, however, EMD needs fewer cut points; therefore obtaining far more representative solutions than those yielded by FUSINTER. This representativeness can be measured counting the number of grids dominated by a class which are completely surrounded by other grids also dominated by the same class. Being approximately 3, for EMD; and approximately 15, for FUSINTER.

The rest of the algorithms do not offer better results. In fact, most of them do not perform very well on banana. CAIM and ChiMerge produce a small number of cut points (only two) whereas PKID exceed a sensible number of cut points.

V. Conclusion

In this paper, we have presented a new evolutionary-based discretization algorithm called EMD, which selects the most adequate combination of boundary cut points to create discrete intervals. For this purpose, EMD uses a wrapper fitness function based on the classification error provided by two important classifiers, and the number of cut points produced. The proposed algorithm follows a multivariate approach, being able to take advantage of the existing interactions and dependencies among the set of input attributes and the class output to improve the discretization process. It also includes a chromosome reduction mechanism to tackle larger problems and, in general, to speed-up its performance on all kinds of datasets. The large experimental study performed allows us to show that EMD is a suitable method for discretization in small and large problems. It requires a lower number of cut points than the other discretizers, thus producing much simpler discretization solutions. Additionally, EMD outperforms the state-of-the-art discretizers on classification accuracy. It can be considered as the best choice for classifiers, such as C4.5, Naive Bayes, PART, or PUBLIC.

References

[1] S. García, J. Luengo, and F. Herrera, *Data Preprocessing in Data Mining*. Cham, Switzerland: Springer, 2015.

[2] S. Li and D. Wei, "Extremely high-dimensional feature selection via feature generating samplings," *IEEE Trans. Cybern.*, vol. 44, no. 6, pp. 737–747, Jun. 2014.

[3] N. García-Pedrajas, J. Pérez-Rodríguez, and A. de Haro-García, "OligoIs: Scalable instance selection for class-imbalanced data sets," *IEEE Trans. Cybern.*, vol. 43, no. 1, pp. 332–346, Feb. 2013.

[4] Y. Yang, G. I. Webb, and X. Wu, "Discretization methods," in *Data Mining and Knowledge Discovery Handbook*. New York, NY, USA: Springer, 2005, pp. 113–130.

[5] S. García, J. Luengo, J. A. Sáez, V. López, and F. Herrera, "A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 4, pp. 734–750, Apr. 2013.

[6] H. Liu, F. Hussain, C. L. Tan, and M. Dash, "Discretization: An enabling technique," *Data Min. Knowl. Disc.*, vol. 6, no. 4, pp. 393–423, 2002.

[7] H.-W. Hu, Y.-L. Chen, and K. Tang, "A novel decision-tree method for structured continuous-label classification," *IEEE Trans. Cybern.*, vol. 43, no. 6, pp. 1734–1746, Dec. 2013.

[8] P. Clark and T. Niblett, "The CN2 induction algorithm," *Mach. Learn.*, vol. 3, no. 4, pp. 261–283, 1989.

[9] G. Cervone, L. Panait, and R. S. Michalski, "The development of the AQ20 learning system and initial experiments," in *Intelligent Information Systems* (Advances in Intelligent and Soft Computing), M. A. Klopotek, M. Michalewicz, and S. T. Wierzchon, Eds. Berlin, Germany: Physica, 2001, pp. 13–29.

[10] Y. Yang and G. I. Webb, "Discretization for naive-Bayes learning: Managing discretization bias and variance," *Mach. Learn.*, vol. 74, no. 1, pp. 39–74, 2009.

[11] X. Wang, Y. He, and D. D. Wang, "Non-naive Bayesian classifiers for classification problems with continuous attributes," *IEEE Trans. Cybern.*, vol. 44, no. 1, pp. 21–39, Jan. 2014.

[12] Q. Wu *et al.*, "Improvement of decision accuracy using discretization of continuous attributes," in *Proc. 3rd Int. Conf. Fuzzy Syst. Knowl. Disc. (FSKD)*, Xi'an, China, 2006, pp. 674–683.

[13] H.-W. Hu, Y.-L. Chen, and K. Tang, "A dynamic discretization approach for constructing decision trees with a continuous label," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 11, pp. 1505–1514, Nov. 2009.

[14] A. A. Freitas, *Data Mining and Knowledge Discovery With Evolutionary Algorithms*. New York, NY, USA: Springer, 2002.

[15] Z. He, S. Tian, and H. Huang, "EMVD-BDC: An evolutionary multivariate discretization approach for association rules," *J. Comput. Inf. Syst.*, vol. 2, no. 4, pp. 1343–1350, 2006.

[16] C.-W. Chen, Z.-G. Li, S.-Y. Qiao, and S.-P. Wen, "Study on discretization in rough set based on genetic algorithm," in *Proc. 2nd Int. Conf. Mach. Learn. Cybern. (ICMLC)*, Xi'an, China, 2003, pp. 1430–1434.

[17] J.-H. Dai, "A genetic algorithm for discretization of decision systems," in *Proc. 3rd Int. Conf. Mach. Learn. Cybern. (ICMLC)*, vol. 3. Shanghai, China, 2004, pp. 1319–1323.

[18] J. L. Flores, I. Inza, and P. Larraaga, "Wrapper discretization by means of estimation of distribution algorithms," *Intell. Data Anal.*, vol. 11, no. 5, pp. 525–545, 2007.

[19] S. García, V. López, J. Luengo, C. J. Carmona, and F. Herrera, "A preliminary study on selecting the optimal cut points in discretization by evolutionary algorithms," in *Proc. Int. Conf. Pattern Recognit. Appl. Methods (ICPRAM)*, Algarve, Portugal, 2012, pp. 211–216.

[20] A. Arauzo-Azofra, J. M. Bentez, and J. L. Castro, "Consistency measures for feature selection," *J. Intell. Inf. Syst.*, vol. 30, no. 3, pp. 273–292, 2008.

[21] M. Dash and H. Liu, "Consistency-based search in feature selection," *Artif. Intell.*, vol. 151, nos. 1–2, pp. 155–176, 2003.

[22] X. Wu and V. Kumar, Eds., *The Top Ten Algorithms in Data Mining* (Data Mining and Knowledge Discovery). Boca Raton, FL, USA: CRC Press, 2009.

[23] S. García, A. Fernández, J. Luengo, and F. Herrera, "A study of statistical techniques and performance measures for genetics-based machine learning: Accuracy and interpretability," *Soft Comput.*, vol. 13, no. 10, pp. 959–977, 2009.

[24] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Inf. Sci.*, vol. 180, no. 10, pp. 2044–2064, 2010.

[25] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 3–18, 2011.

[26] T. Elomaa and J. Rousu, "General and efficient multisplitting of numerical attributes," *Mach. Learn.*, vol. 36, no. 3, pp. 201–244, 1999.

[27] L. A. Kurgan and K. J. Cios, "CAIM discretization algorithm," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 2, pp. 145–153, Feb. 2004.
[28] R. Kerber, "ChiMerge: Discretization of numeric attributes," in *Proc. Nat. Conf. Artif. Intell. Amer. Assoc. Artif. Intell. (AAAI)*, San Jose, CA, USA, 1992, pp. 123–128.
[29] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley, 1989.
[30] D. Simon, "Biogeography-based optimization," *IEEE Trans. Evol. Comput.*, vol. 12, no. 6, pp. 702–713, Dec. 2008.
[31] W. Nuij, V. Milea, F. Hogenboom, F. Frasincar, and U. Kaymak, "An automated framework for incorporating news into stock trading strategies," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 823–835, Apr. 2014.
[32] S.-M. Chen, Y.-C. Chang, and J.-S. Pan, "Fuzzy rules interpolation for sparse fuzzy rule-based systems based on interval type-2 Gaussian fuzzy sets and genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 21, no. 3, pp. 412–425, Jun. 2013.
[33] A. Ning, X. Zhang, and W. Duan, "DHMM speech recognition algorithm based on immune particle swarm vector quantization," in *Proc. 3rd Int. Conf. Artif. Intell. Comput. Intell. (AICI)*, vol. 3. Taiyuan, China, 2011, pp. 420–427.
[34] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Berlin, Germany: Springer, 2003.
[35] L. J. Eshelman, "The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination," in *Foundations of Genetic Algorithms*. San Mateo, CA, USA: Morgan Kaufmann, 1990, pp. 265–283.
[36] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA, USA: Morgan Kaufmann, 1993.
[37] K. J. Cios, W. Pedrycz, R. W. Swiniarski, and L. A. Kurgan, *Data Mining: A Knowledge Discovery Approach*. New York, NY, USA: Springer, 2007.
[38] W. Sheng, X. Liu, and M. C. Fairhurst, "A niching memetic algorithm for simultaneous clustering and feature selection," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 7, pp. 868–879, Jul. 2008.
[39] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization," in *Proc. 15th Int. Conf. Mach. Learn.*, Madison, WI, USA, 1998, pp. 144–151.
[40] R. Rastogi and K. Shim, "PUBLIC: A decision tree classifier that integrates building and pruning," in *Proc. 24th Int. Conf. Very Large Data Bases*, New York, NY, USA, 1998, pp. 404–415.
[41] L. Gonzalez-Abril, F. J. Cuberos, F. Velasco, and J. A. Ortega, "Ameva: An autonomous discretization algorithm," *Expert Syst. Appl.*, vol. 36, no. 3, pp. 5327–5332, 2009.
[42] D. A. Zighed, S. Rabaséda, and R. Rakotomalala, "FUSINTER: A method for discretization of continuous attributes," *Int. J. Uncertain. Fuzz. Knowl.-Based Syst.*, vol. 6, no. 3, pp. 307–326, 1998.
[43] U. M. Fayyad and K. B. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *Proc. 13th Int. Joint Conf. Artif. Intell. (IJCAI)*, Chambéry, France, 1993, pp. 1022–1029.
[44] F. E. H. Tay and L. Shen, "A modified Chi2 algorithm for discretization," *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 3, pp. 666–670, May/Jun. 2002.
[45] J. Alcalá-Fdez *et al.*, "KEEL: A software tool to assess evolutionary algorithms for data mining problems," *Soft Comput.*, vol. 13, no. 3, pp. 307–318, 2009.
[46] K. Bache and M. Lichman. (2013). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml

Sergio Ramírez-Gallego received the M.Sc. degree in computer science from the University of Jaén, Jaén, Spain, in 2012. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain.

His current research interests include data mining, data preprocessing, cloud computing, and big data.

Salvador García received the M.Sc. and Ph.D. degrees in computer science from the University of Granada, Granada, Spain, in 2004 and 2008, respectively.

He is currently an Assistant Professor with the Department of Computer Science, University of Jaén, Jaén, Spain. His current research interests include data mining, data reduction, data complexity, imbalanced learning, semi-supervised learning, statistical inference, and evolutionary algorithms. He has published over 40 papers in international journals, co-edited two special issues of international journals on different data mining topics, and has co-authored the book entitled *Data Preprocessing in Data Mining* (Springer).

José Manuel Benítez (M'98) received the M.S. and Ph.D. degrees in computer science from the University of Granada, Granada, Spain.

He is currently an Associate Professor with the Department of Computer Science and Artificial Intelligence, University of Granada. His current research interests include time series analysis and modeling, distributed/parallel computational intelligence, data mining, and statistical learning theory. He has published in journals such as the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, *Fuzzy Sets and Systems*, the *Journal of Statistical Software*, *Information Sciences, Mathware and Soft Computing*, *Neural Networks*, *Applied Intelligence*, the *Journal of Intelligent Information Systems*, *Artificial Intelligence in Medicine*, *Expert Systems with Applications*, *Computer Methods and Programs in Biomedicine*, *Evolutionary Intelligence*, and *Econometric Reviews.*

Francisco Herrera received the M.Sc. and Ph.D. degrees in mathematics from the University of Granada, Granada, Spain, in 1988 and 1991, respectively.

He is currently a Professor with the Department of Computer Science and Artificial Intelligence, University of Granada. He has supervised 35 Ph.D. students. His current research interests include bibliometrics, computing with words in decision making, information fusion, evolutionary algorithms, evolutionary fuzzy systems, biometrics, data preprocessing, data mining, cloud computing, and big data. He has published over 290 papers in international journals.

Prof. Herrera was a recipient of several awards, including the European Coordinating Committee for Artificial Intelligence Fellow 2009, the International Fuzzy Systems Association Fellow 2013, the 2010 Spanish National Award on Computer Science ARITMEL to the "Spanish Engineer on Computer Science," the International Cajastur "Mamdani" Prize for Soft Computing (4th ed., 2010), the IEEE TRANSACTIONS ON FUZZY SYSTEM Outstanding 2008 Paper Award (bestowed in 2011), the 2011 Lotfi A. Zadeh Prize Best Paper Award of the International Fuzzy Systems Association, and the 2013 Asociación Española para la Inteligencia Artificial Award to a scientific career in artificial intelligence in 2013. He is currently an Editor-in-Chief of the international journals *Information Fusion* (Elsevier) and *Progress in Artificial Intelligence* (Springer). He is an Editorial Board Member of several journals, such as the *International Journal of Computational Intelligence Systems*, the IEEE TRANSACTIONS ON FUZZY SYSTEMS, *Information Sciences*, *Knowledge and Information Systems*, *Fuzzy Sets and Systems*, *Applied Intelligence*, *Knowledge-Based Systems*, and *Swarm and Evolutionary Computation.*

1.4 Data discretization: taxonomy and big data challenge

- S. Ramírez-Gallego, S. García, H. Mouriño Talín, D. Martínez-Rego, V. Bolón-Canedo, A. Alonso-Betanzos, J. M. Benítez, F. Herrera, Data discretization: taxonomy and big data challenge. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 6 (1) (2016) 5–21, doi: 10.1002/widm.1173I.
  - Status: **Published**.
  - Impact Factor (JCR 2016): 2.111
  - Subject Category: Computer Science, Artificial Intelligence. Ranking 53 / 133 (**Q2**).
  - Subject Category: Computer Science, Theory & Methods. Ranking 29 / 104 (**Q2**).

Advanced Review

Data discretization: taxonomy and big data challenge

Sergio Ramírez-Gallego,[1] Salvador García,[1*] Héctor Mouriño-Talín,[2] David Martínez-Rego,[2,3] Verónica Bolón-Canedo,[2] Amparo Alonso-Betanzos,[2] José Manuel Benítez[1] and Francisco Herrera[1]

Discretization of numerical data is one of the most influential data preprocessing tasks in knowledge discovery and data mining. The purpose of attribute discretization is to find concise data representations as categories which are adequate for the learning task retaining as much information in the original continuous attribute as possible. In this article, we present an updated overview of discretization techniques in conjunction with a complete taxonomy of the leading discretizers. Despite the great impact of discretization as data preprocessing technique, few elementary approaches have been developed in the literature for Big Data. The purpose of this article is twofold: a comprehensive taxonomy of discretization techniques to help the practitioners in the use of the algorithms is presented; the article aims is to demonstrate that standard discretization methods can be parallelized in Big Data platforms such as Apache Spark, boosting both performance and accuracy. We thus propose a distributed implementation of one of the most well-known discretizers based on Information Theory, obtaining better results than the one produced by: the entropy minimization discretizer proposed by Fayyad and Irani. Our scheme goes beyond a simple parallelization and it is intended to be the first to face the Big Data challenge. © 2015 John Wiley & Sons, Ltd

How to cite this article:
WIREs Data Mining Knowl Discov 2016, 6:5–21. doi: 10.1002/widm.1173

INTRODUCTION

Data are present in diverse formats, for example in categorical, numerical, or continuous values. Categorical or nominal values are unsorted, whereas numerical or continuous values are assumed to be sorted or represent ordinal data. It is well-known that data mining (DM) algorithms depend very much on the domain and type of data. In this way, the techniques belonging to the field of statistical learning work with numerical data (i.e., support vector machines and instance-based learning) whereas symbolic learning methods require inherent finite values and also prefer to perform a branch of values that are not ordered (such as in the case of decision trees or rule induction learning). These techniques are either expected to work on discretized data or to be integrated with internal mechanisms to perform discretization.

The process of discretization has aroused general interest in recent years[1,2] and has become one of the most effective data preprocessing techniques in DM.[3] Roughly speaking, discretization translates quantitative data into qualitative data, procuring a nonoverlapping division of a continuous domain. It also ensures an association between each numerical value and a certain interval. Actually, discretization is considered a data reduction mechanism because it diminishes data from a large domain of numeric values to a subset of categorical values.

There is a necessity to use discretized data by many DM algorithms which can only deal with

*Correspondence to: salvagl@decsai.ugr.es

[1]Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain

[2]Department of Computer Science, University of A Coruña, A Coruña, Spain

[3]Department of Computer Science, University College London, London, UK

Conflict of interest: The authors have declared no conflicts of interest for this article.

discrete attributes. For example, three of the ten methods pointed out as the top ten in DM[4] demand a data discretization in one form or another: C4.5,[5] Apriori,[6] and Naïve Bayes.[7] Among its main benefits, discretization causes that the learning methods show remarkable improvements in learning speed and accuracy. Besides, some decision tree-based algorithms produce shorter, more compact, and accurate results when using discrete values.[1,8]

The specialized literature reports on a huge number of proposals for discretization. In fact, some surveys have been developed attempting to organize the available pool of techniques.[1,2,9] It is crucial to determine, when dealing with a new real problem or dataset, the best choice in the selection of a discretizer. This will imply the success and the suitability of the subsequent learning phase in terms of accuracy and simplicity of the solution obtained. In spite of the effort made in Ref 2 to categorize the whole family of discretizers, probably the most well-known and surely most effective are included in a new taxonomy presented in this article, which has now been updated at the time of writing.

Classical data reduction methods are not expected to scale well when managing huge data—both in number of features and instances—so that its application can be undermined or even become impracticable.[10] Scalable distributed techniques and frameworks have appeared along with the problem of Big Data. MapReduce[11] and its open-source version Apache Hadoop[12,13] were the first distributed programming techniques to face this problem. Apache Spark[14,15] is one of these new frameworks, designed as a fast and general engine for large-scale data processing based on in-memory computation. Through this Spark's ability, it is possible to speed up iterative processes present in many DM problems. Similarly, several DM libraries for Big Data have appeared as support for this task. The first one was Mahout[16] (as part of Hadoop), subsequently followed by MLlib[17] which is part of the Spark project.[14] Although many state-of-the-art DM algorithms have been implemented in MLlib, it is not the case for discretization algorithms yet.

In order to fill this gap, we face the Big Data challenge by presenting a distributed version of the entropy minimization discretizer proposed by Fayyad and Irani in Ref 18, using Apache Spark, which is based on Minimum Description Length Principle. Our main objective is to prove that well-known discretization algorithms as MDL-based discretizer (henceforth called MDLP) can be parallelized in these frameworks, providing good discretization solutions for Big Data analytics. Furthermore, we have transformed the iterativity yielded by the original proposal in a single-step computation. This new version for distributed environments has supposed a deep restructuring of the original proposal and a challenge for the authors. Finally, to demonstrate the effectiveness of our framework, we perform an experimental evaluation using two large datasets, namely, ECBDL14 and epsilon.

In order to achieve the goals mentioned above, this article is structured as follows. First, we provide in the next Section (Background and Properties) an explanation of discretization, its properties and the description of the standard MDLP technique. The next Section (Taxonomy) presents the updated taxonomy of the most relevant discretization methods. Afterwards, in the Section *Big Data Background*, we focus on the background of the Big Data challenge including the MapReduce programming framework as the most prominent solution for Big Data. The following section (Distributed MDLP Discretization) describes the distributed algorithm based on entropy minimization proposed for Big Data. The experimental framework, results, and analysis are given in last but one section (Experimental Framework and Analysis). Finally, the main concluding remarks are summarized.

BACKGROUND AND PROPERTIES

Discretization is a wide field and there have been many advances and ideas over the years. This section is devoted to providing a proper background on the topic, including an explanation of the basic discretization process and enumerating the main properties that allow us to categorize them and to build a useful taxonomy.

Discretization Process

In supervised learning, and specifically in classification, the problem of discretization can be defined as follows. Assuming a dataset S consisting of N examples, M attributes, and c class labels, a discretization scheme D_A would exist on the continuous attribute $A \in M$, which partitions this attribute into k discrete and disjoint intervals: $\{[d_0, d_1], (d_1, d_2], \ldots, (d_{k_A - 1}, d_{k_A}]\}$, where d_0 and d_{k_A} are, respectively, the minimum and maximal value, and $P_A = \{d_1, d_2, \ldots, d_{k_A - 1}\}$ represents the set of cut points of A in ascending order.

We can associate a typical discretization as a univariate discretization. Although this property will be reviewed in the next section, it is necessary to introduce it here for the basic understanding of the basic discretization process. Univariate discretization operates with one continuous feature at a time while multivariate discretization considers multiple features simultaneously.

A typical discretization process generally consists of four steps (seen in Figure 1): (1) *sorting* the continuous values of the feature to be discretized, either (2) *evaluating* a cut point for splitting or adjacent intervals for merging, (3) *splitting or merging* intervals of continuous values according to some defined criterion, and (4) *stopping* at some point. Next, we explain these four steps in detail.

- Sorting: The continuous values for a feature are sorted in either descending or ascending order. It is crucial to use an efficient sorting algorithm with a time complexity of $O(NlogN)$. Sorting must be done only once and for the entire initial process of discretization. It is a mandatory treatment and can be applied when the complete instance space is used for discretization.
- Selection of a Cut Point: After sorting, the best cut point or the best pair of adjacent intervals should be found in the attribute range in order to split or merge in a following required step. An evaluation measure or function is used to determine the correlation, gain, improvement in performance, or any other benefit according to the class label.
- Splitting/Merging: Depending on the operation method of the discretizers, intervals either can be split or merged. For splitting, the possible cut points are the different real values present in an attribute. For merging, the discretizer aims to find the best adjacent intervals to merge in each iteration.
- Stopping Criteria: It specifies when to stop the discretization process. It should assume a trade-off between a final lower number of intervals, good comprehension, and consistency.

Discretization Properties

In Ref 1,2,9 various pivots have been used in order to make a classification of discretization techniques. This section reviews and describes them, underlining the major aspects and alliances found among them. The taxonomy presented afterwards will be founded on these characteristics (acronyms of the methods correspond with those presented in Table 1):

- *Static versus Dynamic:* This property refers to the level of independence between the discretizer and the learning method. A static discretizer is run prior to the learning task and is autonomous from the learning algorithm,[1] as a data preprocessing algorithm.[3] Almost all isolated known discretizers are static. By contrast, a

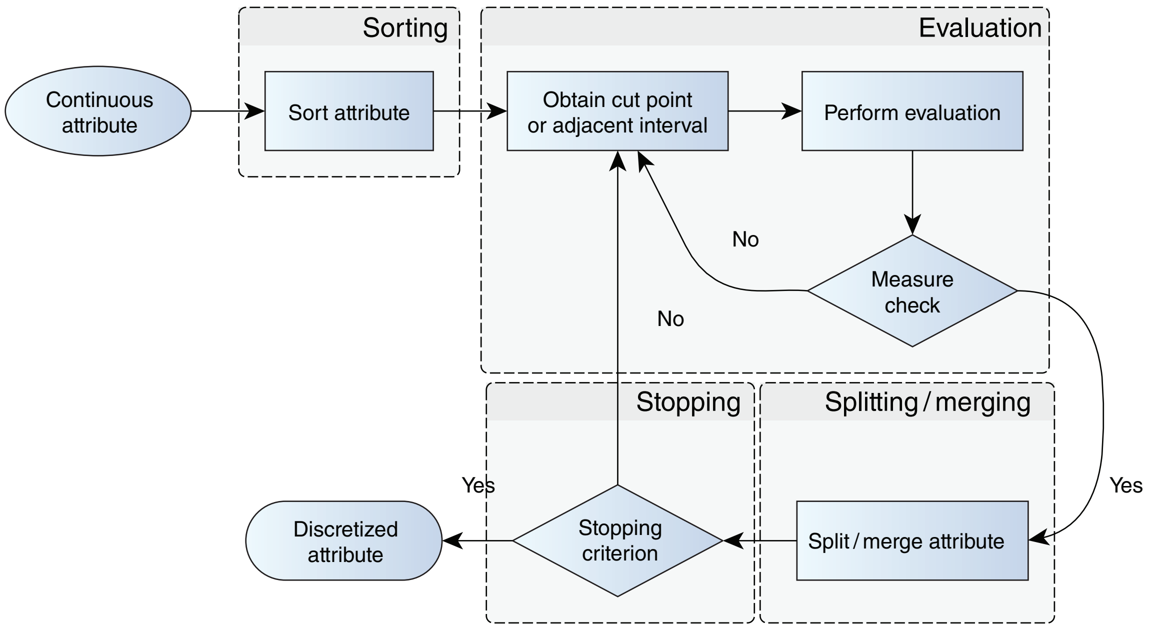


FIGURE 1 | Discretization process.

TABLE 1 | Most Important Discretizers

Acronym	Ref.	Acronym	Ref.	Acronym	Ref.
EqualWidth	19	EqualFrequency	19	Chou91	20
D2	21	ChiMerge	22	1R	23
ID3	5	MDLP	18	CADD	24
MDL-Disc	25	Bayesian	26	Friedman96	27
ClusterAnalysis	28	Zeta	29	Distance	30
Chi2	31	CM-NFD	32	FUSINTER	33
MVD	34	Modified Chi2	35	USD	36
Khiops	37	CAIM	38	Extended Chi2	39
Heter-Disc	40	UCPD	41	MODL	42
ITPF	43	HellingerBD	44	DIBD	45
IDD	46	CACC	47	Ameva	48
Unification	49	PKID	7	FFD	7
CACM	50	DRDS	51	EDISC	52
U-LBG	53	MAD	54	IDF	55
IDW	55	NCAIC	56	Sang14	57
IPD	58	SMDNS	59	TD4C	60
EMD	61				

MDLP, Minimum Description Length Principle.

dynamic discretizer responds when the learner requires so, during the building of the model. Hence, they must belong to the local discretizer's family (see later) embedded in the learner itself, producing an accurate and compact outcome together with the associated learning algorithm. Good examples of classical dynamic techniques are ID3 discretizer[5] and ITFP.[43]

- *Univariate versus Multivariate:* Univariate discretizers only operate with a single attribute simultaneously. This means that they sort the attributes independently, and then, the derived discretization disposal for each attribute remains unchanged in the following phases. Conversely, multivariate techniques, concurrently consider all or various attributes to determine the initial set of cut points or to make a decision about the best cut point chosen as a whole. They may accomplish discretization handling the complex interactions among several attributes to decide also the attribute in which the next cut point will be split or merged. Currently, interest has recently appeared in developing multivariate discretizers because they are decisive in complex predictive problems where univariate operations may ignore important interactions between attributes[60,61] and in deductive learning.[58]
- *Supervised versus Unsupervised:* Supervised discretizers consider the class label whereas unsupervised ones do not. The interaction between the input attributes and the class output and the measures used to make decisions on the best cut points (entropy, correlations, etc.) will define the supervised manner to discretize. Although most of the discretizers proposed are supervised, there is a growing interest in unsupervised discretization for descriptive tasks.[53,58] Unsupervised discretization can be applied to both supervised and unsupervised learning, because its operation does not require the specification of an output attribute. Nevertheless, this does not occur in supervised discretizers, which can only be applied over supervised learning. Unsupervised learning also opens the door to transferring the learning between tasks because the discretization is not tailored to a specific problem.
- *Splitting versus Merging:* These two options refer to the approach used to define or generate new intervals. The former methods search for a cut point to divide the domain into two intervals among all the possible boundary points. On the contrary, merging techniques begin with a predefined partition and search for a candidate cut point to mix both adjacent intervals after removing it. In the literature, the terms

top-down and *bottom-up* are highly related to these two operations, respectively. In fact, top-down and bottom-up discretizers are thought for hierarchical discretization developments, so they consider that the process is incremental, property which will be described later. Splitting/merging is more general than top-down/bottom-up because it is possible to have discretizers whose procedure manages more than one interval at a time.[44,46] Furthermore, we consider the *hybrid* category as the way of alternating splits with merges during running time.[24,61]

- *Global versus Local:* In the time a discretizer must select a candidate cut point to be either split or merged, it could consider either all available information in the attribute or only partial information. A local discretizer makes the partition decision based only on partial information. MDLP[18] and ID3[5] are classical examples of local methods. By definition, all the dynamic discretizers and some top-down-based methods are local, which explains the fact that few discretizers apply this form. The dynamic discretizers search for the best cut point during internal operations of a certain DM algorithm, thus it is impossible to examine the complete dataset. Besides, top-down procedures are associated with the divide-and-conquer scheme, in such manner that when a split is considered, the data is recursively divided, restricting access to partial data.
- *Direct versus Incremental:* For direct discretizers, the range associated with an interval must be divided into k intervals simultaneously, requiring an additional criterion to determine the value of k. One-step discretization methods and discretizers which select more than a single cut point at every step are included in this category. However, incremental methods begin with a simple discretization and pass through an improvement process, requiring an additional criterion to determine when it is the best moment to stop. At each step, they find the best candidate boundary to be used as a cut point and, afterwards, the rest of the decisions are made accordingly.
- *Evaluation Measure:* This is the metric used by the discretizer to compare two candidate discretization schemes and decide which is more suitable to be used. We consider five main families of evaluation measures:
  - *Information:* This family includes *entropy* as the most used evaluation measure in discretization (MDLP,[18] ID3,[5] FUSINTER[33]) and others derived from information theory (*Gini index*, *Mutual information*).[49]
  - *Statistical:* Statistical evaluation involves the measurement of dependency/correlation among attributes (Zeta,[29] ChiMerge,[22] Chi2[31]), interdependency,[38] probability and Bayesian properties[26] (MODL[42]), contingency coefficient,[47] etc.
  - *Rough Sets:* This class is composed of methods that evaluate the discretization schemes by using rough set properties and measures,[59] such as class separability, lower and upper approximations, etc.
  - *Wrapper:* This collection comprises methods that rely on the error provided by a classifier or a set of classifiers that are used in each evaluation. Representative examples are MAD,[54] IDW,[55] and EMD.[61]

  Binning: In this category of techniques, there is no evaluation measure. This refers to discretizing an attribute with a predefined number of bins in a simple way. A bin assigns a certain number of values per attribute by using a non-sophisticated procedure. EqualWidth and EqualFrequency discretizers are the most well-known unsupervised binning methods.

Minimum Description Length-Based Discretizer

Minimum Description Length-based discretizer,[18] proposed by Fayyad and Irani in 1993, is one of the most important splitting methods in discretization. This univariate discretizer uses the MDLP to control the partitioning process. This also introduces an optimization based on a reduction of whole set of candidate points, only formed by the *boundary points* in this set.

Let $A(e)$ denote the value for attribute A in the example e. A boundary point $b \in Dom(A)$ can be defined as the midpoint value between $A(u)$ and $A(v)$, assuming that in the sorted collection of points in A, two examples exist $u, v \in S$ with different class labels, such that $A(u) < b < A(v)$; and the other example $w \in S$ does not exist, such that $A(u) < A(w) < A(v)$. The set of boundary points for attribute A is defined as B_A.

This method also introduces other important improvements. One of them is related to the number of cut points to derive in each iteration. In contrast to discretizers such as ID3,[5] the authors proposed a multi-interval extraction of points demonstrating that better classification models—both in error rate and simplicity—are yielded by using these schemes.

It recursively evaluates all boundary points, computing the class entropy of the partitions derived as quality measure. The objective is to minimize this measure to obtain the best cut decision. Let b_α be a boundary point to evaluate, $S_1 \subset S$ be a subset where $\forall\, a' \in S_1$, $A(a') \leq b_\alpha$, and S_2 be equal to $S - S_1$. The class information entropy yielded by a given binary partitioning can be expressed as:

$$EP(A, b_\alpha, S) = \frac{|S_1|}{|S|}E(S_1) + \frac{|S_2|}{|S|}E(S_2), \tag{1}$$

where E represents the class entropy[a] of a given subset following Shannon's definitions.[62]

Finally, a decision criterion is defined in order to control when to stop the partitioning process. The use of MDLP as a decision criterion allows us to decide whether or not to partition. Thus a cut point b_α will be applied iff:

$$G(A, b_\alpha, S) > \frac{\log_2(N-1)}{N} + \frac{\Delta(A, b_\alpha, S)}{N}, \tag{2}$$

where $\Delta(A, b_\alpha, S) = \log_2(3^c) - [cE(S) - c_1E(S_1) - c_2E(S_2)]$, c_1 and c_2 the number of class labels in S_1 and S_2, respectively; and $G(A, b_\alpha, S) = E(S) - EP(A, b_\alpha, S)$.

TAXONOMY

Currently, more than 100 discretization methods have been presented in the specialized literature. In this section, we consider a subgroup of methods which can be considered the most important from the whole set of discretizers. The criteria adopted to characterize this subgroup are based on the repercussion, availability, and novelty they have. Thus, the precursory discretizers which have served as inspiration to others, those which have been integrated in software suites and the most recent ones are included in this taxonomy.

Table 1 enumerates the discretizers considered in this article, providing the name abbreviation and reference for each one. We do not include the descriptions of these discretizers in this article. Their definitions are contained in the original references, thus we recommend consulting them in order to understand how the discretizers of interest work. In Table 1, 30 discretizers included in KEEL software are considered. Additionally, implementations of these algorithms in Java can be found.[63]

In the previous section, we studied the properties which could be used to classify the discretizers proposed in the literature. Given a predefined order among the seven characteristics studied before, we can build taxonomy of discretization methods. All techniques enumerated in Table 1 are collected in the taxonomy depicted in Figure 2. It represents a hierarchical categorization following the next arrangement of properties: static/dynamic, univariate/multivariate, supervised/unsupervised, splitting/merging/hybrid, global/local, direct/incremental, and evaluation measure.

The purpose of this taxonomy is twofold. First, it identifies a subset of most representative state-of-the-art discretizers for both researchers and practitioners who want to compare them with novel techniques or require discretization in their applications. Second, it characterizes the relationships among techniques, the extension of the families and possible gaps to be filled in future developments.

When managing huge data, most of them become impracticable in real-world settings, due to the complexity they cause (for example, in the case of MDLP, among others). The adaptation of these classical methods implies a thorough redesign that becomes mandatory if we want to exploit the advantages derived from the use of discrete data on large datasets.[64,65] As reflected in our taxonomy, no relevant methods in the field of Big Data have been proposed to solve this problem. Some works have tried to deal with large-scale discretization. For example, in Ref 66, the authors proposed a scalable implementation of Class-Attribute Interdependence Maximization algorithm by using GPU technology. In Ref 67, a discretizer based on windowing and hierarchical clustering is proposed to improve the performance of classical tree-based classifiers. However, none of these methods have been proved to cope with the data magnitude presented here.

BIG DATA BACKGROUND

The ever-growing generation of data on the Internet is leading us to managing huge collections using data analytics solutions. Exceptional paradigms and algorithms are thus needed to efficiently process these datasets so as to obtain valuable information, making this problem one of the most challenging tasks in Big Data analytics.

Gartner[68] introduced the concept of Big Data and the 3V terms that define it as high volume,

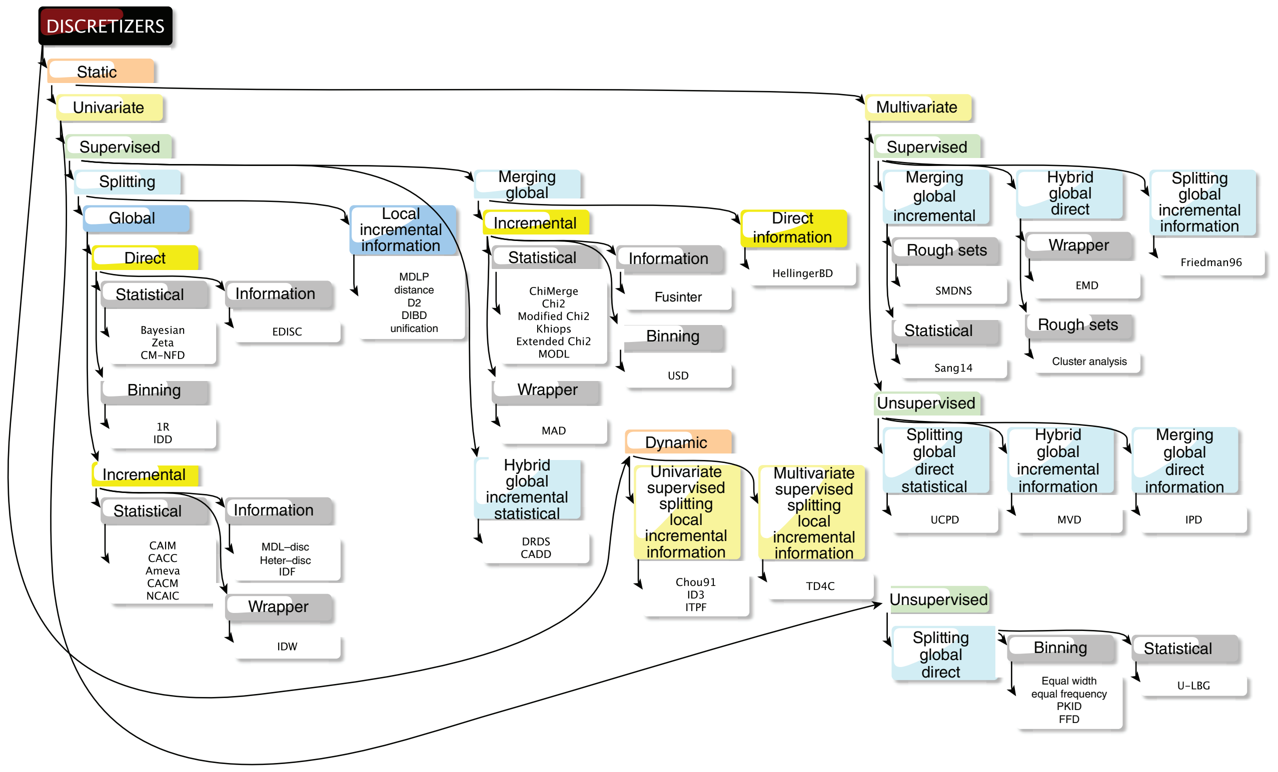


FIGURE 2 | Discretization taxonomy.

velocity, and variety of information that require a new large-scale processing. This list was then extended with two additional terms. All of them are described in the following: *Volume*, the massive amount of data that is produced every day is still exponentially growing (from terabytes to exabytes); *Velocity*, data need to be loaded, analyzed, and stored as quickly as possible; *Variety*, data come in many formats and representations; *Veracity*, the quality of data to process is also an important factor. The Internet is full of missing, incomplete, ambiguous, and sparse data; *Value*, extracting value from data is also established as a relevant objective in big analytics.

The unsuitability of many knowledge extraction algorithms in the Big Data field has meant that new methods have been developed to manage such amounts of data effectively and at a pace that allows value to be extracted from them.

MapReduce Model and Other Distributed Frameworks

The MapReduce framework,[11] designed by Google in 2003, is currently one of the most relevant tools in Big Data analytics. It was aimed at processing and generating large-scale datasets, automatically processed in an extremely distributed fashion through several machines.[b] The MapReduce model defines two primitives to work with distributed data: *Map* and *Reduce*. These two primitives imply two stages in the distributed process, which we describe below. In the first step, the master node breaks up the dataset into several splits, distributing them across the cluster for parallel processing. Each node then hosts several Map threads that transform the generated key-value pairs into a set of intermediate pairs. After all Map tasks have finished, the master node distributes the matching pairs across the nodes according to a key-based partitioning scheme. Then the Reduce phase starts, combining those coincident pairs so as to form the final output.

Apache Hadoop[12,13] is presented as the most popular open-source implementation of MapReduce for large-scale processing. Despite its popularity, Hadoop presents some important weaknesses, such as poor performance on iterative and online computing, and a poor intercommunication capability or inadequacy for in-memory computation, among others.[70] Recently, Apache Spark[14,15] has appeared and integrated with the Hadoop ecosystem. This novel framework is presented as a revolutionary tool capable of performing even faster large-scale processing than Hadoop through in-memory primitives, making this framework a leading tool for iterative and online processing and, thus, suitable for DM algorithms. Spark is built on distributed data structures called resilient distributed datasets (RDDs), which were designed as a fault-tolerant collection of elements that can be operated in parallel by means of data partitioning.

DISTRIBUTED MDLP DISCRETIZATION

In the Background section, a discretization algorithm based on an information entropy minimization heuristic was presented.[18] In this work, the authors proved that multi-interval extraction of points and the use of boundary points can improve the discretization process, both in efficiency and error rate. Here, we adapt this well-known algorithm for distributed environments, proving its discretization capability against real-world large problems.

One important point in this adaption is how to distribute the complexity of this algorithm across the cluster. This is mainly determined by the two time-consuming operations: on one hand, the sorting of candidate points, and, on the other hand, the evaluation of these points. The sorting operation exhibits a $O(|A|\log(|A|))$ complexity (assuming that all points in A are distinct), whereas the evaluation conveys a $O(|B_A|^2)$ complexity. In the worst case, it implies a complete evaluation of entropy for all points.

Note that the previous complexity is bounded to a single attribute. To avoid repeating the previous process on all attributes, we have designed our algorithm to sort and evaluate all points in a single step. Only when the number of boundary points in an attribute is higher than the maximum per partition, computation by feature is necessary (which is extremely rare according to our experiments).

Spark primitives extend the idea of MapReduce to implement more complex operations on distributed data. In order to implement our method, we have used some extra primitives from Spark's API, such as: mapPartitions, sortByKey, flatMap, and reduceByKey.[c]

Main Discretization Procedure

Algorithm 1 explains the main procedures in our discretization algorithm. The algorithm calculates the minimum-entropy cut points by feature according to the MDLP criterion. It uses a parameter to limit the maximum number of points to yield.

The first step creates combinations from instances through a Map function in order to separate values by feature. It generates tuples with the value and the index for each feature as key and a class counter as value ($< (A, A(s)), v >$). Afterwards, the tuples are reduced using a function that aggregates all subsequent vectors with the same key, obtaining the class frequency for each distinct value in the dataset. The resulting tuples are sorted by key

Algorithm 1: Main discretization procedure

```
Input: $S$ Data set
Input: $M$ Feature indexes to discretize
Input: $mb$ Maximum number of cut points to select
Input: $mc$ Maximum number of candidates per partition
Output: Cut points by feature
1: $comb \leftarrow$
2:   map $s \in S$
3:     $v \leftarrow zeros(|c|)$
4:     $ci \leftarrow class\_index(v)$
5:     $v(ci) \leftarrow 1$
6:     for all $A \in M$ do
7:       $EMIT < (A, A(s)), v >$
8:     end for
9:   end map
10: $distinct \leftarrow reduce(comb,\ sum\_vectors)$
11: $sorted \leftarrow sort\_by\_key(distinct)$
12: $first \leftarrow first\_by\_part(sorted)$
13: $bds \leftarrow get\_boundary(sorted, first)$
14: $bds \leftarrow$
15:   map $b \in bds$
16:     $< (att, point), q > \leftarrow b$
17:     $EMIT < (att, (point, q)) >$
18:   end map
19: $(SM, BI) \leftarrow divide\_atts(bds, mc)$
20: $sth \leftarrow$
21:   map $sa \in SM$
22:     $th \leftarrow select\_ths(SM(sa), mb, mc)$
23:     $EMIT < (sa, th) >$
24:   end map
25: $bth \leftarrow ()$
26: for all $ba \in BI$ do
27:   $bth \leftarrow bth + select\_ths(ba, mb, mc)$
28: end for
29: $return(union(bth, sth))$
```

so that we obtain the complete list of distinct values ordered by feature index and feature value. This structure will be used later to evaluate all these points in a single step. The first point by partition is also calculated (line 11) for this process. Once such information is saved, the process of evaluating the boundary points can be started.

Boundary Points Selection

Algorithm 2 (*get_boundary*) describes the function in charge of selecting those points falling in the class borders. It executes an independent function on each partition in order to parallelize the selection process as much as possible so that a subset of tuples is fetched in each thread. The evaluation process is described as follows: for each instance, it evaluates whether the feature index is distinct from the index of the previous point; if it is so, this emits a tuple with the last point as key and the accumulated class counter as value. This means that a new feature has appeared, saving the last point from the current feature as its last threshold. If the previous condition is not satisfied, the algorithm checks whether the current point is a boundary with respect to the previous point or not. If it is so, this emits a tuple with the midpoint between these points as key and the accumulated counter as value.

Finally, some evaluations are performed over the last point in the partition. This point is compared with the first point in the next partition to check whether there is a change in the feature index—emitting a tuple with the last point saved, or not emitting a tuple with the midpoint (as described above). All tuples generated by the partition are then joined into a new mixed RDD of boundary points, which is returned to the main algorithm as *bds*.

In Algorithm 1 (line 14), the *bds* variable is transformed by using a Map function, changing the previous key to a new key with a single value: the feature index (< (*att*, (*point*, *q*)) >). This is done to group the tuples by feature so that we can divide them into two groups according to the total number of candidate points by feature. The *divide_atts* function is then aimed to divide the tuples in two groups (*big* and *small*) depending on the number of candidate points by feature (count operation). Features in each group will be filtered and treated differently according to whether the total number of points for a given feature exceeds the threshold *mc* or not. Small features will be grouped by key so that these can be processed in a parallel way. The subsequent tuples are now reformatted as follows: (< *point*, *q* >).

MDLP Evaluation

Features in each group are evaluated differently from that mentioned before. Small features are evaluated in a single step where each feature corresponds with a single partition, whereas big features are evaluated

Algorithm 2: Function to generate the boundary points *(get_boundary)*

```
Input: $points$ An RDD of tuples ($<(att, point), q>$), where $att$ represents
    the feature index, $point$ the point to consider and $q$ the class counter.
Input: $first$ A vector with all first elements by partition
Output: An RDD of points.
1:  $boundaries \leftarrow$
2:    map partitions $part \in points$
3:      $<(la, lp), lq> \leftarrow next(part)$
4:      $accq \leftarrow lq$
5:      for all $<(a, p), q> \in part$ do
6:        if $a <> la$ then
7:          $EMIT <(la, lp), accq>$
8:          $accq \leftarrow ()$
9:        else if $is\_boundary(q, lq)$ then
10:         $EMIT <(la, (p + lp)/2), accq>$
11:         $accq \leftarrow ()$
12:       end if
13:       $<(la, lp), lq> \leftarrow <(a, p), q>$
14:       $accq \leftarrow accq + q$
15:     end for
16:     $index \leftarrow get\_index(part)$
17:     if $index < npartitions(points)$ then
18:       $<(a, p), q> \leftarrow first(index + 1)$
19:       if $a <> la$ then
20:         $EMIT <(la, lp), accq>$
21:       else
22:         $EMIT <(la, (p + lp)/2), accq>$
23:       end if
24:     else
25:       $EMIT <(la, lp), accq>$
26:     end if
27:   end map
28: $return(boundaries)$
```

iteratively because each feature corresponds with a complete RDD with several partitions. The first option is obviously more efficient, however, the second case is less frequent due to the fact the number of candidate points for a single feature fits perfectly in one partition. In both cases, the *select_ths* function is applied to evaluate and select the most relevant cut points by feature. For small features, a Map function is applied independently to each partition (each one represents a feature) (*arr_select_ths*). In case of big features, the process is more complex and each feature needs a complete iteration over a distributed set of points (*rdd_select_ths*).

Algorithm 3 (*select_ths*) evaluates and selects the most promising cut points grouped by feature according to the MDLP criterion (single-step version). This algorithm starts by selecting the best cut point in the whole set. If the criterion accepts this selection, the point is added to the result list and the current subset is divided into two new partitions using this cut point. Both partitions are then evaluated, repeating the previous process. This process finishes when there is no partition to evaluate or the number of selected points is fulfilled.

Algorithm 4 (*arr_select_ths*) explains the process that accumulates frequencies and then selects the minimum-entropy candidate. This version is more straightforward than the RDD version as it only needs to accumulate frequencies sequentially. First, it obtains the total class counter vector by aggregating all candidate vectors. Afterwards, a new iteration is necessary to obtain the accumulated counters for the two partitions generated by each point. This is done by aggregating the vectors from the most-left point to the current one, and from the current point to the right-most point. Once the accumulated counters for each candidate point are calculated (in form of $< point, q, lq, rq >$), the algorithm evaluates the candidates using the *select_best* function.

Algorithm 5 (*rdd_select_ths*) explains the selection process; in this case for 'big' features (more than one partition). This process needs to be performed in a distributed manner because the number of candidate points exceeds the maximum size defined. For each feature, the subset of points is hence redistributed in a better partition scheme to homogenize the quantity of points by partition and node (*coalesce* function, line 12). After that, a new parallel function is started to compute the accumulated counter by partition. The results (by partition) are then aggregated to obtain the total accumulated frequency for the whole subset. In line 9, a new distributed process is started with the aim of computing the accumulated frequencies at points on both sides (as explained in Algorithm 4). In this procedure, the process accumulates the counter from all previous partitions to the current one to obtain the first accumulated value (the left one). Then, the function computes the accumulated values for each inner point using the counter for points in the current partition, the left value, and the total values (line 7). Once these values are calculated ($< point, q, lq, rq >$), the algorithm evaluates all candidate points and their associated accumulators using the *select_best* function (as above).

Algorithm 6 evaluates the discretization schemes yielded by each point by computing the

Algorithm 3: Function to select the best cut points for a given feature *(select_ths)*

```
Input: cands A RDD/array of tuples (< point, q >), where point represents a
       candidate point to evaluate and q the class counter.
Input: mb Maximum number of intervals or bins to select
Input: mc Maximum number of candidates to eval in a partition
Output: An array of thresholds for a given feature
1:  $st \leftarrow enqueue(st, (candidates, ()))$
2:  $result \leftarrow ()$
3:  while $|st| > 0$ & $|result| < mb$ do
4:      $(set, lth) \leftarrow dequeue(st)$
5:      if $|set| > 0$ then
6:          if $type(set) = 'array'$ then
7:              $bd \leftarrow arr\_select\_ths(set, lth)$
8:          else
9:              $bd \leftarrow rdd\_select\_ths(set, lth, mc)$
10:         end if
11:         if $bd <> ()$ then
12:             $result \leftarrow result + bd$
13:             $(left, right) \leftarrow divide(set, bd)$
14:             $st \leftarrow enqueue(st, (left, bd))$
15:             $st \leftarrow enqueue(st, (right, bd))$
16:         end if
17:     end if
18: end while
19: $return(sort(result))$
```

Algorithm 4: Function to select the best cut point according to MDLP criterion (single-step version) *(arr_select_ths)*

```
Input: $cands$ An array of tuples ($<point, q>$), where $point$ represents a
    candidate point to evaluate and $q$ the class counter.
Output: The minimum-entropy cut point
1: $total \leftarrow sum\_freqs(cands)$
2: $lacc \leftarrow ()$
3: for $< point, q > \in cands$ do
4:     $lacc \leftarrow lacc + q$
5:     $freqs \leftarrow freqs + (point, q, lacc, total - lacc)$
6: end for
7: $return(select\_best(cands, freqs))$
```

Algorithm 5: Function that selects the best cut points according to MDLP criterion (RDD version) *(rdd_select_ths)*

```
Input: $cands$ An RDD of tuples ($< point, q >$), where $point$ represents a
    candidate point to evaluate and $q$ the class counter.
Input: $mc$ Maximum number of candidates to eval in a partition
Output: The minimum-entropy cut point
1: $npart \leftarrow round(|cands|/mc)$
2: $cands \leftarrow coalesce(cands, npart)$
3: $totalpart \leftarrow$
4:     map partitions $partition \in cands$
5:         $return(sum(partition))$
6:     end map
7: $total \leftarrow sum(totalpart)$
8: $freqs \leftarrow$
9:     map partitions $partition \in cands$
10:        $index \leftarrow get\_index(partition)$
11:        $ltotal \leftarrow ()$
12:        $freqs \leftarrow ()$
13:        for $i = 0$ $until$ $index$ do
14:            $ltotal \leftarrow ltotal + totalpart(i)$
15:        end for
16:        for all $< point, q > \in partition$ do
17:            $freqs \leftarrow freqs + (point, q, ltotal + q, total - ltotal)$
18:        end for
19:        $return(freqs)$
20:    end map
21: $return(select\_best(cands, freqs))$
```

entropy for each partition generated, also taking into account the MDLP criterion. Thus, for each point,[d] the entropy is calculated for the two generated partitions (line 8) as well as the total entropy for the whole set (lines 12). Using these values, the entropy gain and the MDLP condition are computed for each point, according to Eq. (2). If the point is accepted by MDLP, the algorithm emits a tuple with the weighted entropy average of partition and the point itself. From the set of accepted points, the algorithm selects the one with the minimum class information entropy.

The results produced by both groups (small and big) are joined into the final point set of cut points.

Analysis of efficiency

In this section, we analyze the performance of the main operations that determined the overall performance of our proposal. Note that the first two operations are quite costly from the point of view of network usage, because they imply shuffling data across the cluster (wide dependencies). Nevertheless, once data are partitioned and saved, these remain unchanged. This is exploited by the subsequent steps, which take advantage of the data locality property. Having data partitioned also benefits operations such as groupByKey, where the grouping is performed locally. The list of such operations (showed in Algorithm 1) is presented below:

Algorithm 6: Function that calculates class entropy values and selects the minimum-entropy cut point *(select_best)*

```
Input: $freqs$ An array/RDD of tuples ($< point, q, lq, rq >$), where point represents a candidate point to evaluate, leftq the left accumulated frequency, rightq the right accumulated frequency and q the class frequency counter.
Input: $total$ Class frequency counter for all the elements
Output: The minimum-entropy cut point
1: $n \leftarrow sum(total)$
2: $totalent \leftarrow ent(total, n)$
3: $k \leftarrow |total|$
4: $accp \leftarrow ()$
5: for all $< point, q, lq, rq > \in freqs$ do
6:     $k1 \leftarrow |lq|; k2 \leftarrow |rq|$
7:     $s1 \leftarrow sum(lq); s2 \leftarrow sum(rq);$
8:     $ent1 \leftarrow ent(s1, k1); ent2 \leftarrow ent(s2, k2)$
9:     $partent \leftarrow (s1 * ent1 + s2 * ent2)/s$
10:    $gain \leftarrow totalent - partent$
11:    $delta \leftarrow log_2(3^k - 2) - (k * hs - k1 * ent1 - k2 * ent2)$
12:    $accepted \leftarrow gain > ((log_2(s - 1)) + delta)/n$
13:    if $accepted = true$ then
14:        $accp \leftarrow accp + (partent, point)$
15:    end if
16: end for
17: $return(min(accp))$
```

1. Distinct points (lines 1–10): this is a standard MapReduce operation that fetches all the points in the dataset. The map phase generates and distributes tuples using a hash partitioning scheme (linear distributed complexity). The reduce phase fetches the set of coincident points and sums up the class vectors (linear distributed complexity).
2. Sorting operation (line 11): this operation uses a more complex primitive of Spark: sortByKey. This samples the set and produces a set of bounds to partition this set. Then, a shuffling operation is started to redistribute the points according to the previous bounds. Once data are redistributed, a local sorting operation is launched in each partition (loglinear distributed order).
3. Boundary points (lines 12–13): this operation is in charge of computing the subset candidate of points to be evaluated. Thanks to the data partitioning scheme generated in the previous phases, the algorithm can yield the boundary points for all attributes in a distributed manner using a linear map operation.
4. Division of attributes (lines 14–19): once the reduced set of boundary points is generated, it is necessary to separate the attributes into two sets. To do that, several operations are started to complete this part. All these suboperations are performed linearly using distributed operations.
5. Evaluation of small attributes (lines 20–24): this is mainly formed by two suboperations: one for grouping the tuples by key (done locally thanks to the data locality), and one map operation to evaluate the candidate points. In the map operation, each feature starts an independent process that, like the sequential version, is quadratic. The main advantage here is the parallelization of these processes.
6. Evaluation of big features (lines 26–28): The complexity order for each feature is the same as in the previous case. However, in this case, the evaluation of features is done iteratively.

EXPERIMENTAL FRAMEWORK AND ANALYSIS

This section describes the experiments carried out to demonstrate the usefulness and performance of our discretization solution over two Big Data problems.

Experimental Framework

Two huge classification datasets are employed as benchmarks in our experiments. The first one (hereinafter called *ECBDL14*) was used as a reference at the ML competition of the Evolutionary Computation for Big Data and Big Learning held on July 14, 2014, under the international conference GECCO-2014. This consists of 631 characteristics (including both numerical and categorical attributes) and 32 million instances. It is a binary classification problem where the class distribution is highly imbalanced involving 2% of positive instances. For this problem, the MapReduce version of the Random Over Sampling (ROS) algorithm presented in Ref 71 was applied in order to replicate the minority class instances from the original dataset until the number of instances for both classes was equalized. As a second dataset, we have used *epsilon*, which consists of 500,000 instances with 2000 numerical features. This dataset was artificially created for the Pascal Large Scale Learning Challenge in 2008. It was further preprocessed and included in the LibSVM dataset repository.[72]

Table 2 gives a brief description of these datasets. For each one, the number of examples for training and test (#Train Ex., #Test Ex.), the total number of attributes (#Atts.), and the number of classes (#Cl) are shown. For evaluation purposes, Naïve Bayes[73] and two variants of Decision Tree[74]—with different impurity measures—have been chosen as reference in classification, using the distributed implementations included in MLlib library.[17] The recommended parameters of the classifiers, according to their authors' specification,[e] are shown in Table 3.

As evaluation criteria, we use two well-known evaluation metrics to assess the quality of the underlying discretization schemes. On one hand, *Classification accuracy* is used to evaluate the accuracy yielded by the classifiers—number of examples correctly labeled divided by the total number of examples. On the other hand, in order to prove the time benefits of using discretization, we have employed the overall classification *runtime* (in seconds) in training as well as the overall time in discretization as additional measures.

For all experiments, we have used a cluster composed of 20 computing nodes and 1 master node. The computing nodes hold the following characteristics: 2 processors × Intel Xeon CPU E5-2620, 6 cores per processor, 2.00 GHz, 15 MB cache, QDR InfiniBand Network (40 Gbps), 2 TB HDD, 64 GB RAM. Regarding software, we have used the following configuration: Hadoop 2.5.0-cdh5.3.1 from Cloudera's open-source Apache Hadoop distribution,[f] Apache Spark and MLlib 1.2.0, 480 cores (24 cores/node), 1040 RAM GB (52 GB/node). Spark implementation of the algorithm can be downloaded from the first author' GitHub repository.[g] The design of the algorithm has been adapted to be integrated in MLlib Library.

Experimental Results and Analysis

Table 4 shows the classification accuracy results for both datasets.[h] According to these results, we can assert that using our discretization algorithm as a preprocessing step leads to an improvement in classification accuracy with Naïve Bayes, for the two datasets tested. It is especially relevant in ECBDL14 where there is an improvement of 5%. This shows the importance of discretization in the application of some classifiers such as Naïve Bayes. For the other classifiers, our algorithm is capable of producing the same competitive results as those performed implicitly by the decision trees.

Table 5 shows classification runtime values for both datasets distinguishing whether discretization is applied or not. As we can see, there is a slight improvement in both cases on using MDLP, but not enough significant. According to the previous results, we can state that the application of MDLP is relevant at least for epsilon, where the best accuracy result has been achieved by using Naïve Bayes and our discretizer. For ECBDL14, it is better to use the implicit discretization performed by the decision trees, because our algorithm is more time-consuming and obtains similar results.

Table 6 shows discretization time values for the two versions of MDLP, namely, sequential and distributed. For the sequential version on ECBDL14, the time value was estimated from small samples of this dataset, because its direct application is unfeasible. A graphical comparison of these two versions is shown in Figure 3. Comparing both implementations, we can notice the great advantage of using the distributed version against the sequential one. For ECBDL14, our version obtains a speedup

TABLE 2 | Summary Description for Classification Datasets

Dataset	#Train Ex.	#Test Ex.	#Atts.	#Cl.
Epsilon	400,000	100,000	2000	2
ECBDL14 (ROS)	65,003,913	2,897,917	631	2

TABLE 3 | Parameters of the Algorithms Used

Method	Parameters
Naive Bayes	Lambda = 1.0
Decision Tree—gini (DTg)	Impurity = gini, max depth = 5, max bins = 32
Decision Tree—entropy (DTe)	Impurity = entropy, max depth = 5, max bins = 32
Distributed MDLP	Max intervals = 50, max by partition = 100,000

MDLP, Minimum Description Length Principle.

TABLE 4 | Classification Accuracy Values

Dataset	NB	NB-disc	DTg	DTg-disc	DTe	DTe-disc
ECBDL14	0.6276	**0.7260**	**0.7347**	0.7339	0.7459	**0.7508**
Epsilon	0.6550	**0.7065**	0.6616	**0.6623**	0.6611	**0.6624**

TABLE 5 | Classification Time Values: with Versus w/o Discretization (In Seconds)

Dataset	NB	NB-Disc	DTg	DTg-Disc	DTe	DTe-Disc
ECBDL14	31.06	**26.39**	347.76	**262.09**	281.05	**264.25**
Epsilon	5.72	**4.99**	68.83	**63.23**	74.44	**39.28**

TABLE 6 | Sequential Versus Distributed Discretization Time Values (In Seconds)

Dataset	Sequential	Distributed	Speedup Rate
ECBDL14	295,508	**1087**	271.86
Epsilon	5764	**476**	12.11

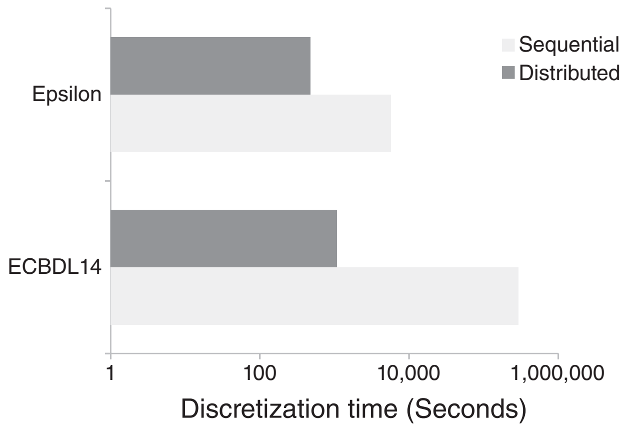


FIGURE 3 | Discretization time: sequential versus distributed (logaritmic scale).

ratio (*speedup = sequential/distributed*) of 271.86, whereas for epsilon, the ratio is equal to 12.11. This shows that the bigger the dataset, the higher the efficiency improvement; and, when the data size is large enough, the cluster can distribute fairly the computational burden across its machines. This is notably the case study of ECBDL14, where the resolution of this problem was found to be impractical using the original approach.

Discretization, as an important part in DM preprocessing, has raised general interest in recent years. In this work, we have presented an updated taxonomy and description of the most relevant algorithms in this field. The aim of this taxonomy is to help the researchers to better classify the algorithms that they use, on one hand, while also helping to identify possible new future research lines. At this respect, and although Big Data is currently a trending topic in science and business, no distributed approach has been developed in the literature, as we have shown in our taxonomy.

Here, we propose a completely distributed version of the MDLP discretizer with the aim of demonstrating that standard discretization methods can be parallelized in Big Data platforms, boosting both performance and accuracy. This version is capable of transforming the iterativity yielded by the original proposal in a single-step computation through a complete redesign of the original version. According to our experiments, our algorithm is capable of performing 270 times faster than the sequential version, improving the accuracy results in all used datasets. For future works, we plan to tackle the problem of discretization in large-scale online problems.

NOTES

[a] Logarithm in base 2 is used in this function.

[b] For a complete description of this model and other distributed models, please review Ref 69.

[c] For a complete description of Spark's operations, please refer to Spark's API: https://spark.apache.org/docs/latest/api/scala/index.html.

[d] If the set is an array, it is used as a loop structure, else it is used as a distributed map function.

[e] https://spark.apache.org/docs/latest/api/scala/index.html.

[f] http://www.cloudera.com/content/cloudera/en/documentation/cdh5/v5-0-0/CDH5-homepage.html.

[g] https://github.com/sramirez/SparkFeatureSelection.

[h] In all tables, the best result by column (best by method) is highlighted in bold.

ACKNOWLEDGMENTS

This work is supported by the National Research Project TIN2014-57251-P, TIN2012-37954, and TIN2013-47210-P, and the Andalusian Research Plan P10-TIC-6858, P11-TIC-7765, and P12-TIC-2958, and by the Xunta de Galicia through the research project GRC 2014/035 (all projects partially funded by FEDER funds of the European Union). S. Ramírez-Gallego holds a FPU scholarship from the Spanish Ministry of Education and Science (FPU13/00047). D. Martínez-Rego and V. Bolón-Canedo acknowledge support of the Xunta de Galicia under postdoctoral Grant codes POS-A/2013/196 and ED481B 2014/164-0.

REFERENCES

1. Liu H, Hussain F, Lim Tan C, Dash M. Discretization: an enabling technique. *Data Min Knowl Discov* 2002, 6:393–423.
2. García S, Luengo J, Antonio Sáez J, López V, Herrera F. A survey of discretization techniques: taxonomy and empirical analysis in supervised learning. *IEEE Trans Knowl Data Eng* 2013, 25:734–750.
3. García S, Luengo J, Herrera F. *Data Preprocessing in Data Mining*. Germany: Springer; 2015.
4. Wu X, Kumar V, eds. *The Top Ten Algorithms in Data Mining*. USA: Chapman & Hall/CRC Data Mining and Knowledge Discovery; 2009.
5. Ross Quinlan J. *C4.5: Programs for Machine Learning*. USA: Morgan Kaufmann Publishers Inc.; 1993.
6. Agrawal R, Srikant R. Fast algorithms for mining association rules. In: *Proceedings of the 20th Very Large Data Bases conference (VLDB)*, Santiago de Chile, Chile, 1994, pages 487–499.
7. Yang Y, Webb GI. Discretization for Naïve-Bayes learning: managing discretization bias and variance. *Mach Learn* 2009, 74:39–74.
8. Hu H-W, Chen Y-L, Tang K. A dynamic discretization approach for constructing decision trees with a continuous label. *IEEE Trans Knowl Data Eng* 2009, 21:1505–1514.
9. Yang Y, Webb GI, Wu X. Discretization methods. In: *Data Mining and Knowledge Discovery Handbook*. Germany: Springer; 2010, 101–116.
10. Wu X, Zhu X, Wu G-Q, Ding W. Data mining with big data. *IEEE Trans Knowl Data Eng* 2014, 26:97–107.
11. Dean J, Ghemawat S. Mapreduce: simplified data processing on large clusters. In: San Francisco, CA, *OSDI*, 2004, pages 137–150.
12. Apache Hadoop Project. Apache Hadoop, 2015. [Online https://hadoop.apache.org/; Accessed March, 2015].
13. White T. *Hadoop, The Definitive Guide*. USA: O'Reilly Media, Inc.; 2012.
14. Apache Spark: lightning-fast cluster computing. Apache spark, 2015. [Online http://spark.apache.org/; Accessed March, 2015].
15. Hamstra M, Karau H, Zaharia M, Konwinski A, Wendell P. *Learning Spark: Lightning-Fast Big Data Analytics*. USA: O'Reilly Media, Incorporated; 2015.
16. Apache Mahout Project. Apache Mahout, 2015. [Online http://mahout.apache.org/; Accessed March, 2015].
17. Machine Learning Library (MLlib) for Spark. Mllib, 2015. [Online https://spark.apache.org/docs/1.2.0/mllib-guide.html; Accessed March, 2015].
18. Fayyad UM, Irani KB. Multi-interval discretization of continuous-valued attributes for classification learning. In: *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, San Francisco, CA, 1993, pages 1022–1029.
19. Wong AKC, Chiu DKY. Synthesizing statistical knowledge from incomplete mixed-mode data. *IEEE Trans Pattern Anal Mach Intell* 1987, 9:796–805.
20. Chou PA. Optimal partitioning for classification and regression trees. *IEEE Trans Pattern Anal Mach Intell* 1991, 13:340–354.
21. Catlett J. On changing continuous attributes into ordered discrete attributes. In: *European Working Session on Learning (EWSL)*. Lecture Notes on Computer Science, vol. 482. Germany: Springer-Verlag; 1991, 164–178.
22. Kerber R. Chimerge: discretization of numeric attributes. In: *National Conference on Artifical Intelligence American Association for Artificial Intelligence (AAAI)*, San Jose, California, 1992, pages 123–128.
23. Holte RC. Very simple classification rules perform well on most commonly used datasets. *Mach Learn* 1993, 11:63–90.
24. Ching JY, Wong AKC, Chan KCC. Class-dependent discretization for inductive learning from continuous and mixed-mode data. *IEEE Trans Pattern Anal Mach Intell* 1995, 17:641–651.

25. Pfahringer B. Compression-based discretization of continuous attributes. In: *Proceedings of the 12th International Conference on Machine Learning (ICML)*, Tahoe City, California, 1995, pages 456–463.
26. Xindong W. A Bayesian discretizer for real-valued attributes. *Comput J* 1996, 39:688–691.
27. Friedman N, Goldszmidt M. Discretizing continuous attributes while learning Bayesian networks. In: *Proceedings of the 13th International Conference on Machine Learning (ICML)*, Bari, Italy, 1996, pages 157–165.
28. Chmielewski MR, Grzymala-Busse JW. Global discretization of continuous attributes as preprocessing for machine learning. *Int J Approx Reason* 1996, 15:319–331.
29. Ho KM, Scott PD. Zeta: a global method for discretization of continuous variables. In: *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD)*, Newport Beach, California, 1997, pages 191–194.
30. Cerquides J, De Mantaras RL. Proposal and empirical comparison of a parallelizable distance-based discretization method. In: *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD)*, Newport Beach, California, 1997, pages 139–142.
31. Liu H, Setiono R. Feature selection via discretization. *IEEE Trans Knowl Data Eng* 1997, 9:642–645.
32. Hong SJ. Use of contextual information for feature ranking and discretization. *IEEE Trans Knowl Data Eng* 1997, 9:718–730.
33. Zighed DA, Rabaséda S, Rakotomalala R. FUSINTER: a method for discretization of continuous attributes. *Int J Unc Fuzz Knowl Based Syst* 1998, 6:307–326.
34. Bay SD. Multivariate discretization for set mining. *Knowl Inform Syst* 2001, 3:491–512.
35. Tay FEH, Shen L. A modified chi2 algorithm for discretization. *IEEE Trans Knowl Data Eng* 2002, 14:666–670.
36. Giráldez R, Aguilar-Ruiz JS, Riquelme JC, Ferrer-Troyano FJ, Rodríguez-Baena DS. Discretization oriented to decision rules generation. In: *Frontiers in Artificial Intelligence and Applications*, vol. 82. Netherlands: IOS press; 2002, 275–279.
37. Boulle M. Khiops: a statistical discretization method of continuous attributes. *Mach Learn* 2004, 55:53–69.
38. Kurgan LA, Cios KJ. CAIM discretization algorithm. *IEEE Trans Knowl Data Eng* 2004, 16:145–153.
39. Chao-Ton S, Hsu J-H. An extended chi2 algorithm for discretization of real value attributes. *IEEE Trans Knowl Data Eng* 2005, 17:437–441.
40. Liu X, Wang H. A discretization algorithm based on a heterogeneity criterion. *IEEE Trans Knowl Data Eng* 2005, 17:1166–1173.
41. Mehta S, Parthasarathy S, Yang H. Toward unsupervised correlation preserving discretization. *IEEE Trans Knowl Data Eng* 2005, 17:1174–1185.
42. Boullé M. MODL: a Bayes optimal discretization method for continuous attributes. *Mach Learn* 2006, 65:131–165.
43. Au W-H, Chan KCC, Wong AKC. A fuzzy approach to partitioning continuous attributes for classification. *IEEE Trans Knowl Data Eng* 2006, 18:715–719.
44. Lee C-H. A Hellinger-based discretization method for numeric attributes in classification learning. *Knowl Based Syst* 2007, 20:419–425.
45. Wu QX, Bell DA, Prasad G, McGinnity TM. A distribution-index-based discretizer for decision-making with symbolic AI approaches. *IEEE Trans Knowl Data Eng* 2007, 19:17–28.
46. Ruiz FJ, Angulo C, Agell N. IDD: a supervised interval Distance-Based method for discretization. *IEEE Trans Knowl Data Eng* 2008, 20:1230–1238.
47. Tsai C-J, Lee C-I, Yang W-P. A discretization algorithm based on class-attribute contingency coefficient. *Inform Sci* 2008, 178:714–731.
48. González-Abril L, Cuberos FJ, Velasco F, Ortega JA. Ameva: an autonomous discretization algorithm. *Expert Syst Appl* 2009, 36:5327–5332.
49. Jin R, Breitbart Y, Muoh C. Data discretization unification. *Knowl Inform Syst* 2009, 19:1–29.
50. Li M, Deng S, Feng S, Fan J. An effective discretization based on class-attribute coherence maximization. *Pattern Recognit Lett* 2011, 32:1962–1973.
51. Gethsiyal Augasta M, Kathirvalavakumar T. A new discretization algorithm based on range coefficient of dispersion and skewness for neural networks classifier. *Appl Soft Comput* 2012, 12:619–625.
52. Shehzad K. EDISC: a class-tailored discretization technique for rule-based classification. *IEEE Trans Knowl Data Eng* 2012, 24:1435–1447.
53. Ferreira AJ, Figueiredo MAT. An unsupervised approach to feature discretization and selection. *Pattern Recognit* 2012, 45:3048–3060.
54. Kurtcephe M, Altay Güvenir H. A discretization method based on maximizing the area under receiver operating characteristic curve. *Intern J Pattern Recognit Artif Intell* 2013, 27.
55. Ferreira AJ, Figueiredo MAT. Incremental filter and wrapper approaches for feature discretization. *Neurocomputing* 2014, 123:60–74.
56. Yan D, Liu D, Sang Y. A new approach for discretizing continuous attributes in learning systems. *Neurocomputing* 2014, 133:507–511.
57. Sang Y, Qi H, Li K, Jin Y, Yan D, Gao S. An effective discretization method for disposing high-dimensional data. *Inform Sci* 2014, 270:73–91.

58. Nguyen H-V, Müller E, Vreeken J, Böhm K. Unsupervised interaction-preserving discretization of multivariate data. *Data Min Knowl Discov* 2014, 28:1366–1397.

59. Jiang F, Sui Y. A novel approach for discretization of continuous attributes in rough set theory. *Knowl Based Syst* 2015, 73:324–334.

60. Moskovitch R, Shahar Y. Classification-driven temporal discretization of multivariate time series. *Data Min Knowl Discov* 2015, 29:871–913.

61. Ramírez-Gallego S, García S, Benítez JM, Herrera F. Multivariate discretization based on evolutionary cut points selection for classification. *IEEE Trans Cybern*. In press. doi:10.1109/TCYB.2015.2410143.

62. Shannon CE. A mathematical theory of communication. ACM *SIGMOBILE Mob Comput Commun Rev* 2001, 5:3–55.

63. Alcalá-Fdez J, Sánchez L, García S, del Jesus MJ, Ventura S, Garrell JM, Otero J, Romero C, Bacardit J, Rivas VM, et al. KEEL: a software tool to assess evolutionary algorithms for data mining problems. *Soft Comput* 2009, 13:307–318.

64. Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. On the effectiveness of discretization on gene selection of microarray data. In: *International Joint Conference on Neural Networks, IJCNN 2010*, Barcelona, Spain, 18–23 July, 2010, pages 1–8.

65. Bolón-Canedo V, Sánchez-Maroño N, Alonso-Betanzos A. Feature selection and classification in multiple class datasets: an application to KDD Cup 99 dataset. *Expert Syst Appl* May 2011, 38:5947–5957.

66. Cano A, Ventura S, Cios KJ. Scalable CAIM discretization on multiple GPUs using concurrent kernels. *J Supercomput* 2014, 69:273–292.

67. Zhang Y, Cheung Y-M. Discretizing numerical attributes in decision tree for big data analysis. In: *ICDM Workshops*, Shenzhen, China, 2014, pages 1150–1157.

68. Beyer M.A., Laney D. 3D data management: controlling data volume, velocity and variety, 2001. [Online http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf; Accessed March, 2015].

69. Fernández A, del Río S, López V, Bawakid A, del Jesús MJ, Benítez JM, Herrera F. Big data with cloud computing: an insight on the computing environment, mapreduce, and programming frameworks. *WIREs Data Min Knowl Discov* 2014, 4:380–409.

70. Lin J. Mapreduce is good enough? If all you have is a hammer, throw away everything that's not a nail!. *Clin Orthop Relat Res* 2012, abs/1209.2191.

71. Rio S, Lopez V, Benitez JM, Herrera F. On the use of mapreduce for imbalanced big data using random forest. *Inform Sci* 2014, 285:112–137.

72. Chang C-C, Lin C-J. LIBSVM: a library for support vector machines. *ACM Trans Intell Syst Technol* 2011, 2:1–27. Datasets available at http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/.

73. Duda RO, Hart PE. *Pattern Classification and Scene Analysis*, vol. 3. New York: John Wiley & Sons; 1973.

74. Quinlan JR. Induction of decision trees. In: Shavlik JW, Dietterich TG, eds. *Readings in Machine Learning*. Burlington, MA: Morgan Kaufmann Publishers; 1990. Originally published in *Machine Learning* 1:81–106, 1986.

1.5 A distributed evolutionary multivariate discretizer for Big Data processing on Apache Spark

- S. Ramírez-Gallego, S. García, J. M. Benítez and F. Herrera, A distributed evolutionary multivariate discretizer for Big Data processing on Apache Spark. Swarm and Evolutionary Computation 38 (2018) 240–250, doi: 10.1016/j.swevo.2017.08.005.
  - Status: **Published**.
  - Impact Factor (JCR 2016): 3.893.
  - Subject Category: Computer Science, Artificial Intelligence. Ranking 19 / 133 (**Q1**).
  - Subject Category: Computer Science, Theory & Methods. Ranking 11 / 104 (**Q1**).

Swarm and Evolutionary Computation xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

A distributed evolutionary multivariate discretizer for Big Data processing on Apache Spark

S. Ramírez-Gallego[b,*], S. García[a,b], J.M. Benítez[b], F. Herrera[b]

[a] *Department of Computer Science and Artificial Intelligence, CITIC-UGR, University of Granada, 18071 Granada, Spain*
[b] *Faculty of Computing and Information Technology, King Abdulaziz University, North Jeddah, Saudi Arabia*

A R T I C L E I N F O

Keywords:
Discretizacion
Evolutionary computation
Big Data
Data Mining
Apache Spark

A B S T R A C T

Nowadays the phenomenon of Big Data is overwhelming our capacity to extract relevant knowledge through classical machine learning techniques. Discretization (as part of data reduction) is presented as a real solution to reduce this complexity. However, standard discretizers are not designed to perform well with such amounts of data. This paper proposes a distributed discretization algorithm for Big Data analytics based on evolutionary optimization. After comparing with a distributed discretizer based on the Minimum Description Length Principle, we have found that our solution yields more accurate and simpler solutions in reasonable time.

1. Introduction

Among all Data Mining tasks, Data Preprocessing [1,2] stands as one of the most important steps in the knowledge discovery process. As input data must be provided in a suitable structure and format for a subsequent high-quality mining process, Data Preprocessing becomes essential in most of data analytic problems. Preparation techniques aim at cleaning negative factors present in current databases –missing, noise, inconsistent and superfluous data–. Conversely, data reduction family is applied to simplify data and their inherent complexity, while maintaining their original structure. Discretization [3,4], as part of data reduction, has received increasing attention in last years. It transforms quantitative data into qualitative data by performing a non-overlapping partitioning of continuous attributes, and then associating a set of discrete values to the resulting partitions.

Although the Data Mining discipline has been successfully applied for several years [5], in this new era of Big Datum [6,7], the capabilities of traditional mining systems have been surpassed by the exponential growth of databases. Learning from large-scale datasets has become a labored or even impracticable task when classical algorithms are used. As in standard mining, Big Data preprocessing [8] plays an essential role in improving the quality of large-scale data. This importance can be deemed even greater in Big Data scenario since large amounts of data usually implies more noise. Novel scalable, and efficient discretizers [4], developed on recent distributed paradigms and tools [9], are thus required to face the Big Data discretization problem. Up to date, only one distributed solution for Big Data discretization [4] has been presented in the literature.

Data discretization can be deemed as an optimization problem, where partial solutions can be coded via binary representation. Given the previous problem, an evolutionary-based metaheuristic [10] can be useful at dealing with binary-based optimization. Although quite effective, evolutionary algorithms are known for being time-consuming and hardly scalable, specially when large-scale problems are faced [11]. A distributed solution based on evolutionary heuristics would bring us an scalable and effective solution for Big Data discretization [12].

Evolutionary algorithms (EA) have shown their usefulness on several optimization-based learning problems, see the following overviews for several mining contexts, such as: rule learning [13], evolutionary fuzzy systems [14], clustering [15], or multi-objective learning [16]. Recently, we can find novel evolutionary and bio-inspirited approaches for learning, such as: data discretization [17], rule induction [18], feature selection [19], clustering [20], or diverse applications, like face recognition [21].

In this paper we propose a novel design for a distributed multivariate discretizer for Apache Spark [22] based on an evolutionary points selection scheme. Our approach, called Distributed Evolutionary Multivariate Discretizer (DEMD), has been inspired by the EMD discretizer [17]. EMD is an evolutionary-based discretizer with binary representation and a wrapper fitness function. Although both algorithms share some common aspects (like representation and fitness function), DEMD goes beyond a simple parallelization, and offers an approximative, scalable and resilient solution to deal with the Big Data discretization problem. Alike EMD, in DEMD, partial solutions are generated locally, and eventually fused to produce the final discretization scheme. Up to our knowledge, our proposal is the first evolutionary

* Corresponding author at: Department of Computer Science and Artificial Intelligence, CITIC-UGR, University of Granada, 18071 Granada, Spain.
E-mail addresses: sramirez@decsai.ugr.es (S. Ramírez-Gallego), salvagl@decsai.ugr.es (S. García), J.M.Benitez@decsai.ugr.es (J.M. Benítez), herrera@decsai.ugr.es (F. Herrera).

http://dx.doi.org/10.1016/j.swevo.2017.08.005
Received 28 December 2016; Received in revised form 19 April 2017; Accepted 19 August 2017

approach in dealing with the large-scale discretization problem.

In order to show the usefulness of our solution, a thorough experimental evaluation has been performed using several huge datasets (up to $O(10^7)$ instances and $O(10^4)$ features). A distributed discretizer based on the Minimum Description Length Principle (called DMDLP) [4] has also been included for comparison purposes. Experiments on these real-world datasets have shown that DEMD obtains more accurate and simpler discretization schemes than its competitor.

The remainder of this paper is organized as follows. Section 2 briefly explains some concepts about Big Data, and the environment of tools and paradigms around this phenomenon. Section 3 introduces the discretization task, some related concepts, as well as a brief description of EMD. Section 4 discusses the complexity problems faced, as well as the solution adopted by our proposal. Section 5 describes the experimental framework carried out, and the results derived from these experiments. Lastly Section 6 gives the conclusions derived from this work.

2. Big Data: concepts, paradigms and tools

In this section, the Big Data phenomenon is introduced through the scheme of 5Vs. Here we also present the distributed frameworks, paradigms and tools which have served to address Big Data problems in a distributed manner.

Humongous amounts of information are stored in data centers now, ready to be processed. The efficient extraction of valuable knowledge from these datasets raises a considerable challenge for data scientists. Gartner [23] introduced the popular concept of Big Data in 2001. In its report, Gartner defines this concept as the conjunction of the 3Vs: high volume, velocity and variety information that require a new large-scale processing. This list was extended with 2 extra terms: veracity and value.

One of the most relevant frameworks in Big Data analytics is the MapReduce framework [24]. This framework, devised by Google in 2003, allows us to automatically process huge data by distributing the complexity burden among a cluster of machines. Final users only have to design their tasks specifying the Map and Reduce functions. Partitioning and distributing of data, job scheduling or fault-tolerance are responsibility of the platform.[1]

One of the most popular open-source implementation of MapReduce is Apache Hadoop [25,26]. Despite of being well-used and very popular, Hadoop seems not to work well with iterative and online processes [27]. In general, it is not intended for those programs that continuously reads data and need to keep them in memory.

The MapReduce model offers two primitives –Map and Reduce–, which correspond with two execution stages in the whole process. Firstly, the master node retrieves the dataset (split into several chunks) from the distributed file system so that each node reads those data chunks allocated in its local disk. Each node then starts one or more Map threads to process the raw chunks. The result is a set of key-value pairs (intermediate pairs), which are also stored in disk. After all Map tasks have ended, the master node starts the Reduce phase by distributing those pairs with coincident keys to the same node. Each Reduce task combines those matching pairs to yield the final output. Fig. 1 depicts a simplified scheme of MapReduce and its two main functions.

Related to the Hadoop Ecosystem, Apache Spark [28,22] has emerged as a new revolutionary tool capable of outperforming Hadoop for certain cases (100x faster). This is possible due to the in-memory primitives available in Spark. This platform allows users to persist data into memory and to read them rapidly, making it suitable for iterative and online jobs.

[1] For a complete review of this model and other distributed models, please check [9].

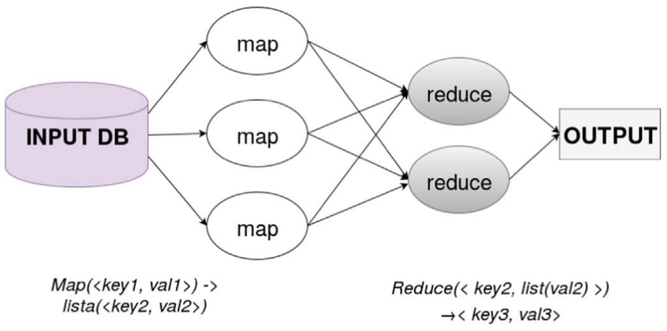

INPUT DB
map
map
map
reduce
reduce
OUTPUT
Map(<key1, val1>) -> lista(<key2, val2>)
Reduce(< key2, list(val2) >) →< key3, val3>

Fig. 1. Scheme of the MapReduce paradigm.

3. Discretization: theoretical background

In this section, the problem of discretization, as well as some optimizations are presented. Additionally, a brief description of EMD is also given.

3.1. Definitions

Discretization is a data preprocessing technique that generates disjoint intervals from continuous features. The resulting intervals are associated to a set of discrete values to yield nominal data. A complete description and taxonomy of discretization algorithms can be found in [3,4].

Given a dataset D with n examples, a set of features F, the subset of continuous features $FC \subset F$, and o target classes, a discretization process would divide a continuous feature c into k_c disjoint and discrete intervals, yielding the following discretization scheme:

$$D_c = \{[d_0, d_1], (d_1, d_2], \dots, (d_{k_c-1}, d_{k_c}]\} \tag{1}$$

where d_0 and d_{k_c} are the minimum and maximum value, respectively. Note that all values in D_c are sorted in ascending order. Likewise,

$$CP_c = \{d_1, d_2, \dots, d_{k_c-1}\} \tag{2}$$

is defined as the set of cut points of feature c, and CP denotes the whole set of cut points for all the continuous features in D.

Optimal discretization can be considered as a NP-hard problem [29], where the search space is basically composed by all the different values (for all features) in the training set. To alleviate the subsequent complexity, it can be considered a reduced subset of points, formed by the *boundary points* among classes.

Let c has its values sorted in ascending order, and val_c be a function that returns the value for c, given an example in D. Given two examples $a, b \in D$ with different classes, such that $val_c(a) < val_c(b)$. If there is no sample $c \in D$ such that $val_c(a) < val_c(c) < val_c(b)$, a boundary point can be defined as the half-point value between $val_c(a)$ and $val_c(b)$. The set of boundary points for c is denoted as BP_c, whereas the complete set as BP.

Boundary points has shown to form the optimal intervals for most of the evaluation measures [30], thanks to that the previous definition always search the maximum separability between classes. Additionally, a reduced subset of points offers significant savings in complexity, as shown in [17]. This fact is specially relevant in Big Data, where the performance is a determining factor.

3.2. Evolutionary multivariate discretizer

An important contribution to the discretization field is EMD [17], an evolutionary-based discretizer that uses a wrapper fitness function to evaluate binary solutions. This approach goes beyond deterministic algorithms, and offers the possibility of improving the discretization schemes generated by tuning several parameters. EMD follows a multivariate approach to leverage the existing dependencies among different features.

EMD utilizes the operators and mechanisms defined in CHC [31] to tackle the cut points selection problem. CHC is a well-known evolutionary algorithm specially designed for binary optimization, which looks for a proper balance between exploration and exploitation. Our algorithm includes several features from CHC in order to achieve this balance, such as: incest prevention (via mating threshold) or chromosome reboot.

EMD consists of two steps that are constantly repeated in each iteration. Firstly, pairs of chromosomes are fused using a crossover operation. Intermediate population is then randomly paired to generate new offspring. Afterwards a survival competition decides what is the best solution from the set of parent and offspring. The result is a new population which will be mixed and selected in further steps.

Like CHC, EMD implements HUX, a heterogeneous crossover operation to recombine chromosomes. HUX is specially designed to generate maximally distant offspring from two parents by exchanging half of the different points according to the Hamming distance. Regarding the mutation operator, EMD replaces it by a rebooting process which randomly changes 35% of the bits in the best chromosome in order to create new templates. This reseeding process will be applied whenever the population gets stuck for a number of evaluations.

This discretizer uses a binary chromosome representation to annotate the selection (1) or not-selection (0) of all the boundary points in BP. There is therefore an unique correspondence between each gene and each boundary point. Finally, all points marked as selected in the best chromosome will be included in S.

In order to evaluate the different binary solutions (chromosomes), EMD defines a wrapper fitness function that aggregates two factors: the classification error on training and the number of cut points selected. This function has showed more promising accuracy results and simpler solutions than those based on inconsistency measures [32]. The objective of this EA is the minimization of this function, which is defined as follows:

$$Fitness(P') = \alpha \cdot \frac{|P'|}{|BP|} + (1 - \alpha) \cdot \Delta \tag{3}$$

where P' is the subset of selected points, Δ the error obtained after classifying the discretized data, and α a weight factor for these two factors.

In EMD, the classification error is computed as the average mean error yielded by two classifiers: Naïve Bayes [33] and an unpruned version of C4.5 [34].

EMD also introduces a reduction mechanism to speed up the convergence of our method, which is based on the reduction of the chromosome size. In our solution, this is done by maintaining those points selected more than a determined number of evaluations (the most relevant ones). In each reduction phase, several points are removed according to a counter that indicates the number of selections by point.

This reduction mechanism fixes the long delays associated to the application of EAs when facing huge problems. As any optimization problem, the cut points selection problem offers multiple valid solutions (local optima). It is thus convenient to reach some local optima in order to avoid long executions.

4. Distributed evolutionary multivariate discretizer

This section explains the design of our distributed discretization solution for Big Data. Our algorithm splits both the set of cut points and instances into partitions, and evaluates them through a cross-evaluation system. With this distributed scheme we maximize the resource usage throughout the entire process. If a point is selected by one of the evaluation processes, it counts as a single vote. All these votes are aggregated to obtain the final score per point. Finally, the final discretization scheme is obtained through a voting scheme.

Section 4.1 starts discussing the main concerns that affect our distributed approach: a large number of instances and cut points. In Section 4.2, the main procedure in charge of partitioning the instances are feature, and aggregating the partial solutions is presented. Section 4.3 illustrates the process of computing boundary points. Section 4.4 exposes how the chromosomes are evaluated in a distributed manner by using EMD.

4.1. Discussion about the DEMD's distributed design

This section presents the main problems that our proposal needs to overcome to produce discretization schemes efficiently. The two problems to consider are: a high number of cut points to evaluate, and therefore, long chromosomes to evaluate; and a huge amount of instances to use in this evaluation phase.

The first problem is related to the high complexity derived from EA problems. In the cut points selection problem, discretizers are mainly affected by the number of boundary points to evaluate (long chromosomes). In particular, this problem is influenced by two factors: the number of instances and features present in the problem. Another hidden factor that influences the complexity is the number of distinct points present in each feature. If this value is high, the algorithm will have to process a high number of boundary points.

In order to keep the multivariate philosophy and to alleviate the complexity derived from these two problems, our distributed proposal has introduced some major changes with respect to the sequential version. The first change we propose is to divide the complete set of features into partitions so that the evaluation of points is performed in a parallel way. This modification has also demonstrated to maintain the effectiveness of the original method.

For the second problem (high number of instances), we propose to partition the set of instances into a set of equal-sized partitions. Each data partition will serve to evaluate different parts of the chromosome. Once the partitions have been evaluated following the EMD scheme (Section 3.2), the subsequent partial solutions are aggregated through a voting scheme (complete description in Section 4.4). This modification has showed to work well with large datasets, even when the generated schemes are approximative.

Regarding the evaluation, Naïve Bayes has been elected to evaluate the candidate solutions in the fitness function because of its simplicity and efficiency in its close-form expression [35] (linear order). Nevertheless, users can introduce another classifier/s to customize the distributed approach.

To implement our method, some extra primitives from Spark's API have been used. Spark primitives implement more complex operations than those proposed by MapReduce. Some of them are: mapPartitions, broadcast, sortByKey, Map and reduceByKey.[2]

DEMD includes several user-defined input parameters, which are described in Table 1.

4.2. Main discretization procedure

Procedure 1 explains the main procedure of our discretization algorithm. Hereafter we will use the term partition to describe the data partitions, and the term chunk to describe the feature partitions. This procedure is in charge of distributing the initial cut points (computed in Section 4.3) among the set of chunks. The partitions already created are associated with these chunks so that each chunk is evaluated on the instances contained in one or more partitions. After the parallel selection process is performed (in Section 4.4), this procedure creates

[2] For a complete description of Spark's operations, please refer to Spark's API: https://spark.apache.org/docs/latest/api/scala/index.html.

the final matrix of selected cut points.

The first step computes the boundary points (BF) in a distributed way using the function *getBoundary* (line 1, Section 4.3). Each tuple in BF consists of a feature ID (fid) and a list of points. Based on this variable, DEMD creates FI (feature information), and BP (boundary points per feature). All this information will serve us to create the chromosome chunks.

Procedure 1. Main discretization procedure.

```
Input:  $D$ dataset
Input:  $M$ Feature indexes to discretize
Input:  $uf$ Multivariate user factor
Input:  $alp$ Alpha parameter
Input:  $ne$ Number of evaluations
Input:  $sr$ Sampling rate
Input:  $vp$ Percentage of selected points
Output: Cut points by feature
 1:  $BF \leftarrow getBoundary(D, M)$
 2:  $BP \leftarrow ();\ FI \leftarrow ();$
 3:  for all $<fid, l> \in BF$ do
 4:    $BP(fid) = l$
 5:    $FI.add(Feature(fid, l.size))$
 6:  end for
 7:  $FI \leftarrow broadcast(sortBySize(FI))$
 8:  $BP \leftarrow broadcast(BP)$
 9:  $nbp \leftarrow totalSize(BP)$
10:  $ds \leftarrow nbp/D.npartitions$
11:  $ms \leftarrow max(FI(0).size, ds)$
12:  $df \leftarrow max(uf, ms/ds)$
13:  $ncp \leftarrow nbpoints/(df*ds)$
14:  $windows \leftarrow makeGroups(FI, ncp)$
15:  $CH \leftarrow ()$
16:  for all $w \in windows$ do
17:    $p \leftarrow shuffle(w)$
18:    for $i = 0 \rightarrow i < p.size$ do
19:        $CH(i).add(p(i))$
20:    end for
21:  end for
22:  $CH \leftarrow broadcast(CH)$
23:  $SD \leftarrow stratifiedSampling(D, sr)$
24:  $SP \leftarrow select(SD, CH, uf, alp, sr, vp)$
25:  $TH \leftarrow ()$
26:  for $<chid, lf> \in SP$ do
27:    $ind \leftarrow 0;\ chunk \leftarrow CH(chid)$
28:    for $feat \in chunk$ do
29:        for $i = 0 \rightarrow i < feat.size$ do
30:          if $lf(i + ind) == true$ then
31:              $point \leftarrow BP(feat.id)(i + ind)$
32:              $TH(feat.id).add(point)$
33:          end if
34:        end for
35:          $ind \leftarrow ind + feat.size$
36:    end for
37:  end for
38:  $return(TH)$
```

The procedure divides the evaluation of cut points using subsets of features (called chunks) (lines 2–13). To do that DEMD first sorts all features by the number of boundary points contained in each one (ascending order). Then, DEMD computes the number of chunks (ncp) in which the entire list of boundary points will be divided. ncp is computed using several variables which are related according to Eq. (4).

Table 1
DEMD's parameters. For each parameter, name, description and range are shown.

Parameter	Description	Range
D	Input dataset (RDD)	–
M	Feature indexes to discretize	$[0, f]$
uf	Ratio between the number of feature chunks and the number of data partitions	$[1, \infty)$
alp	Weight factor for the fitness function	$[0, 1]$
ne	Number of chromosome evaluations to be performed in each process	$[100, \infty)$
sr	Percentage of instances used in evaluation	$[0, 1]$
vp	Percentage of points selected in each aggregation process	$[0, 1]$

$$ncp = np/(max(uf, ms/ds)\cdot ds) \tag{4}$$

where np is the total number of boundary points, ds the current proportion of points by data partition, uf the split factor specified by the user, and ms the maximum between the largest feature size and ds.

Usually each feature is contained in a single chunk, but it may change in case the user specifies a greater value, or the largest feature surpasses the default size since points belonging to the same feature can not be separated. In the latter case, a finer-grained division will be performed, which means more chunks. This scenario normally entails a quicker evaluation, but a loss in effectiveness.

The evaluation procedure starts to distribute points between the chunks (CH) (lines 14-21). In each iteration, a group of nc features is collected and randomly distributed among the chunks. The loop ends when there is no feature to collect. This mechanism will enable a fairly distribution of boundary points, without points from the same feature in different chunks, and with a similar number of features per chunk.

Once the distribution of points is completed, a stratified sampling process (by class) is performed on D (line 23). The resulting sample SD is used to evaluate the boundary points in a distributed manner. According to the multivariate factor ($max(uf, ms/ds)$), each partition randomly selects as many chunks as indicated by these factor (usually only one). Then, each partition is responsible of evaluating the points contained in their associated chunks (line 24). The selection phase is described in detail in Section 4.4.

Each selection process returns its aggregated partial solution (the best chromosome per chunk), and saves the tuples (chunk ID, best solution) in SP. All these partial results are then summarized using a voting scheme, considering the threshold (vp). Finally, the main procedure processes the binary vectors to obtain the final matrix of cut points (TH) (line 26-38). This procedure fetches the features in each chunk, and its correspondent points. If a given point has been selected, it is added to the final matrix. If not, this is omitted.

An illustrative scheme of the entire process is detailed in Fig. 2. In this example, there are four features with different amounts of boundary points (8, 5, 4,10). Boundary points are then uniformly distributed into three chunks where features may be mixed, like in chunk $C1$. Afterwards chromosome chunks are grouped with seven data partitions following a correspondence table that relates chunks and partitions according to the multivariate factor. Once local evaluation threads have ended, partial discretization results (binary vectors) for the same chromosome part are aggregated by summing votes. Most-voted points in each chunk according to vp (proportion of points to select) are selected, and adapted to create the global selection matrix.

4.3. Computing the boundary points

Procedure 2. Function to generate the boundary points (*getBoundary*).

```
Input:   $D$ dataset
Input:   $M$ Feature indexes to discretize
Output:  The set of boundary points (feature index, point value).
1:   $CB \leftarrow$
2:     map $s \in D$
3:       $v \leftarrow zeros(|c|)$
4:       $ci \leftarrow classIndex(v)$
5:       $v(ci) \leftarrow 1$
6:       for all $A \in M$
7:           $EMIT < (A, A(s)), v>$
8:       end for
9:     end map
10:  $D \leftarrow reduce(CB, sumVectors)$
11:  $S \leftarrow sortByKey(D)$
12:  $FP \leftarrow firstByPart(S)$
13:  $BP \leftarrow$
14:    map partitions $PT \in S$
15:        $<(la, lp), lq > \leftarrow next(PT)$
16:        for all $<(a, p), q > \in PT$ do
17:          if $a <> la$ then
18:              $EMIT < la, lp>$
19:          else if $isBoundary(q, lq)$ then
20:              $EMIT < la, (p + lp)/2>$
21:          end if
22:          $<la, lp > \leftarrow < a, p>$
23:      end for
24:      $index \leftarrow getIndex(PT)$
25:      if $index < npartitions(S)$ then
26:          $<(a, p), q > \leftarrow FP(index + 1)$
27:          if $a <> la$ then
28:              $EMIT < la, lp>$
29:          else
30:              $EMIT < la, (p + lp)/2>$
31:          end if
32:      else
33:          $EMIT < la, lp>$
34:      end if
35:    end map
36:  $return(BP.groupByKey())$
```

Procedure 2 (*getBoundary*) describes the function that computes border points in data. This procedure consists of three steps. Firstly, the distinct points (D) in the dataset are calculated by removing duplicated elements. Secondly, the resulting points are sorted (S) and distributed by feature index so that all the points from the same feature will not be separated. Finally, the boundary points (BP) in each feature are evaluated sequentially.

The procedure starts by launching a parallel process on each partition (taking advantage of data locality) with the aim of computing the distinct points (CB) (lines 1-9). Once the points are sorted (D) and the first point by partition is distributed (FP), DEMD evaluates whether each points belong to any border as follows (lines 13–36): for each point, it checks whether the feature index is distinct from the index of the previous point; if it is so, DEMD generates a tuple with the feature index of the last point as key, and its correspondent value as value. By doing so, the last point from the current feature is always kept as the last threshold. If there are more points in this feature, the procedure evaluates whether the current point accomplishes the boundary condition with respect to the previous point. If it is so, this generates a tuple with the feature index as key, and the midpoint between these two points as value.

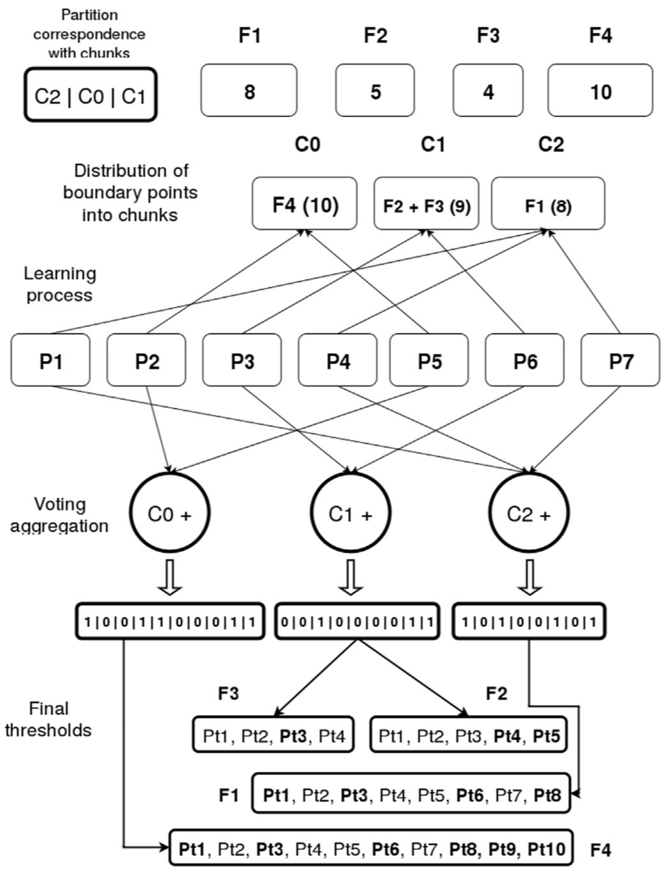


Fig. 2. A simplified representation of the DEMD process. F represent the features, C the chromosome chunks, P the dataset partitions to evaluate, and Pt the boundary points. The selected points have been highlighted in bold.

The last point in each partition is considered as an special case (lines 25–34). These points are compared with the first point in the following partition (broadcasted). If the feature indexes are different, the procedure emits a tuple with the last point. If not and the point is boundary, DEMD emits a tuple with the midpoint between these two points. Finally, all the tuples generated in each partition are joined into a RDD of boundary points, which is returned to the main procedure.

The previous process is depicted in Fig. 3. In this figure we can see three partitions with six different points. The points are sorted by key (feature index and point value) to perform the evaluation. The first point for each partition is sent to the following partition to perform the evaluation of the last points. As result, three boundary points are generated, some are midpoints and some are the last points in features.

4.4. Distributed cut points selection

Procedure 3 explains the distributed operations used to aggregate the solutions generated by each local evaluation process, and to decide the final discretization scheme. Note that local evaluation of points is performed by launching a single instance of EMD on each data partition (Section 3.2). This process consists of two steps: the first one starts a selection process (map) on each pair chunk-partition and aggregates the subsequent solutions to produce the final number of votes. The second step is aimed at selecting the most voted points by chunk according to the threshold (vp) defined by the user.

Procedure 3. Function to perform the evolutionary selection process *(select).*

```
Input:   $SM$ Sampled boundary points
Input:   $CH$ Feature chunks
Input:   $uf$ Multivariate user factor
Input:   $alpha$ Alpha parameter (evolutionary process)
Input:   $ne$ Number of evaluations (evolutionary process)
Input:   $sr$ Sampling rate
Input:   $vp$ Percentage of selected points
Output:  Cut points by feature
1:  $PO \leftarrow shuffle(seq(0, CH.size))$
2:  $R \leftarrow$
3:    map partitions $<index, DT> \in SM$
4:        $chid \leftarrow PO(index\%CH.size)$
5:        $C \leftarrow CH(chid)$
6:        $npoints \leftarrow totalSize(chunk)$
7:        $CP \leftarrow ()$; $FD \leftarrow ()$
8:        for $i = 0 \rightarrow i < chunk.size$ do
9:          $FD(i) \leftarrow DT(i)$
10:         $CP(i) \leftarrow C(i)$
11:       end for
12:       $BI \leftarrow EMD(FD, CP, alpha, ne)$
13:       $CO \leftarrow ()$
14:       for $i = 0 \rightarrow i < BI.size$ do
15:         if $BI(i) == 1$ then
16:             $CO(i) = 1$
17:         else
18:             $CO(i) = 0$
19:         end if
20:       end for
21:       EMIT $< chid, (CO, 1)>$
22:   end map
23: $CO \leftarrow R.reduceByKey(sum())$
24: $SL \leftarrow$
25:   map $<chid, (AC, c)> \in CO$
26:       $S \leftarrow sort(AC)$
27:       $ps \leftarrow AC.size*vp$
28:       $BA \leftarrow take(S, ps)$
29:       EMIT $< chid, BA>$
30:   end map
31: $return(SL)$
```

Firstly, each chunk is associated with one or more data partitions using PO, which is a table formed by tuples (chunk ID, data partition). For each tuple, a map operation is started (lines 3–22). This map operation starts by creating a data matrix with the instances contained on each partition and those features present in the chunk. Afterwards, the procedure executes an evaluation thread on each submatrix (FD) in order to evaluate the corresponding boundary points (CP). As result, the best chromosome (a binary vector) in the population is returned (BI).

The binary vector will be transformed into a numeric vector to annotate number of selections (CO) (lines 13–22). The final result emitted by the partition is a tuple with the identifier of the chunk as key, and the vector count –number of times each point has been selected– and a chunk count –maximum number of partitions in which has been evaluated– as value. This procedure will indicate the selection ratio for each point. The partial values generated above are aggregated by reducing the tuples by key.

Secondly, the procedure starts a map operation (lines 25–30) to select the most voted points by chunk (SL). The procedure orders all the points by number of votes, and selects in order as much points as specified by vp. The result is a tuple with chunk ID as key, and the selection vector as value. Finally, previous results will eventually be transformed to a matrix of points in the main procedure.

4.5. Computational and communication complexity analysis

In this section we analyze the computational and communication complexity for all procedures presented. Big O notation is used to specify the upper limit for the run-time and communication cost of each procedure.

- Boundary points (line 1 – Algorithm 2): complexity here is determined by the computation of distinct points: $O(\frac{|D|\cdot|M|}{nc})$ for run-time, and $O(|D|\cdot|M|)$ for communication. nc represents the total number of cores used to distribute the complexity burden.
- Selection process (line 24 – Algorithm 3): a single evolutionary selection process is executed on each partition. The overall computational complexity is linear: $O(ne\cdot\frac{|D|\cdot|BP|\cdot df}{nc})$, and it is mainly bounded by Naive Bayes's complexity ($O(|D|\cdot|BP|)$). The procedure communicates $O(|BP|)$ integer data.
- Main algorithm (Algorithm 1): all sequential operations here are linear ($O(|BP|)$), as well as the communication processes between the nodes and the master node ($O(|BP|)$).

Notice that, in most of cases, the number of boundary points to be processed and communicated is much lower than the number of original points according to the Table 6 (#Pt.). This fact allows us to say that our algorithm can perform efficiently in many large-scale problems.

5. Experimental framework and results

This section describes the experimental framework carried out and analyzes the results derived from these experiments. The aim of these experiments is to prove the benefit derived from using our discretization solution. DMDLP, an distributed discretizer based on entropy minimization, is included in the experiments for comparison purposes.

5.1. Datasets and methods

In these experiments, we have used four large-scale classification datasets as benchmarks. The largest dataset in our framework is ECBDL14. This dataset was used as benchmark at the international conference GECCO-2014, in an classification competition for Big Data. This consists of 32 million instances with a high imbalance ratio: 98% of negative instances. To equalize both classes, the MapReduce version of the Random OverSampling (ROS) technique [36] was used to replicate the minority class (henceforth called ECBDL14R). This version has been used in the experiments instead of the original one.

From the LibSVM dataset repository [37], the dataset *epsilon* has been used as example of artificially created (and noisy) dataset with many features and boundary points. The rest of datasets (*higgs* and *susy*) have been taken from the UCI Machine Learning Repository [38]. *susy* is also an imbalanced problem with a ratio of 34%. All datasets presented in this section are two-class problems.

Table 2 gives a short description of these datasets. For each one, the number of examples for training and test (#Train Ex., #Test Ex.), the total number of attributes (#Atts.), and the number of classes (#Cl) are shown. In order to reduce the number of candidate points, all data has been rounded up to four decimal places. It affects to problems like higgs or epsilon where the number of decimal places is large enough.

Naïve Bayes has been elected as the reference classifier to assess the quality of solutions. Namely, the distributed version of Naïve Bayes in MLlib [39] have been chosen for the experiments. In Table 3, the recommended parameters (according to their authors' specification)

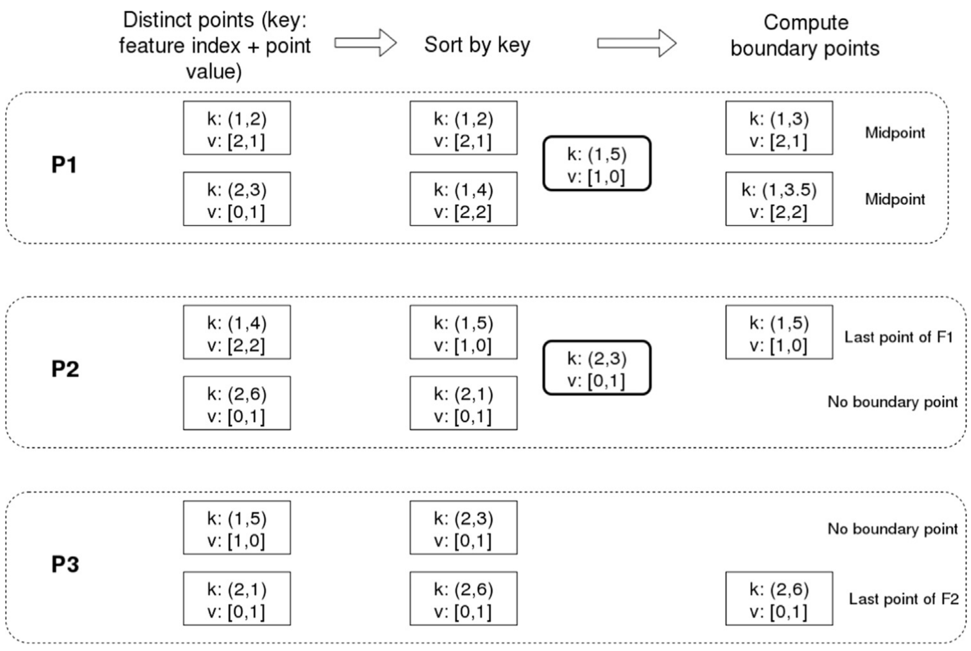


Fig. 3. Distributed computation of boundary points. P represents the partitions. The points broadcasted have been highlighted in bold.

Table 2
Summary description of datasets. For each one, the number of examples in training and test (#Train Ex., #Test Ex.), the total number of features (#Atts.), and the number of continuous features (#Cont.) are shown.

Dataset	#Train Ex.	#Test Ex.	#Atts.	#Cont.
ECBDL14R	65,003,913	2,897,917	630	539
higgs	8,800,000	2,200,000	28	28
susy	4,000,000	1,000,000	18	18
epsilon	400,000	100,000	2000	2000

Table 3
Parameters of the algorithms used.

Method	Parameters
DEMD	$\alpha = 0.5$, $sr = 1.0$, $vp = \{0.25, 0.5, 1.0\}$, $ne = 10,000$
Distributed MDLP	Max cut points = 15, max by partition = 100,000
Naïve Bayes	Lambda = 1.0

for this algorithm[3] is shown.

The distributed discretizer DMDLP [4] has been included in the experiment for comparison purposes. The parameter values for both discretizers are also defined in Table 3. For some special cases, some modifications to these parameters have been introduced, as explained in Section 5.4. For all experiments, five executions for each pair method-dataset have been launched to assess the quality of the solutions generated by our non-deterministic algorithm. The details of all runs are reported in Appendix A.

As evaluation measures, three standard metrics have been used to assess the performance of the discretizers and the quality of the subsequent solutions. The classification accuracy and the Area Under Curve Receiver Operating Characteristic (AUC-ROC) have been used for quality evaluation of test set. The overall discretization time has also been used to measure the quickness of discretizers.

A cluster of machines of twenty computing nodes and a master node was used to accomplish the experiments. All nodes hold the following features: 2 processors x Intel Xeon CPU E5-2620, 6 cores per processor, 2.00 GHz, 15 MB cache, QDR InfiniBand Network (40 Gbps), 2 TB HDD, 64 GB RAM. The software installed on these machines was the following: Hadoop 2.5.0-cdh5.3.1 from Cloudera's open-source Apache Hadoop distribution,[4] Apache Spark and MLlib 1.5.0, 460 cores (23 cores/node), 960 RAM GB (48 GB/node). The source code of DEMD, designed to be integrated in MLlib, can be downloaded from the correspondent author' GitHub account.[5]

5.2. Analysis of classification performance

In this section, the classification performance of our discretizer is evaluated against two classifiers and several huge datasets. The discretization schemes generated by our solution and another alternative are used as a preprocessing step before the classification phase.

In Table 4, the average classification results on test after applying Naïve Bayes are shown. Before classifying, the datasets have been discretized in a preprocessing stage using both discretizers. As can be seen in this table, the accuracy results yielded by our method outperforms those yielded by DMDLP in 4 out of 5 cases. This is specially remarkable for *ECBDL14R* (the biggest dataset), with a difference of several tenths. Notice that a slight improvement in accuracy in these large-scale problems could imply a high number of instances is correctly classified (1300 instances for susy).

Likewise, we have measured the impact of discretization in classifying two imbalanced datasets: *ECBDL14R* and *susy*. Table 5 shows the AUC results on the test set. No remarkable difference between both methods can be seen in this table, but only a slight advantage for DMDLP.

Beyond the improvement in accuracy, our solution has shown to yield simpler discretization schemes, with far lower number of points. These simpler solutions, apart from being much more understandable for experts, also have a positive impact on the learning process (from

[3] https://spark.apache.org/docs/latest/api/scala/index.html.
[4] http://www.cloudera.com/content/cloudera/en/documentation/cdh5/v5-0-0/CDH5-homepage.html.
[5] https://github.com/sramirez/.

Table 4
Classification test accuracy by discretizer and dataset.

Method		*ECBDL14R*	*higgs*	*epsilon*	*susy*
DEMD - 0.25	*Avg*	0.6912	**0.5960**	0.6734	0.7133
	Std-Dev	0.0127	0.0131	0.0106	0.0077
DEMD - 0.5	*Avg*	0.7215	0.5680	0.6752	0.7175
	Std-Dev	0.0141	0.0256	0.0098	0.0102
DEMD - 1.0	*Avg*	**0.7500**	0.5630	0.6718	**0.7406**
	Std-Dev	0.0107	0.0261	0.0185	0.0033
DMDLP	*Avg*	0.7272	0.5933	**0.7047**	0.7393
	Std-Dev	0.00	0.00	0.00	0.00

Table 5
Test AUC by discretizer and dataset.

Method		*ECBDL14R*	*susy*
DEMD - 0.25	*Avg*	0.5113	0.7006
	Std-Dev	0.0007	0.0076
DEMD - 0.5	*Avg*	0.5128	0.7048
	Std-Dev	0.0007	0.0060
DEMD - 1.0	*Avg*	**0.5143**	**0.7157**
	Std-Dev	0.0006	0.0032
DMDLP	*Avg*	0.5131	0.7126
	Std-Dev	0.00	0.00

Table 6
Number of cut points generated by discretizer and dataset. The best solution for each dataset according to Naïve Bayes is highlighted in bold, whereas the best one for DEMD is highlighted in italic.

Method	*ECBDL14R*	*higgs*	*epsilon*	*susy*
DEMD - 0.25	210	**248**	240	247
DEMD - 0.5	462	496	**495**	494
DEMD - 1.0	**959**	1000	990	988
DMDLP	8624	410	1718	**267**

Table 7
Discretization time by discretizer and dataset (in seconds).

Method		*ECBDL14R*	*higgs*	*epsilon*	*susy*
DEMD - 0.25	*Avg*	1178.70	733.26	1850.80	268.84
	Std-Dev	25.66	60.54	15.95	3.42
DEMD - 0.5	*Avg*	1181.42	771.75	1858.94	271.20
	Std-Dev	18.23	74.46	13.21	7.23
DEMD - 1.0	*Avg*	1169.80	764.68	1861.83	271.22
	Std-Dev	20.97	51.01	22.92	6.26
DMDLP	*Avg*	**975.80**	**36.01**	**117.44**	**23.45**
	Std-Dev	0.00	0.00	0.00	0.00

Table 8
Information derived from data and chromosome partitioning tasks. For each dataset, the original chromosome size (#Pt.), the computed multivariate factor (Mvf.), the maximum feature size (Mfs.), the final chunk size (Cs.), and the number chunks generated (#Ch.) are shown.

Dataset	**#Pt.**	**Mvf.**	**Mfs.**	**Cs.**	**#Ch.**
ECBDL14R	41,937	10.82	985	985	42
higgs	514,524	52,96	59,214	59,214	8
susy	573,792	33.71	42,046	42,046	13
epsilon	3,013,813	1.00	2555	2555	460
epsilon[*]	3,013,813	30.00	76,650	76,650	15

[*] Epsilon without uf multivariate factor set to 30.

both time and accuracy points). This is essential in Big Data environments where efficiency and simplicity are a plus. Table 6 illustrates the simplicity of these solutions. The results prove that the most accurate solutions for DEMD are also those with a lower number of points, except for *susy*.

5.3. Analysis of efficiency

Another aspect when evaluating discretizers is the efficiency in generating the discretization schemes. This is specially important in Big Data environments, where the quickness is an important factor. This section presents a comparison between DEMD and DMDLP, in terms of time used to obtain the discretization model.

Table 7 illustrates this comparison by presenting the average time results for both algorithms. For all cases, DMDLP performs much faster than our method due to its greedy iterative nature. Nevertheless, our solution offers competitive results. All of them far below a limit of one hour, which are quite reasonable in Big Data analytics.

5.4. Case study: explosive growth of chromosomes and use of sampling

An overwhelming number of candidate points and instances to evaluate are the two most important problems when dealing with large-scale datasets and EAs. This section aims at showing how our approach can be tuned to deal with these problems, quite common in some big datasets.

The number of candidates points to be evaluated straightly determines the chromosome size that has to be managed by the EA. In classical learning problems, this size can range to 15,000, as we verified in a our study [17] where a long list of UCI datasets was analyzed. Big datasets though presents a complexity (number of points) much higher than presented in small/medium datasets, as shown in Table 8. Despite some preprocessing stage has been applied to these datasets (as presented in 5.1), the chromosome size can go to several millions of genes. This is the case of *epsilon*, in which the EA starts with 3,013,813 points to evaluate.

Table 8 shows the value of the factors and variables which are implied in the data partitioning and the creation of point chunks. It is specially remarkable the *epsilon* case where the number of chunks corresponds with the number of data partitions (460). This case represents the simplest case of voting so that all chunks will be evaluated by a single data partition, which implies a clear degradation on the overall performance of the discretizer. In the experiments, the multivariate factor variable uf was changed for *epsilon* in order to cope with this problem (marked with an asterisk in the table). uf was then established to 30, a similar value to that present in *susy* (the closest problem in number of points to *epsilon*).

EAs are also affected by the sample size. In our case, the wrapper classifier used in our EA needs to evaluate each solution using the complete set of instances. Even after partitioning the points into chunks, the size of chromosomes remains quite complex for the fitness evaluation. In order to make feasible this evaluation for big datasets (like ECBDL14R), some simplification techniques could be applied to alleviate this complexity. One of them is the stratified sampling of instances. In our algorithm, this technique is applied just after computing the candidate points so as to only use this sample to evaluate solutions.

According to the previous idea, a stratified sampling was applied on ECBDL14R, the biggest dataset in terms of number of instances. The sampling rate *sr* was then established to 0.1 in order to equalize the performance of our solutions for all datasets used (see Table 7). Furthermore, the accuracy results confirms that even using a reduced sample of instances, there is a considerable improvement in classification results.

6. Conclusions

In this paper, we have presented DEMD, a distributed multivariate discretization algorithm for Big Data based on evolutionary optimization under Apache Spark. Our solution is aimed at optimizing the cut points selection problem by selecting accurate and simple solutions. In this version, a new system of cross-evaluation between partitions of instances and points has been introduced. Despite its non-deterministic nature, this kind of evaluation offers promising discretization schemes.

The experimental results obtained on big datasets (up to $O(10^7)$ instances and $O(10^4)$ features) have shown the improvement on accuracy and simplicity when using DEMD. Our approach also allows to tune the simplicity/accuracy rate of the generated solutions using several parameters.

Our future work will concentrate on showing that evolutionary computation can also be useful in dealing with other Big Data preprocessing tasks [12], such as feature or instance selection. We will also envision that our approach can be adapted to the streaming environment where discretization schemes evolve over time, and concept drifts might affect them [40].

Acknowledgements

This work is supported by the Spanish National Research Project TIN2014-57251-P, the Foundation BBVA project 75/2016 BigDaPTOOLS, and the Andalusian Research Plan P11-TIC-7765. S. Ramírez-Gallego holds a FPU scholarship from the Spanish Ministry of Education and Science (FPU13/00047).

Appendix A. Detailed classification results on test

In this section, we present the detailed results (by execution) derived from test classification. Tables A.9, A.10, A.11, and A.12 show the acurracy results for all datasets, whereas Tables A.13 and A.14 show the results on AUC for the two imbalanced problems used in the experiments (*ECBDL14R* and *susy*).

Table A.9
Test accuracy obtained for ECBDL14R.

Method - *vp*	*Ex. #1*	*Ex. #2*	*Ex. #3*	*Ex. #4*	*Ex. #5*	**Avg**	**Std-Dev**
DEMD - 0.25	0.6923	0.696	0.6832	0.6757	0.7089	0.6912	0.0127
DEMD - 0.5	0.7172	0.7262	0.7434	0.7068	0.7137	0.7215	0.0141
DEMD - 1.0	0.7456	0.7619	0.7483	0.7587	0.7353	**0.7500**	**0.0107**
DMDLP	0.7272	–	–	–	–	0.7272	0.0000

Table A.10
Test accuracy obtained for higgs.

Method - *vp*	*Ex. #1*	*Ex. #2*	*Ex. #3*	*Ex. #4*	*Ex. #5*	**Avg**	**Std-Dev**
DEMD - 0.25	0.6172	0.587	0.5926	0.5844	0.5987	**0.5960**	**0.0131**
DEMD - 0.5	0.5741	0.5891	0.5614	0.5884	0.5271	0.5680	0.0256
DEMD - 1.0	0.5249	0.5507	0.5705	0.5927	0.5762	0.5630	0.0261
DMDLP	0.5933	–	–	–	–	0.5933	0.0000

Table A.11
Test accuracy obtained for epsilon.

Method - *vp*	*Ex. #1*	*Ex. #2*	*Ex. #3*	*Ex. #4*	*Ex. #5*	**Avg**	**Std-Dev**
DEMD - 0.25	0.6599	0.6871	0.6668	0.6737	0.6797	0.6734	0.0106
DEMD - 0.5	0.6679	0.6646	0.6837	0.6728	0.6870	0.6752	0.0098
DEMD - 1.0	0.6403	0.6839	0.681	0.6838	0.6698	0.6718	0.0185
DMDLP	0.7047	–	–	–	–	**0.7047**	**0.0000**

Table A.12
Test accuracy obtained for susy.

Method - *vp*	*Ex. #1*	*Ex. #2*	*Ex. #3*	*Ex. #4*	*Ex. #5*	**Avg**	**Std-Dev**
DEMD - 0.25	0.7188	0.7160	0.7004	0.7121	0.7192	0.7133	0.0077
DEMD - 0.5	0.7322	0.7234	0.7086	0.7086	0.7147	0.7175	0.0102
DEMD - 1.0	0.7413	0.7445	0.7406	0.7354	0.7411	**0.7406**	**0.0033**
DMDLP	0.7393	–	–	–	–	0.7393	0.0000

Table A.13
Test AUC obtained for ECBDL14R.

Method - *vp*	*Ex. #1*	*Ex. #2*	*Ex. #3*	*Ex. #4*	*Ex. #5*	**Avg**	**Std-Dev**
DEMD - 0.25	0,5112	0,5118	0,5109	0,5103	0,5121	0,5113	0,0007
DEMD - 0.5	0,5125	0,513	0,5139	0,5122	0,5122	0,5128	0,0007
DEMD - 1.0	0,5142	0,515	0,5146	0,5143	0,5133	**0,5143**	0,0006
DMDLP	0.5131	–	–	–	–	0.5131	0.0000

Table A.14
Test AUC obtained for susy.

Method - *vp*	*Ex. #1*	*Ex. #2*	*Ex. #3*	*Ex. #4*	*Ex. #5*	**Avg**	**Std-Dev**
DEMD - 0.25	0,7072	0.7068	0.6892	0.7028	0.6968	0.7006	0.0076
DEMD - 0.5	0.7110	0.7065	0.6954	0.7032	0.7081	0.7048	0.0060
DEMD - 1.0	0.7152	0.7212	0.7128	0.7148	0.7144	**0.7157**	0.0032
DMDLP	0.7126	–	–	–	–	0.7126	0.0000

References

[1] S. García, J. Luengo, F. Herrera, Data Preprocessing in Data Mining, Springer, 2015.
[2] S. García, J. Luengo, F. Herrera, Tutorial on practical tips of the most influential data preprocessing algorithms in data mining, Knowl.-Based Syst. 98 (2016) 1–29.
[3] S. García, J. Luengo, J.A. Sáez, V. López, F. Herrera, A survey of discretization techniques: taxonomy and empirical analysis in supervised learning, IEEE Trans. Knowl. Data Eng. 25 (4) (2013) 734–750.
[4] S. Ramírez-Gallego, S. García, H. Mouriño Talín, D. Martínez-Rego, V. Bólon-Canedo, A. Alonso-Betanzos, J.M. Benítez, F. Herrera, Data discretization: taxonomy and big data challenge, Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. 6 (1) (2016) 5–21.
[5] C.C. Aggarwal, Data Mining: The Textbook, Springer, 2015.
[6] M. Minelli, M. Chambers, A. Dhiraj, Big Data, Big Analytics: Emerging Business Intelligence and Analytic Trends For Today's Businesses, John Wiley and Sons, 2013.
[7] V. Mayer-Schnberger, K. Cukier, Big Data: A Revolution That Will Transform How We Live, Work and Think, John Murray Publishers, 2013.
[8] S. García, S. Ramírez-Gallego, J. Luengo, J.M. Benítez, F. Herrera, Big data preprocessing: methods and prospects, Big Data Anal. 1 (1) (2016) 9.
[9] A. Fernández, S. del Río, V. López, A. Bawakid, M.J. del Jesús, J.M. Benítez, F. Herrera, Big data with cloud computing: an insight on the computing environment, mapreduce, and programming frameworks, Wiley Interdiscip. Rewiews: Data Min. Knowl. Discov. 4 (5) (2014) 380–409.
[10] N. Xiong, D. Molina, M.L. Ortiz, F. Herrera, A walk into metaheuristics for engineering optimization: principles, methods and recent trends, International, J. Comput. Intell. Syst. 8 (2015) 606–636.
[11] A. LaTorre, S. Muelas, J.-M. Peña, A comprehensive comparison of large scale global optimizers, Inf. Sci. 316 (2015) 517–549.
[12] S. Cheng, B. Liu, Y. Shi, Y. Jin, B. Li, Evolutionary computation and big data: Key challenges and future directions, in: Proceedings of the Data Mining and Big Data, First International Conference, DMBD 2016, Bali, Indonesia, June 25–30, 2016, pp. 3–14.
[13] A. Fernández, S. García, J. Luengo, E. Bernado-Mansilla, F. Herrera, Genetics-based machine learning for rule induction: state of the art, taxonomy, and comparative study, IEEE Trans. Evolut. Comput. 14 (6) (2010) 913–941.
[14] A. Fernández, V. López, M.J. del Jesús, F. Herrera, Revisiting evolutionary fuzzy systems: taxonomy, applications, new trends and challenges, Knowl. Based Syst. 80 (2015) 109–121.
[15] S.J. Nanda, G. Panda, A survey on nature inspired metaheuristic algorithms for partitional clustering, Swarm Evolut. Comput. 16 (2014) 1–18.
[16] A. Mukhopadhyay, U. Maulik, S. Bandyopadhyay, C.A.C. Coello, A survey of multiobjective evolutionary algorithms for data mining: Part I, IEEE Trans. Evolut. Comput. 18 (1) (2014) 4–19.
[17] S. Ramírez-Gallego, S. García, J.M. Benítez, F. Herrera, Multivariate discretization based on evolutionary cut points selection for classification, IEEE Trans. Cybern. 46 (3) (2016) 595–608.
[18] N. Sreeja, A. Sankar, A hierarchical heterogeneous ant colony optimization based approach for efficient action rule mining, Swarm Evolut. Comput. 29 (2016) 1–12.
[19] P. Mohapatra, S. Chakravarty, P. Dash, Microarray medical data classification using kernel ridge regression and modified cat swarm optimization based gene selection system, Swarm Evolut. Comput. 28 (2016) 144–160.
[20] W. Sheng, S. Chen, M. Sheng, G. Xiao, J. Mao, Y. Zheng, Adaptive multi-subpopulation competition and multiniche crowding-based memetic algorithm for automatic data clustering, IEEE Trans. Evolut. Comput. 20 (6) (2016) 838–858.
[21] S. Nebti, A. Boukerram, Swarm intelligence inspired classifiers for facial recognition, Swarm and Evolutionary Computation 23 (2017) 150–166. http://dx.doi.org/10.1016/j.swevo.2016.07.001 (In press).
[22] Apache Spark: Lightning-fast Cluster Computing, Apache spark, 2016. (Online; Accessed December 2016). ⟨https://spark.apache.org/⟩.
[23] M. Beyer, D. Laney, 3D Data Management: Controlling Data Volume, Velocity and Variety, 2001. ⟨http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf⟩.
[24] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation, vol. 6 of OSDI'04, 2004, pp. 10–10.
[25] T. White, Hadoop The Definitive Guide, O'Reilly Media, Inc., 2012.
[26] Apache Hadoop Project, Apache Hadoop, 2016. (Online; Accessed December 2016). ⟨http://hadoop.apache.org/⟩.
[27] J. Lin, Mapreduce is good enough? If all you have is a hammer, throw away everything that's not a nail!, Big Data 1 (1) (2013) 28–37.
[28] M. Hamstra, H. Karau, M. Zaharia, A. Konwinski, P. Wendell, Learning Spark: Lightning-Fast Big Data Analytics, O'Reilly Media, Incorporated, 2015.
[29] B.S. Chlebus, S.H. Nguyen, On finding optimal discretizations for two attributes, in: Proceedings of the First International Conference on Rough Sets and Current Trends in Computing, RSCTC '98, 1998, pp. 537–544.

[30] T. Elomaa, J. Rousu, General and efficient multisplitting of numerical attributes, Mach. Learn. 36 (1999) 201–244.

[31] L.J. Eshelman, The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination, in: FOGA, 1990, pp. 265–283.

[32] M. Dash, H. Liu, Consistency-based search in feature selection, Artif. Intell. 151 (1–2) (2003) 155–176.

[33] K.J. Cios, W. Pedrycz, R.W. Swiniarski, L.A. Kurgan, Data Mining: A Knowledge Discovery Approach, Springer, 2007.

[34] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc, 1993.

[35] S.J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 2nd ed., Pearson Education, 2003.

[36] S. Río, V. López, J. Benítez, F. Herrera, On the use of mapreduce for imbalanced big data using random forest, Inf. Sci. 285 (2014) 112–137.

[37] C.-C. Chang, C.-J. Lin, LIBSVM: a library for support vector machines, ACM Trans. Intell. Syst. Technol. 2 (2011) 27:1–27:27 (datasets available at ⟨http://www.csie.ntu.edu.tw/cjlin/libsvmtools/datasets/⟩).

[38] K. Bache, M. Lichman, UCI machine learning repository, 2013. ⟨http://archive.ics.uci.edu/ml⟩.

[39] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M.J. Franklin, R. Zadeh, M. Zaharia, A. Talwalkar, Mllib: machine learning in apache spark, J. Mach. Learn. Res. 17 (34) (2016) 1–7.

[40] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, F. Herrera, A survey on data preprocessing for data stream mining: current status and future directions, Neurocomputing 239 (2017) 39–57.

2 Data reduction for streaming data

The journal papers associated to this part are:

2.1 A survey on data preprocessing for data stream mining: Current status and future directions

- S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak and F. Herrera, A survey on data preprocessing for data stream mining: Current status and future directions. Neurocomputing 239 (2017) 39–57, doi: 10.1016/j.neucom.2017.01.078.
  - Status: **Published**.
  - Impact Factor (JCR 2016): 3.317.
  - Subject Category: Computer Science, Artificial Intelligence. Ranking 24 / 133 (**Q1**).

Neurocomputing 239 (2017) 39–57

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

A survey on data preprocessing for data stream mining: Current status and future directions

Sergio Ramírez-Gallego[a,*], Bartosz Krawczyk[b], Salvador García[a], Michał Woźniak[c], Francisco Herrera[a,d]

[a] Department of Computer Science and Artificial Intelligence, CITIC-UGR, University of Granada, Granada 18071, Spain
[b] Department of Computer Science, Virginia Commonwealth University, Richmond, VA 23284, USA
[c] Department of Systems and Computer Networks, Wrocław University of Science and Technology, Wyb. Wyspiańskiego 27, Wrocław 50-370, Poland
[d] Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia

ARTICLE INFO

Article history:
Received 28 December 2016
Revised 31 January 2017
Accepted 31 January 2017
Available online 14 February 2017

Communicated by Zidong Wang

Keywords:
Data mining
Data stream
Concept drift
Data preprocessing
Data reduction
Feature selection
Instance selection
Data discretization
Online learning

ABSTRACT

Data preprocessing and reduction have become essential techniques in current knowledge discovery scenarios, dominated by increasingly large datasets. These methods aim at reducing the complexity inherent to real-world datasets, so that they can be easily processed by current data mining solutions. Advantages of such approaches include, among others, a faster and more precise learning process, and more understandable structure of raw data. However, in the context of data preprocessing techniques for data streams have a long road ahead of them, despite online learning is growing in importance thanks to the development of Internet and technologies for massive data collection. Throughout this survey, we summarize, categorize and analyze those contributions on data preprocessing that cope with streaming data. This work also takes into account the existing relationships between the different families of methods (feature and instance selection, and discretization). To enrich our study, we conduct thorough experiments using the most relevant contributions and present an analysis of their predictive performance, reduction rates, computational time, and memory usage. Finally, we offer general advices about existing data stream preprocessing algorithms, as well as discuss emerging future challenges to be faced in the domain of data stream preprocessing.

1. Introduction

Data preprocessing [1,2] is one of the major phases within the knowledge discovery process. Despite being less known than other steps like data mining, data preprocessing actually very often involves more effort and time within the entire data analysis process ($>$ 50% of total effort) [3]. Raw data usually comes with many imperfections such as inconsistencies, missing values, noise and/or redundancies. Performance of subsequent learning algorithms will thus be undermined if they are presented with low-quality data. Thus by conducting proper preprocessing steps we are able to significantly influence the quality and reliability of subsequent automatic discoveries and decisions.

Data preparation, as part of preprocessing [1], is aimed at transforming raw input into high-quality one that properly fits the mining process to follow. Preparation is considered as a mandatory step and it includes techniques such as integration, normalization, cleaning and transformation.

Presently, the amount generated data is growing exponentially following the emergence of Big Data phenomenon [4,5]. Contemporary datasets grow in three dimensions –features, examples and cardinality– making complexity reduction a mandatory step if standard algorithms are to be used. Data reduction techniques perform this simplification by selecting and deleting redundant and noisy features and/or instances, or by discretizing complex continuous feature spaces. This allows to maintain the original structure and meaning of the input, but at the same time obtaining a much more manageable size. Faster training and improved generalization capabilities of learning algorithms, as well as better understandability and interpretability of results, are among the many benefits of data reduction.

With the advent of Big Data comes not only an increase in the volume of data, but also the notion of its velocity. In many emerging real-world problems we cannot assume that we will deal with a static set of instances. Instead, they may arrive continuously,

* Corresponding author.
E-mail addresses: sramirez@decsai.ugr.es (S. Ramírez-Gallego), bkrawczyk@vcu.edu (B. Krawczyk), salvagl@decsai.ugr.es (S. García), michal.wozniak@pwr.edu.pl (M. Woźniak), herrera@decsai.ugr.es (F. Herrera).

http://dx.doi.org/10.1016/j.neucom.2017.01.078
0925-2312/© 2017 Elsevier B.V. All rights reserved.

leading to a potentially unbounded and ever-growing dataset. It will expand itself over time and new instances will arrive continuously in batches or one by one. Such problems are known as data streams [6] and pose many new challenges to data mining methods. One must be able to constantly update the learning algorithm with new data, to work within time-constraints connected with the speed of arrival of instances, and to deal with memory limitations. Additionally, data streams may be non-stationary, leading to occurrences of the phenomenon called *concept drift*, where the statistical characteristics of the incoming data may change over the time. Thus, learning algorithms should take this into consideration and have adaptation skills that allow for online learning from new instances, but also for quick changes of underlying decision mechanisms [7].

Despite the importance of data reduction, not many proposals in this domain may be found in the literature for online learning from data streams [8]. Most of methods are just incremental algorithms, originally designed to manage finite datasets. Direct adaptation of static reduction techniques is not straightforward since most of techniques assume the whole training set is available from the beginning and properties of data do not change over time:

- Most of static instance selectors require multiple passes over data, at the same time being mainly based on time-consuming neighbor searches that makes them useless for handling high-speed data streams [1].
- On the contrary, feature selection techniques are easily adaptable to online scenarios. Yet, they suffer from other problems such as concept evolution or dynamic [9] and drifting [10] feature space.
- Online supervised discretization methods also remain fairly unexplored. Most of standard solutions require several iterations of sharp adjustments before getting a fully operating solution [11].

Therefore, further development of data pre-processing techniques for data stream environments is thus a major concern for practitioners and scientists in data mining areas.

This survey aims at a thorough enumeration, classification, and analysis of existing contributions for data stream preprocessing. Although there exist previous studies that have performed a coarse-grained analysis on some tasks individually (e.g., feature selection or instance selection) [12,13], this work is a first deep overview of advances in this filed, additionally outlining vital future challenges that need to be addressed to ensure meaningful progress and development of novel methods.

In addition to discussing the literature in preprocessing methods for mining data streams, we propose a thorough experimental study to further enrich this survey. We have analyzed predictive, reduction, time and memory performance of selected most relevant algorithms in this field. Additionally, nonparametric statistical tests are used to give support to the final conclusions. The discussed experimental framework involves a total of 20 datasets and 10 reduction methods: three feature selectors, three discretizers, and four instance selectors.

The structure of this work is as follows. First, we present related concepts such as: data streaming and concept drift (Section 2), and data reduction (Section 3). Then online reduction contributions are grouped by task, and described in Section 4. To assess performance and usefulness of methods, a thorough experimental framework is proposed in Section 5, also grouped by task. Section 6 summarizes the lessons learned from this survey and experimental study, and discusses open challenges in data preprocessing for data stream mining, while Section 7 concludes this work.

2. Data streams and concept drift

Data stream is a potentially unbounded and ordered sequence of instances that arrive over time [14]. Therefore, it imposes specific constraints on the learning system that cannot be fulfilled by canonical algorithms from this domain. Let us list the main differences between static and streaming scenarios:

- instances are not given beforehand, but become available sequentially (one by one) or in the form of data chunks (block by block) as the stream progresses;
- instances may arrive rapidly and with various time intervals between each other;
- streams are of potentially infinite size, thus it is impossible to store all of incoming data in the memory;
- each instance may be only accessed a limited number of times (in specific cases only once) and then discarded to limit the memory and storage space usage;
- instances must be processed within a limited amount of time to offer real-time responsiveness and avoid data queuing;
- access to true class labels is limited due to high cost of label query for each incoming instance;
- access to the true labels may be delayed as well, in many cases they are available after a long period, i.e., for credit approval could be 2–3 years;
- statistical characteristics of instances arriving from the stream may be subject to changes over time.

Let us assume that our stream consists of a set of states $\mathbf{S} = \{S_1, S_2, \ldots, S_n\}$, where S_i is generated by a distribution D_i. By a stationary data stream we will consider a sequence of instances characterized by a transition $S_j \rightarrow S_{j+1}$, where $D_j = D_{j+1}$. However, in most modern real-life problems the nature of data may evolve over time due to various conditions. This phenomenon is known as concept drift [7,15] and may be defined as changes in distributions and definitions of learned concepts over time. Presence of drift can affect the underlying properties of classes that the learning system aims to discover, thus reducing the relevance of used classifier as the change progresses. At some point the deterioration of the quality of used model may be too significant to further consider it as a meaningful component. Therefore, methods for handling drifts in data streams are of crucial importance to this area of research.

Let us now present shortly a taxonomy of concept drift. There are two main aspects that must be taken under consideration when analyzing the nature of changes taking place in the current state of any data stream:

- **Influence on the learned classification boundaries** - here we distinguish two types of concept drift. A **real** concept drift affects the decision boundaries (posterior probabilities) and may impact unconditional probability density function, thus poses a threat to the learning system. A **virtual** concept drift does not impact the decision boundaries (posterior probabilities), but affect the conditional probability density functions, thus not influencing the currently used learning models. However, it should still be detected. Visualization of these drift types is presented in Fig. 1.
- **Types of change** - here we may distinguish three main types of concept drift taking into consideration its rapidness. **Sudden** concept drift is characterized by S_j being rapidly replaced by S_{j+1}, where $D_j \neq D_{j+1}$. **Gradual** concept drift can be considered as a transition phase where examples in S_{j+1} are generated by a mixture of D_j and D_{j+1} with their varying proportions. **Incremental** concept drift has a much slower ratio of changes, where the difference between D_j and D_{j+1} is not so significant, usually not statistically significant.
- We may also face with so-called **Recurring** concept drift, what means that a concept from kth previous iteration may

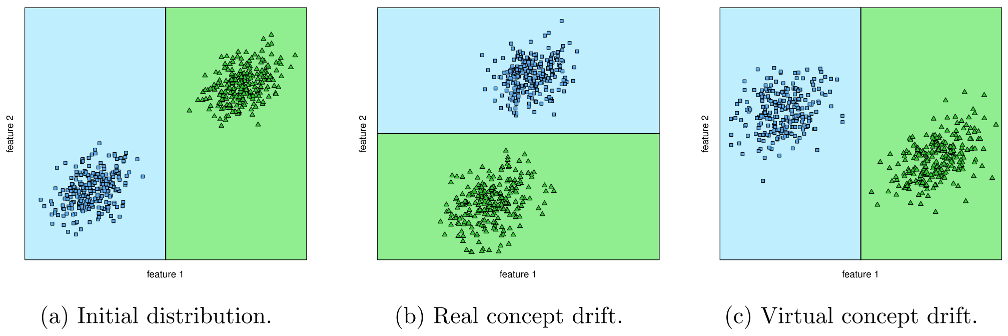


Fig. 1. Two main types of concept drift with respect to their influence over decision boundaries.

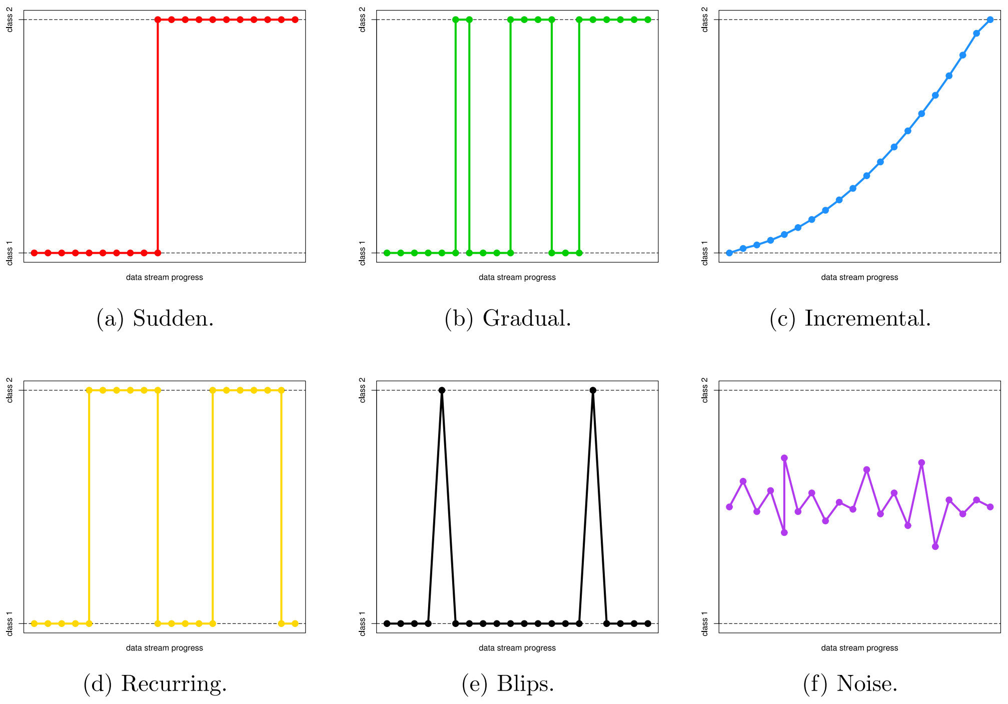


Fig. 2. Six types of drifts with respect to the ratio of changes. Graphs show transitions between the concepts along during the data stream progress.

reappear $D_{j+1} = D_{j-k}$ and it may happen once or periodically. **Blips**, also known as outliers which should be ignored as the change it represents is random [16]. **Noise**, which represents insignificant fluctuations of the concept and should be filtered out [17]. **Mixed** concept drift is a hybrid phenomenon, where more than a single type of concept drift may appear during the stream mining process. One should note that in real-life scenarios types of changes to appear are unknown beforehand and must be determined during the stream processing. Visualization of these types of drifts are presented in Fig. 2.

- Minku et al. [18] proposed *severity* criterion which allows to distinguish between **local** and **global** drift. The local drifts mean that changes affects only the small region of the feature space, while global drift affects the overall feature space, what cause that it is easier detected than the local one [19]. Additionally, we may also face with so-called ”feature drift” [10], where the changes affect only selected attributes.
- Unfortunately, in real classification tasks concept drift may appear as a mixture of mentioned above changes.

As mentioned before, managing concept drift is a crucial issue in learning from data streams. Here we may use on of three solutions: (a) retrain classification system from scratch every time a new instance or chunk becomes available; (b) detecting changes and retraining classifier only when the degree of changes has been considered as significant enough; and (c) using adaptive learning method that can follow the shifts and drifts in stream on its own. Obviously, the first approach is characterized by an unacceptable computational cost and therefore two remaining solutions are used in this field.

Let us now discuss four main approaches to efficiently tackling drifting data streams:

- **Concept drift detectors** are external tools used together with the classification module. They measure various properties of data stream, such as standard deviation [20], predictive error [21], instance distribution [22], or stability [23]. Any changes in these properties are attributed to the potential presence of drift and thus allow to monitor the continuous progress of data stream. Most of drift detectors work in a two-stage setting. A warning signal is emitted when the changes start to occur, being a single to the learning system that a new classifier should be trained on the most recent instances. A detection signal informs the learning system that current degree of changes is severe and the old classifier should be replaced by a new one. This solution is also known as explicit drift handling. One should notice that ensembles of detectors start to attract the attention of research community, although there is still much work needed to be done in this area [24,25].

- **Sliding windows** assume that we keep a buffer of fixed size containing most recent examples [26]. They are used for the classification purposes and then discarded when new instances become available. This allows us to keep a track on the progress of data stream by storing its current state in the memory [27]. This is realized either by cutting-off oldest instances or weighting them dynamically according to their relevance [28]. However, the size of the window has a crucial impact on its performance. A small window will be able to adjust to small and rapid changes, but may lose the general context of the analyzed problem and be prone to overfitting. A large window can efficiently store more information, but may contain instances originating from different concepts. To solve this issue recent studies focus on dynamically adapting size [29] or using multiple windows at the same time [30]. One should notice that a properly set sliding window will be able to adjust to changes in the stream. This is known as implicit drift handling.
- **Online learners** are updated instance by instance, thus accommodating changes in stream as soon as they occur. Such models must fulfill a set of requirements [31]: each object must be processed only once in the course of training, computational complexity of handling each instance must be as small as possible, and its accuracy should not be lower than that of a classifier trained on batch data collected up to the given time. One must notice that a set of standard classification algorithms may work in online mode, e.g., Neural Networks [32] or Naïve Bayes. However, there exist a plethora of methods modified to provide efficient online mode of operation [33,34]. These methods also offer implicit drift handling.
- **Ensemble learners** are a popular family of methods for data stream mining [35,36]. Due to their compound structure they can easily accommodate changes in the stream, offering gains in both flexibility and predictive power. Two main approaches here assume a changing line-up of the ensemble [37–39] or updating base classifiers [40,41]. In the former solution a new classifier is being trained on recently arrived data (usually collected in a form of chunk) and added to the ensemble. Pruning is used to control the size of the committee and remove irrelevant or oldest models. A weighting scheme allows to assign highest importance to newest ensemble components, although more sophisticated solutions allow to increase weights of classifiers that are recently best-performing. Here one can use static classifier, as the dynamic line-up keeps a track of stream progress. Latter solutions assume that a fixed-size ensemble is kept, but update each component when new data become available. Here managing the diversity of the ensemble is crucial for achieving good predictive power [42]. Additionally, ensembles must consist of classifiers working in incremental or online modes. There also exist hybrid approaches that combine both of these solutions within the ensemble structure [43,44].

Proper experiment design and evaluation of the examined algorithms is a key issue in machine learning domain. One need an unbiased, fair and repeatable way of comparing tested algorithms that will allow to shed lights on their strength and weaknesses, at the same time leading to valuable conclusions towards better understanding of used methods. We may evaluate certain method to assess some of our hypothesis about it, or to check its usability for a particular real-life application. Before starting any computations one must reasonably state goals of the experiment to be undertaken, choose relevant datasets, select proper metrics that will reflect the nature of examined data and establish a correct procedure for learning and comparing different models. This issue has been well-discussed in static scenarios and there exist a number of generally accepted procedures to be undertaken [45]. In the context of data stream mining, especially in non-stationary environments, canonical metrics and procedures become no longer applicable. We deal with massive, continuously incoming and evolving data that requires updating the learning model and adjusting to shifts and drifts. New classes may appear, feature space change and decision rules loose relevance over time. Additionally, canonical metrics for measuring the quality of learning process are not sufficient to perform a meaningful evaluation of models [46]. Let us discuss the correct metrics to be used for algorithms applied to data stream mining. One must understand that good algorithm must aim to strike a balance among all of these criteria.

- **Predictive power** is an obvious criterion measured in all learning systems. However, in data stream mining we must accommodate the fact that the relevance of instances diminishes over time. Therefore, simply using any averaged measure does not reflect how the learning system was able to adapt and react to changes in the stream and constant increase in the number of processed instances. Therefore, one needs to use prequential metrics that are calculated only over the most recent examples with a forgetting mechanism embedded. Prequential accuracy [47] and prequential area under the Receiver Operating Characteristics curve (AUC) [48] are the two most widely used ones.
- **Memory consumption** is a necessary criterion due to the hardware limitations during processing potentially unbounded data stream [49]. Not only the average memory usage should be taken under consideration, but also how it changes over time and with specific actions made by each algorithm.
- **Recovery time** informs us how much time an algorithm needs to accommodate new instances and update its structure. This is a crucial measure that can be a bottleneck of many methods. Assuming that new instances arrive rapidly, a good stream mining algorithm should be able to process instances before new ones will arrive to avoid queuing [50].
- **Decision time** is another time-complexity measure used. Here we are interested how long certain algorithms need to make a prediction for each new instance. As recognition phase usually precedes the update phase, it may be another bottleneck for the system. Additionally, in many applications we require a real-time response and cannot allow for a delay when speed is decision speed is vital [51].
- **Requirement for true class labels** can strongly limit the real-life applicability of many data stream mining algorithms. Many works on supervised learning in streaming scenarios assume that class labels become available soon after the instance was being classified by the system, or arrive with some delay. However, the costs of labeling the entire data stream are far from realistic and thus we must deal with limited availability of true class labels. It is useful to examine the influence of available budget (number of labeled samples) on the effectiveness of algorithms. Active learning strategies allow to select only the most relevant samples for labeling [52,53]. Semi-supervised and unsupervised methods for both classification [54,55] and drift detection [56,57] are also of interest in order to cope with this issue.

3. Data reduction

Data reduction [2] is an important preprocessing step in data mining, as we aim at obtaining accurate, fast and adaptable model that at the same time is characterized by low computational complexity in order to quickly respond to incoming objects and changes. Therefore, dynamically reducing the complexity of the incoming data is crucial to obtain such models. Additionally, due to the presence of concept drift the number and relevance of instances and features may change over time. This must also be taken into consideration while maintaining and updating an online model. Let us now discuss the main areas in data preprocessing for reducing the complexity of data.

- **Dimensionality reduction**: There exist a wide range of techniques in the literature that aim at reducing the number of features, among others: Feature Selection (FS), Feature Extraction (FE) or locality preserving projection [58–60]. In this paper, we focus on FS and FE techniques. FS [61] eliminates irrelevant or redundant features/columns, whereas Feature Extraction (FE) generates a simpler feature space through transformations of the original one. The aim here is to yield a minimum set of features so that the subsequent distribution probability of classes remains as unchanged as possible. As FS maintains the original features, it is more convenient for model interpretation. Depending on the relationship between the selector and the predictive algorithms, we can classify FS algorithms into three categories: filters, which act before the learning process, being independent from it; wrappers, which use the specified learning algorithm to evaluate subgroups of features; and embedded, where the search is a part of the learning process itself. Wrappers methods tend to be more accurate than filters, but more complex. Embedded methods are less costly than wrappers, but require direct modifications of the learning procedure.
- **Instance reduction**: Instance Selection (IS) or Instance Generation (IG) [62]. IS is aimed at reducing the number of training instances by selecting the most representative examples. IG methods can generate new instances to fill the gaps in concept definitions. IS differs from data sampling in that the former categorizes instances depending on the problem, whereas sampling is more stochastic. Based upon the kind of search implemented by the IS algorithms, they can be classified into three categories: condensation, which removes redundant points far from the borders; edition, which removes noisy points close to the class boundaries; or hybrid, which combines both noise and redundancy removal.
- **Feature space simplification**: Normalization, Discretization, and etc. Discretization [63] summarizes a set of continuous values into a finite set of discrete intervals. This process returns nominal features that can be used by any mining process. Although most of mining algorithms work with continuous data, many of them can only cope with nominal features, specially those based on statistical and information measures (e.g.: Naïve Bayes (NB)) [64]. Other algorithms, like tree-based classifiers [65], generate more accurate and compact results when using discrete values. Good discretizers try to achieve the best predictive performance derived from discrete data, while reducing the number intervals as much as possible [66,67]. We can distinguish two main categories, based upon how intervals are generated by discretizers: splitting methods, which split the most promising interval in each iteration into two partitions; and merging methods, which merge the best two adjacent intervals in each iteration.

4. Data reduction on data streams

In streaming scenarios reduction techniques are demanded to preferably process elements online or in batch-mode as quick as possible and without making any assumptions about data distribution in advance. In the next sections, we describe those reduction proposals that were tailored for mining data streams. These methods are grouped by family/task: dimensionality reduction (Section 4.1), instance reduction (Section 4.2), and feature space simplification (Section 4.3).

4.1. Dimensionality reduction

Many FS algorithms for data streams have been proposed in the literature. Most of them are naturally incremental algorithms designed for offline processing [1], whereas others are specifically thought to cope with flowing streams [12]. All FS methods can be divided into three groups: filters, wrappers, and hybrid; according to when selection is performed: before and independently to the learning step, or tightly coupled with it.

Most of online selectors proposed in the literature are incremental adaptations of offline filters. As these filters rely on cumulative functions (mainly based on information or statistical measures), these are easily adaptable to the online environment. Despite being simple, online filters seems to adapt well to drifts, and do not need to ingest all data at once like their offline counterparts. Furthermore, online methods usually face problems derived from streams that cannot be addressed by offline methods, like the arrival of new features or classes.

Focusing on online FS, further distinctions can be made depending on the properties of streams. Some FS methods suppose that features arrive one-by-one (*streaming features*) while feature vectors are initially available [68,69]; whereas others assume that the instances always arrive sequentially, and the feature set may be subject to potential changes [70] (*online FS*). New classes can also emerge from streams without previous knowledge (concept evolution), requiring a complete redefinition of the used model. In data stream mining, feature space can also be affected by changes in data distribution. *Feature drifts* occur whenever the relevance of a given attribute changes over time when new instances arrive to the system [71]. As in other concept drifts, changes in relevance enforce algorithms to discard or adapt the model already learned by removing the most irrelevant features in the new scenario [72], as well as including the most relevant ones (*dynamic FS*). As changes in relevance directly affect the decision boundaries, feature drift can be seen as a specific type of real concept drift.

As the set of selected features evolves over time, it is likely that the feature space in test instances differs from the current selection. Therefore, when a new instance is being classified, we need to perform a conversion between feature spaces for homogenization purposes [9]. The types of conversion to consider are the following:

- Lossy Fixed (Lossy-F): the same feature set is used for the whole stream. It is generated from the first batch. All the following instances (training and test) will be mapped to this set, resulting in a clear loss in future information.
- Lossy Local (Lossy-L): a different feature space is used for each new training batch. Test instances are thus mapped to the training space in each iteration. This conversion is also troublesome because relevant features in test may be omitted.
- Lossless Homogenizing (Lossless): Lossless is similar to the previous conversion, except that the feature space in the test set is being considered here. There exist a homogenization between spaces, for example, by unifying both spaces and padding with zeros any missing feature in the other set. This conversion results in using all current and previous information, so it can be seen as the best option.

In this paper, we will focus on **online** techniques that allow the arrival of new instances and features at the same time, because they represent a scenario present in real-world problems. Let us now present a list formed by the most relevant algorithms on this topic:

- Katakis et al. [70] was among the first to introduce the problem of dynamic feature space in data streams. They proposed a technique that includes a feature ranking (filter) method to select relevant features. As the importance score of each feature can be measured using many cumulative functions like Information Gain (IG), χ^2 or mutual information, it can be seen as a versatile solution for online feature ranking.

- Carvalho et al. [73] proposed Extremal Feature Selection (EFS), an online FS method that uses the weights computed by an online classifier (Modified Balanced Winnow) to measure the relevance of features. The score is computed as the absolute difference between the positive and negative weights for each feature.
- Masud et al. [9] proposed a streaming classification technique (DXMiner), which uses the deviation weight measure to rank features during the classification phase. Furthermore, DXMiner naturally address the problem of novel classes (concept-evolution) by building a decision boundary around the training data. In contrast to previous methods, DXMiner uses lossless conversion, which is useful for novelty detection. To rank features in the test space, DXMiner uses a unsupervised technique (e.g., the highest frequency in the batch) that selects features more representative for incoming concepts. Note that this requires a batch-mode setting to compute such statistics.
- Nguyen et al. [72] designed an ensemble technique based on windowing to detect feature drifts. The algorithm is based on a ensemble of classifiers, where each classifier has its own feature set. If a drift is detected, the ensemble is updated with a new classifier together with a new feature subset; otherwise, each classifier is updated accordingly. Fast Correlation-Based Filter (FCBF) based on Symmetrical Uncertainty is being used here. FCBF heuristically applies a backward technique with a sequential search strategy to remove irrelevant and redundant features.
- In [74], authors propose an algorithm to mine recurring concepts (called MReC-DFS). Here, they adopt the same selection solution proposed in [70]. Hover, instead of selecting a fixed number of features, they propose to use either a fixed threshold or an adaptive one based on percentiles. They also compare the effects of using different space conversions [9] (like Lossy-F, Lossy-L or Lossless).
- Wu et al. [75] proposed two approaches for handling streams with growth of feature volumes over time, named Online Streaming Feature Selection (OSFS) and Fast Online Streaming Feature Selection (Fast-OSFS). They are based on a two-phase optimal subset discovery scheme: online analysis of relevance and then redundancy. Class-based relevance is used to select or discard a new feature. Then a new and extended feature set is analyzed to detect if there exist a subset of features that may make one of the used features and class variable conditionally independent. If yes, then such a feature is discarded. This allows to to control the expansion of the feature space. In Fast-OSFS the redundancy analysis is divided into two parts. Firstly a redundancy of new feature is being checked, in order to decide if this feature should be selected. Only if new feature was included, the redundancy of previous features is being analyzed. This leads to a significant computational speed-up of this method.
- Wang et al. [76,77] proposed a greedy online FS method (called OFS) based on a classical technique that makes a trade-off between exploration and exploitation of features. The algorithm spends ε iterations on exploration by randomly choosing N attributes from the whole set of attributes, and the remaining steps on exploitation by choosing the N attributes for which the linear classifier has nonzero values. In this work, no feature drift is addressed explicitly, and no comparison with previous works is performed.
- An online feature selection method based on group structure analysis was proposed in [78]. This work was based on assumption that features may arrive in specific groups, like textures, colors etc. Authors proposed Online Group Feature Selection (OFGS) algorithm that utilized intra-group and inter-group criteria. The former criterion used spectral analysis to select discriminative features in each group. The latter one applied linear regression model to chose an optimal subset of from all preselected features. It is worth noticing that a similar problem was discussed by Li et al. [79].

Table 1 details the type of selection and space conversion performed by each algorithm. Two remarkable selection strategies emerges from this summary: one based on information filtering and another based on the use of classifier weights (wrapper).

Apart from the previously mentioned most relevant algorithms there exist a number of other online and streaming feature selection proposals in the literature. Let us now discuss them shortly. Yan et al. [80] proposed simultaneous feature extraction and selection using orthogonal centroid algorithm. Tadeuchi et al. [81] proposed a quick online feature selection that used filters to generate several potential subsets and a wrapper to chose the best one from them. Authors speculated that this solution should be able to handle concept drift appearance. Cai et al. [82] proposed to use $l1$-norm regularization for continuous variable selection. Similar approach was used by Ooi and Ninomiya, however they had employed a regularized regression for this task [83]. Fan and Bouguila [84,85] presented a combination of clustering based on a Dirichlet process mixture of generalized Dirichlet distributions and unsupervised feature selection in incremental learning scenarios. Amayri and Bouguila [86] discussed similar combination of group discovery and feature reduction using finite mixtures of von Mises distributions, while Yao and Liu [87] combined online selection with density estimation. A problem of online feature selection for multi-task learning was discussed in [88]. The issue of scalability of the discussed family of models for big data mining was addressed in [89]. Roy [90] discussed how to use ensemble of Kohonen neurons for choosing features from high-dimensional streams. Recently, Yang et al. [91] introduced a parallel method using limited memory, while Hammoodi et al. [92] discussed a concept drift detection approach using only selected features. Extension of OSFS

Table 1
Summary description of streaming FS methods. Information about the type of selector (wrapper or filter), the feature conversion accomplished (if appropriate), and whether concept-evolution appears, is presented below.

Method	Selection type (measure)	Streaming features (conversion)	Concept-evolution
Katakis' method [70]	Filter (IG, χ^2, etc.)	no (Lossy-F)	no
EFS [73]	Wrapper (online classifier's weights)	no (Lossy-F)	no
DXMiner [9]	Filter (deviation weight) + unsupervised	yes (Lossless)	yes
HEFT-Stream [72]	Filter (SU)	no (Lossy-F)	no
MReC-DFS [74]	Filter (IG, χ^2, etc.)	yes (all)	no
OSFS / Fast-OSF [75]	Filter (relevance and redundancy)	no (Lossy-F)	no
OFS [77]	Wrapper (online classifier's weights)	no (Lossy-F)	no
OFGS [78]	Filter (spectral clustering and regression)	no (Lossy-F)	no

method using rough set approach for data streams was analyzed in [93], while a combination of online discretization with feature selection for neural networks was depicted in [94].

One may view a video sequence as a stream of images and in this domain online feature selection has also been explored in order to handle dynamic object detection. Yeh et al. [95] introduced an online Boosting-based feature selection, where new features were selected one at a time to compensate for changes in the background. Yang et al. [96] described an online Fisher discrimination boosting feature selection mechanism for real-time visual tracking.

Finally, it is worthwhile to mention work by Yu et al. [97], where authors implemented several popular online feature selection methods and created an open software package for Matlab.

Besides FS, dimensionality reduction can be accomplished through an artificial mapping between the original space of features and a new space of fewer dimensions. Feature extraction techniques, although less popular than FS ones, have shown their ability in many predictive problems. One of the most important contributions here is Principal Component Analysis (PCA) [98]. In [99], two online gradient-based versions of PCA are studied in depth. The aim of previous work is to obtain an online model with the lowest difference in cumulative losses with respect to the best offline alternative. A novel analysis of theorethical properties of Oja's streaming PCA was discussed in [100]. Although optimal, online PCA is not able to update projections in less than $O(n^3)$ per iteration [101]. Thus more efficient techniques needs to be developed in the future if we want a real streaming solution in feature extraction. So far it is worth mentioning streaming versions of kernel PCA proposed by Joseph et al. [102] and by Ghashami et al. [103]. Additionally, PCA was successfully applied for concept drift detection in non-stationary data streams by Kuncheva and Faithfull [104], as well as by Qahtan et al. [105]. One must notice that feature extraction from data streams is not only limited to PCA and other works, although few in numbers, exist. Allahyar and Yazdi [106] described Online Discriminative Component Analysis for continuous computation of Linear Discriminant Analysis. Sheikholeslami et al. [107] proposed a kernel-based feature extraction for mining streams with limited computational resources. Li et al. [108] introduced canonical correlation analysis with uncertainty suitable for multi-view classification of data streams.

4.2. Instance reduction

Lazy learning has been broadly used in predictive analytics [109]. Yet, case-bases naturally deteriorate and grow in size over time. In data stream scenario, past preserved cases that belong to a previous concept may degrade the performance of the learner if a new concept appears. Likewise, new instances that represent a new concept may be classified as noise and removed by a misbehavior of the IS mechanism, because they disagree with past concepts [13].

Some enhancement (**edition**) and maintenance (**condensation**) [1] should be thus performed on case-bases in form of sophisticated IS processes, which select those cases that best represent the current state of the data stream. However, most of current techniques are designed for stationary environments and ignore the concept drift phenomenon. Firstly, we present a subset of IS techniques that incrementally or in a batch way select instances from a case-base [110]:

- Instance-Based learning Algorithm 3 (IB3) [111] is one of the first attempts to deal with non-stationary nature of data. It is based on accuracy and retrieval frequency measures. By means of a confidence interval test, IB3 decides whether a case should be added to the case-base or it needs to wait until its insertion is marked as appropriate. Removal of cases is performed whenever the accuracy of a case is below (in a certain degree) its class frequency. Due to IB3 defers the inclusion of examples, it is only suitable for gradual concept drift.
- The Locally Weighted Forgetting (LWF) algorithm [112] is an instance weighting technique based on k-nearest neighbors (kNN). In LWF, those cases with a weight below a threshold are removed. LWF algorithm has been criticized by its lower asymptotic classification in static environments and by its tendency to overfitting [113]. This method has shown good performance for both gradual and sudden concept drifts.
- Salganicoff [114] designed the Prediction Error Context Switching (PECS) algorithm, which is designed to work in both dynamic and static environments. PECS algorithm is based on the same measures used by IB3, also adopting the same confidence test. In order to introduce time dimension in its decisions, PECS only consider the newest predictions in its computations. Furthermore PECS immediately add new cases to the base to expedite the slow adaptation process. PECS disables cases instead of permanently deleting them. Those cases can be re-introduced if their may once again contribute towards improved accuracy. It is argued in [115] that PECS holds high memory requirements and a slow removal process, as new instances are retained right after they arrive.
- Iterative Case Filtering Algorithm (ICF) [116] is a redundancy removal technique that discards those instances with a coverage set size smaller than its reachability set. Authors included Repeated Edited-NN [117] to remove the noise around the borders.

Although there exist more complex proposals in the literature [110], the previous list includes those methods that have served as a keystone for further developments in IS for concept drift [13]. The next list deal with those techniques that explicitly address concept drift:

- Delany et al. [118] proposed a drift control mechanism with two levels, called Competence-Based Editing (CBE). In the first level, an hybrid of two competence-based editing methods[1]: Blame Based Noise Removal (BBNR) and Conservative Redundancy Reduction (CRR), is launched. BBNR is aimed at deleting those cases whose removal do not imply coverage loss, whereas CRR selects misclassified cases with the smallest coverage. Note that both methods are designed for stationary environments, which can cause some problems like the removal of novel concepts when gradual drift appears, or forgetting of small groups of cases where examples covers each other but misclassifies all the surrounding neighbors. BBNR do not keep the competence model up-to-date, it only rebuild the model in the second level. An outdated competence model may yield inconsistencies during the evaluation phase as the model does not accurately reflect the current concept.
- Instance-Based Learning on Data Streams (IBL-DS) [115], and IBLStreams [120] are presented as the first solutions that deem both time and space factors to control the shape and size of the case-base. In both algorithms, every neighbor in a test range is removed if the class of new instance is dominant in this range. IBL-DS also introduces an explicit drift detection method developed by Gama [20], which determines when to remove a fixed number of instances considering space and time. Number of removals is computed considering the minimum error rate and the aggregated error of last predictions. Both algorithms control the size of the case-base by removing the oldest instances. However the time-based removal strategy

[1] Basic concepts about competence models can be reviewed in [119]

Table 2
Summary description of streaming IS methods. Information about the selection measure, whether drift detection is used or not, and the type of selection is shown here.

Method	Selection type	Drift detection	Edition/Condensation
IB3 [111]	Case accuracy	no	yes/no
LWF [112]	Instance weighting	no	yes/no
PECS [114]	Case accuracy	no	yes/no
ICF [116]	Competence	no	yes/yes
CBE [118]	Competence	no	yes/yes
IBL-DS [115]	Time-space distance	yes	yes/yes
FISH [121]	Time-space distance	no	yes/yes
AES [122]	Bio-inspired	yes	no/yes
COMPOSE [123]	Geometry	no	no/yes
SimC [124]	Time-space distance/case accuracy	no	yes/yes
NEFCS-SRR [13]	Competence & case accuracy	yes	yes/yes

implemented by them has been criticized because some old, yet still relevant instances may be eliminated in this process.

- FISH algorithms [121] are also based on a combination of time and space, in this case, computed as distances. The idea behind these algorithms is to dynamically select the most relevant examples, which will serve as training for next model. Three different versions of FISH were proposed. In FISH1, the training size is fixed at the start. FISH2 selects the best training size according to the accuracy (through leave-one-out cross validation). FISH3 also weights time and space by using a different loop of cross validation. FISH2 is considered as the leader of the family. FISH represents a time-consuming option since it stores all seen examples in order to compute space/time distances.
- Zhao et al. [122] present a new nearest neighbor algorithm for data streaming, based on an artificial endocrine system; called AES. This system removes the necessity of a complete case-base as in previous models, replacing case-base by representative cells. A condensation-based process is also a key feature in AES. The algorithm maintains only K boundary prototypes or cells. These prototypes keep moving during the whole process in order to adapt concept boundaries to incoming drifts.
- COMPOSE [123] is a geometry-based framework for semi-supervised learning and active learning. The idea behind COMPOSE is to label incoming instances through a semi-supervised approach, and then to create and select those α-shapes that better model the current state. This selection is, in fact, a compaction process that maintains only those shapes/prototypes more representatives for the current state. COMPOSE is mainly designed to address gradual drifts.
- SimC [124] aims at creating groups of instances for each class so that each one represents a different region of the space. Noisy and old examples are removed by selecting and discarding the least relevant example in the oldest group. As concept drift appears, the algorithm creates new groups to allocate examples that represents new concepts. Relevance in groups is measured by using space distances and their ages. For individual instances, the precision using the nearest rule is employed.
- Lu et al. [13] propose a case-base editing technique based on competence preservation and enhancement [119]. Their solution consists of three stages: the first one compares the distribution between two windows in order to detect if there is a drift or not. Apart from detecting the drift, this method also limits the area where the distribution changes most. After that the Noise-Enhanced Fast Context Switch (NEFCS) method is applied. NEFCS examines all new cases and determines whether there is noise or not (enhancement). However only the noisy cases that lie outside the detected competence areas are removed, because they may be part of novel concepts. Stepwise Redundancy Removal (SRR) method is aimed at controlling the size of the case-base (preservation). SRR removes redundant examples recursively until the case-bases' coverage starts to deteriorate.

Table 2 lists the most relevant instance selectors for drifting streams. We can draw three major types of selection from this table: competence-based, weighting-based, and accuracy-based. Competence-based methods (like CBE or ICF) tend to be more accurate but time-consuming, because they require a constant update of the competence model. Distance-based selection strategies can require even more time than competence-based models, when the number of distances and/or the features involved are high. Accuracy-based methods have difficulties in identifying noisy examples coming during drifts. Finally, feature weighting techniques tends to over-fit data and to perform worse than instance selectors according to [113].

Another relevant topic to be considered when electing instance selectors is whether enhancement and/or maintenance tasks are applied or not. Competence-based methods usually consists of two techniques, one for noise removal and another for redundancy. Redundancy is mainly ignored in accuracy-based techniques since most of them select instances according to the number incorrect predictions committed by each one. Distance-based algorithms implicitly removes redundancy through the space factor in the distance formula.

4.3. Feature space simplification

Discretization algorithms for data stream scenarios must also be able to handle the appearance of concept drifts. Definition and number of discretization intervals may change over time, following shifts in data characteristics. Therefore, it is desirable that discretization intervals are able to smoothly adapt to concept drift, without imposing increased computational cost when being recalculated.

Equal-frequency discretization (based on histograms) can be considered as one of the first techniques in dealing with incremental discretization. By using quantiles as cut points, the feature space can be partitioned in equal-frequency intervals. Estimation of quantiles in streams have been studied in depth in the literature, in approximate [11,125] and exact [126,127] forms. One of the agilest and most effective discretization alternatives is Incremental Discretization Algorithm (IDA) [11]. IDA approximates quantiles through the maintenance of a reservoir sample of the input stream. Intervals here are structured using interval heaps, an efficient data structure that allows to insert and delete elements in $O(log(n))$, and to retrieve the maximum and minimum (the interval boundaries) in constant time. As in most of cases it is not feasible to maintain a complete record of all data, approximative solutions have shown much more suitable for processing high-throughput streams than exact solutions.

Other techniques based on frequency has relied on establishing size thresholds assigned bins to cope with evolving discretization. Lu et al. [128] presented the Incremental Flexible Frequency Discretization (IFFD) algorithm. IFFD defines a range instead of a strict number of quantiles. If the updated intervals' frequency reaches the maximum and the resulting frequencies are not below the minimum (in order to prevent a high classification variance), IFFD splits the interval into two partitions.

Equal-width discretizer is another unsupervised approach that only requires as input the range of features and the number of splitting intervals. However, the main drawback here is that both approaches require streamed records arriving in random order, which is impossible in many learning problems.

Another important requirement to be considered is that some incremental algorithms require to maintain the same set of cut points (number, structure and meaning) over time [11]. That is the case of the most discriminative learning algorithms. Here usage of either an equal-width or an equal-frequency discretizer is suggested, as both of them define the number of bins in advance. Other static algorithms (e.g.: NB) does not require the preservation of intervals during subsequent predictive phases, but only to save some statistics for the current discretization step. However, generalization capabilities of such classifiers are still affected by such displacements in definitions, specially if they are sharp.

According to [129], one of the main problems of unsupervised discretizers is the necessity of defining the number of intervals in advance. Such decision can be assisted by some pre-defined rules (e.g., Sturges' rule) or by an exploratory analysis process. However, exploratory analysis is no longer possible in the present days where the number of instances is too large and pre-defined rules have shown to work only with small-sized datasets. However, unsupervised discretizers are naturally designed for streaming environments since the number of intervals remains invariant.

Most of supervised approaches tend to perform several merges and splits before obtaining a functional final scheme. Abrupt changes in intervals' definition may negatively influence the online learning process. Therefore, methods should strive for a smoother transitions. We present a short list of supervised discretization approaches:

- Gama et al. [129] presented the Partition Incremental Discretization algorithm (PiD), consisting of two layers. The first one summarizes data and creates the preliminary intervals, which will be optimized in the next layer. An equal-width strategy can be used to initialize this step. Then the first layer is updated through a splitting process whenever the number of elements in an interval is above a pre-defined threshold. The second layer performs a merging process over the previous phase in order to yield the final discretization scheme. Any discretizer can be used in the second layer, since the intervals generated in the previous phase are used as inputs. Minimum Description Length Discretizer is used as reference in the original paper. However, there are three main reasons for criticism of PiD approach. Firstly, there is no exact correspondence between the first layer and the second one, which produces inaccuracies that will chain and increase over time. Secondly, if the distribution of data is highly skewed, the number of intervals generated will dramatically increase, due to frequency overflowing. Finally, the splitting process may become even more inaccurate if many repetitions of a single value appear. In this case such a cut point might be generated that divides instances with the same feature values into two different bins, leading to inconsistencies.
- In [130] an online version of ChiMerge (OC), which maintains the O($nlog(n)$) time complexity held by the original algorithm, is proposed. In order to guarantee equal discretization results, authors implement an online approach based on sliding windows. Several data structures are being used to emulate the same behavior held by the original version. Despite of the great effectiveness claimed by the authors, a high increase in the memory usage derived from the set of data structures is displayed by this online version. This fact may prevent from its usage in some data stream scenarios with limited computational resources.

Table 3
Summary description of streaming discretization methods. Information about the name and type of discretization strategy is shown here.

Method	Discretization strategy
PiD [129]	Binning & information (split & merge)
OC [130]	Statistical (merge)

A brief classification about streaming discretizers is given in Table 3. Two alternatives representing different discretization types [1] are shown here. Classification is performed according to two factors: evaluation measures (statistical/binning/information/others) and the type of interval generation (merging/splitting intervals). The most important lesson here is that there is no wrapper online discretization solution. An approach that generate intervals by means of an online classifier weights, as proposed before by some feature selectors, would be highly suitable for this task. A wrapper approach could even solve the problem of displacements in intervals' definitions due to the closer relationship between the classifier and discretizer.

5. Experiments

In this section, we evaluate the usefulness and performance of the data preprocessing algorithms for mining data streams from different perspectives:

- Effectiveness: measured as the number of correctly classified instances divided by the total number of instances in the training set (accuracy). It can be considered as the most relevant factor in measuring usefulness of proposals.
- Time and memory performance: measured as the total time spent by the algorithm in the reduction/discretization phase. Usually performed before the learning phase, although sometimes it runs simultaneously to the prediction phase. Additionally, memory usage for the preprocessing step is being measured to show the resource consumption displayed by each tested method.
- Reduction rate: measured as the amount of reduction accomplished with respect to the original set (in percentage). For selection methods, it is related to the number of rows/columns removed, whereas for discretization, it is related to the degree of simplification of the feature space.

The experimental framework is defined in Section 5.1. Here, the list of datasets and methods, and other considerations are presented. The results and discussion of examined algorithms are presented with respect to the type of task being performed. Each task requires different settings due to its specific characteristics, which will be explained in each section. The order is as follows: FS (Section 4.1), IS (Section 4.2), and discretization (Section 4.3).

5.1. Experimental framework: datasets, methods and parameters

Table 4 shows the complete list of artificial and real datasets used in our experiments to evaluate the reduction techniques. Artificial datasets have been generated using Massive Online Analysis (MOA) benchmark [131], providing a wide range of drifting environments (blips, sudden and gradual, among others described in

Table 4
Relevant information about classification datasets. For each row, the number of instances evaluated (#Inst.), the number of attributes (#Atts.) (which ones are numerical (#Num.) and which ones nominal (#Nom.)), the number of classes (#Cl), and whether the dataset is artificially generated or not (artificial) are shown.

Data set	#Inst.	#Atts.	#Num.	#Nom.	#Cl.	Artificial
blips	500,000	20	20	0	4	yes
gradual_drift	500,000	3	3	0	2	yes
gradual_recurring_drift	500,000	20	20	0	4	yes
incremental_fast	500,000	10	10	0	4	yes
incremental_slow	500,000	10	10	0	4	yes
no_drift	500,000	24	0	24	10	yes
sudden_drift	500,000	3	3	0	2	yes
airlines	539,383	6	3	3	2	no
covtypeNorm	581,011	54	10	44	7	no
elecNormNew	45,311	8	7	1	2	no
kddcup_10	494,020	41	39	2	2	no
poker-lsn	829,201	10	5	5	10	no
spambase	4601	57	57	0	2	no
spam_nominal	9324	40,000	0	40,000	2	no
usenet_recurrent	5931	659	0	659	2	no
spam_data	9324	499	0	499	2	no
usenet1	1500	100	0	100	2	no
usenet2	1500	100	0	100	2	no
usenet3	5997	27,893	0	27,893	2	no
power_supply	29,928	2	2	0	24	no

Table 5
Parameters of methods. Default values for each block of methods are detailed in first rows. Unless specified, these values are common to every method in block.

Method	Parameters
Feature selection	window size = 1 (default)
NB	–
IG [70]	–
SU [72]	–
OFS [77]	$\eta = 0.2$, $\lambda = 0.01$
Instance selection	$k = 3$, window size = 100 (default)
kNN	window size = 1
NEFCS-SRR [13]	l = 10, pmax = 0.5, size limit = 1000
CBE [118]	–
ICF [116]	–
FISH [121]	learner = kNN, distance proportion (time/space) = 0.5, window size = 1
Discretization	initial elements = 100, window size = 1 (default)
NB	–
OC [130]	–
PiD [129]	$\alpha = 0.75$, initial bins = 500, instances to update layer #2 = 10,000, min/max = 0/1

Section 2). Each artificial dataset has been created using different combinations of generators and different parameter values. For a complete description of datasets, and source code, please refer to our GitHub repository[2].

Real datasets come from different sources:

- *airlines, elecNormNew, poker-lsn*, and *covtypeNorm* can be found in MOA's streams repository.
- *spam_data, usenet1, usenet2*, and *usenet3* are e-mail datasets affected by concept drift, collected by The Machine Learning and Knowledge Discovery (MLKD) group (http://mlkd.csd.auth.gr/concept_drift.html).
- *spambase* is a collection of e-mails classified as spam [132].
- *kddcup_10, spam_nominal* (SpamAssasin), and *usenet_recurrent* were collected by Dr. Gama and his research group KDUS (http://www.liaad.up.pt/kdus/products/datasets-for-concept-drift).
- Last dataset (*power_supply*) comes from Stream Data Mining Repository (http://www.cse.fau.edu/~xqzhu/stream.html), and contains power supply registers collected hourly from an electricity company.

Not all datasets described above have been used for every experiment. Some algorithms are designed to deal with a particular data types. For instance, most of feature selectors require discrete features, especially if they utilize information-based measures. Because MOA generators [131] only generate datasets with continuous attributes, these datasets will not be considered for FS. The final choice of datasets and any detail concerned to their features will be described in further sections.

No previous fixed partitioning has been performed on datasets, instead an online evaluation approach has been elected to asses the quality of methods, known as **interleaved test-then-train**. This technique, proposed by Bifet et al. in [133], defines a model in which each example/batch (arriving at time t) is evaluated against $t-1$-model, and then it serves as input to update that model and forms the subsequent t-model.

Reduction techniques used in experiments are listed and grouped by task in Table 5. The default parameter values has been established according to the authors' criteria. Common parameters, like window size or the number of initial elements to consider before starting the reduction process, tends to have common values within the same group. A window size equal to one means that the algorithms work in an online manner, whereas a value higher than one implies a batch-based processing. For instance, FS and discretization methods are suitable for online scenarios, whereas most of instance selectors process elements in batches (except FISH and kNN).

As most of feature selectors and discretizers are focused on NB, it has been elected as a base classifier for these groups. Likewise, kNN serves as reference for instance selectors. Training and testing processes are performed differently for each task.

In FS contingency tables in NB are updated whenever an example arrives. During the classification phase NB only makes predictions by considering the most relevant features.

Training in discretization is also accomplished following the previous scheme, with the particularity that the structure of contingency tables may change whenever new intervals are generated. A new discretization scheme means old model will be outdated and the amount of errors will sharply increase.

As to IS, those methods with best results according to [13] have been selected for our experiments. Different update schemes have been adopted depending on the original design held by each selector. For kNN and FISH, an instant-update scheme has been adopted. In this scheme new instances are immediately added to the case-base. Note that this approach gives kNN a clear advantage over the rest of methods since an ever-updated case-base tends to adapt well to changes. However, it also introduces a lot of redundancy which does not affect accuracy.

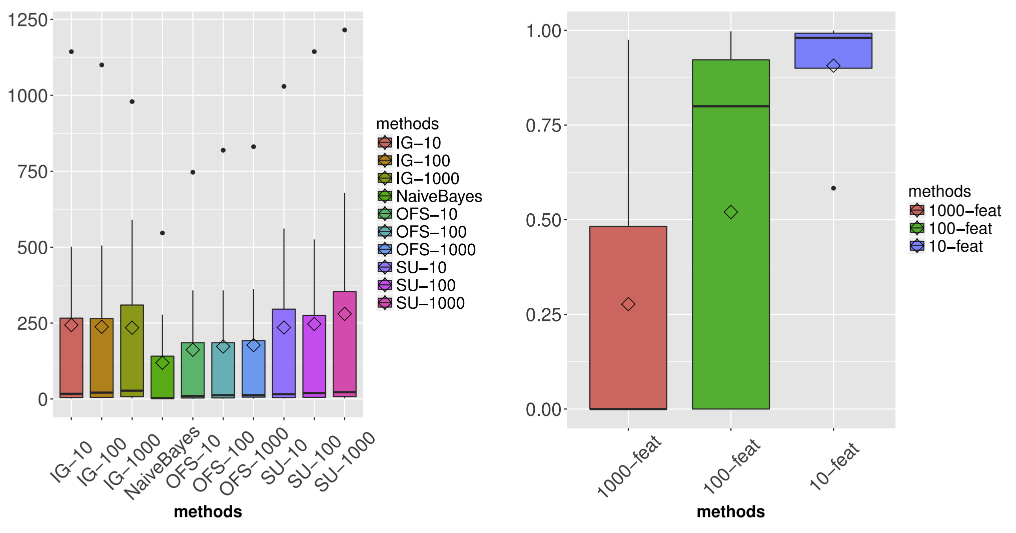
FISH selects a different training set whenever a new example arrives, thus acting in an online way. In counterpart, NEFCS shows a batch-like behavior which requires two windows for drift detection. Here, the updating of the case-base is deferred until a complete batch of examples is available. For a fair comparison between competence models (CBE, ICF, NEFCS-SRR), we have adopted a model based on batches for all these algorithms. New instances are immediately added to the case-base in CBE and ICF, but reduction is only performed when the batch size condition is met.

The whole experimental environment has been executed in a single standard machine, with the following features: 2 processors Intel Core i7 CPU 930 (4 cores/8 threads, 2.8 GHz, 8 MB cache), 24 GB of DDR2 RAM, 1 TB SATA HDD (3 Gb/s), Ethernet network connection, CentOS 6.4 (Linux). Examined algorithms have been

[2] https://github.com/sramirez/MOAReduction

Table 6
Final test accuracy by method (FS). The best outcome for each dataset is highlighted in bold. The second row in header represents the number of feature selected. No selection is performed for NB.

	Naïve Bayes	InfoGain			SU			OFS		
		10	100	1000	10	100	1000	10	100	1000
spam_data	90.6692	89.2750	90.8516	90.6692	88.9103	90.4333	90.6692	90.0579	**91.7417**	90.6692
spam_nominal	**100.0000**	**100.0000**	**100.0000**	**100.0000**	**100.0000**	**100.0000**	**100.0000**	**100.0000**	**100.0000**	**100.0000**
usenet1	**63.3333**	53.6667	**63.3333**	**63.3333**	53.2667	**63.3333**	**63.3333**	58.3333	**63.3333**	**63.3333**
usenet2	**72.1333**	66.9333	**72.1333**	**72.1333**	66.6667	**72.1333**	**72.1333**	68.2000	**72.1333**	**72.1333**
usenet3	**84.6038**	68.8073	78.2319	82.9024	69.0242	77.8816	82.8691	54.0951	57.4646	70.5922
usenet_recurrent	**100.0000**	**100.0000**	**100.0000**	**100.0000**	**100.0000**	**100.0000**	**100.0000**	**100.0000**	**100.0000**	**100.0000**
no_drift	51.4120	**51.4240**	51.4120	51.4120	**51.4240**	51.4120	51.4120	32.5830	51.4120	51.4120
MEAN	**80.3074**	75.7295	79.4232	80.0643	75.6131	79.3134	80.0596	71.8956	76.5836	78.3057



(a) Selection time (in seconds)

(b) Reduction rate (in percentage)

Fig. 3. Box-plot representation for selection time and reduction (FS).

integrated in MOA software (16.04v) as an extension library [3]. MOA has also served as benchmark for our experiments.

5.2. *Feature selection*

Here, we evaluate how well selection of relevant features is performed by the streaming methods. As most of these approaches assume features are discrete, we have only selected from Table 4 those benchmarks with no numerical attributes. Please note that all these datasets comes from the text mining field, in which each attribute represents the presence or the absence of a given word. These datasets fits well for FS as the corpus of words/features is normally quite large.

Firstly, in Table 6 we measure the classification accuracy held by the three feature selectors considered in the experimental framework: IG, SU, and OFS; plus native NB using all features. From these results, we can conclude that:

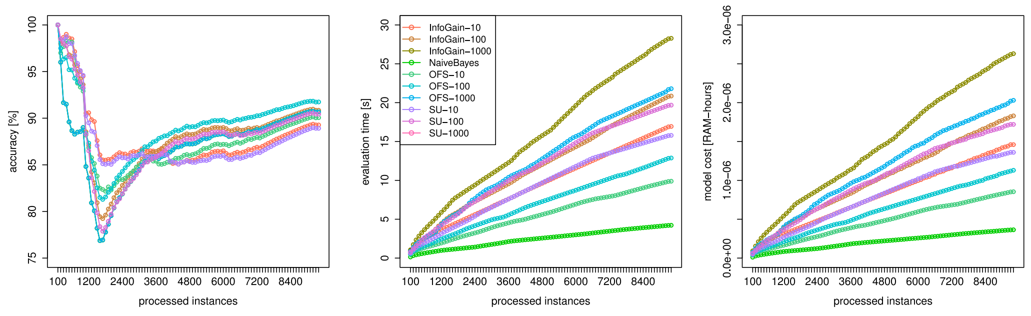
- NB yields better accuracy when all features are available during prediction. None of the selection schemes show better effectiveness than NB.
- However, IG and SU generate results pretty close to NB, with the advantage of generating much simpler solutions (as can be seen in Fig. 3b).
- Information-based methods are more accurate than OFS (based on feature weighting). Specially remarkable are the *spam_data* and *usenet_recurrent* cases, where even with only ten words the classifier is able to predict perfectly all examples.

To assert that no method is better than NB, we convey an statistical analysis on classification accuracy results through two non-parametric tests: Wilcoxon Signed-Ranks Test (one vs. one) and Friedman–Holm Test (one vs. all) [134,135]. Wilcoxon Test conducts pairwise comparisons between the reference method and the rest. A level of significance $\alpha = 0.05$ has been chosen for this experiment. The first two columns in Table 7 show Wilcoxon's results for accuracy, where '+' symbol indicates the number of methods outperformed by each algorithm in row. Symbol '±' represents the number of wins and ties yielded by each method. The best value by column is highlighted by a shaded background. The remaining columns show the results for the Friedman test. The first one shows effectiveness ranking of methods, ordered from the best mark (top row) to the worst. Note that the best method is established as the control algorithm. The second column contains the adjusted p-values for each method according to the post hoc Holm's test. The same level of significance ($\alpha = 0.05$) has been established for this test.

According to the results shown in Table 7, we can assert no method is significantly better than NB without discretization when using 10 features. As to 100 and 1000 features, the new outperforming method is SU although without showing statistically significance with respect to most of the alternatives.

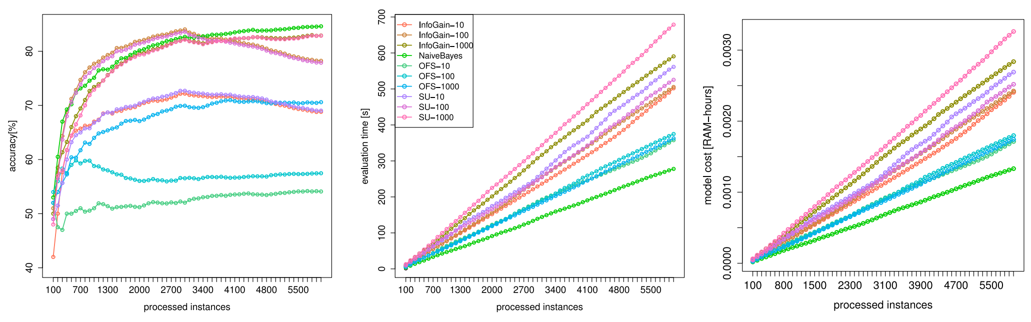
Fig. 3 depicts selection time spent by each algorithm, as well as the amount of reduction performed by each selection scheme, ranging from ten to one thousand features. No selection stands

[3] http://moa.cms.waikato.ac.nz/moa-extensions/



(a) Prequential accuracy. (b) CPU usage. (c) Memory usage.

Fig. 4. Plots of prequential accuracy (in %), CPU processing time (in seconds) and memory usage (in RAM-hours) over the data stream progress (processed instances) for feature selection methods on *spam_data* benchmark.



(a) Prequential accuracy. (b) CPU usage. (c) Memory usage.

Fig. 5. Plots of prequential accuracy (in %), CPU processing time (in seconds) and memory usage (in RAM-hours) over the data stream progress (processed instances) for feature selection methods on *usenet3* benchmark.

Table 7
Wilcoxon test results and average rankings of feature selectors (Friedman Procedure & Adjusted *p*-value with Holm's Test) for accuracy.

Algorithms	Accuracy +	Accuracy ±	Ranking	p_{Holm}
NB	1	3	7	–
OFS-10	0	2	16.7143	0.063201
InfoGain-10	0	3	17.1429	0.063201
SU-10	0	3	17.1429	0.063201

(a) 10 features selected

Algorithms	Accuracy +	Accuracy ±	Ranking	p_{Holm}
SU-100	0	3	11.6429	–
OFS-100	0	3	11.9286	0.017455
NB	0	3	17.2143	0.615351
InfoGain-100	0	3	17.2143	0.615351

(b) 100 feature selected

Algorithms	Accuracy +	Accuracy ±	Ranking	p_{Holm}
SU-1000	0	3	10.5714	–
OFS-1000	0	3	14.1429	0.487181
NB	0	3	16.5714	0.487181
InfoGain-1000	0	3	16.7143	0.487181

(c) 1,000 feature selected

as the fastest alternative. Despite the complete set of feature is used for predictions, this alternative offers better results due to the avoidance of feature relevance computations. Among the selection alternatives, OFS performs faster than the information-based selectors. However, OFS has shown in Table 6 to obtain less accurate schemes than its competitors.

Although a better reduction rate is achieved in *10-features* scheme (close to 100% in mean), by selecting 1000 features we can yield better accuracy, while the obtained reduction rate is still acceptable ($> 25\%$). Please note that no selection is conducted on 4/7 problems when we choose the *1000-features* scheme since there are not enough attributes to select.

In conclusion, *SU-1000* can be elected as the best choice because of its competitive accuracy results similar to those yielded by NB and displayed reduction rates. Time results do not show significant differences between examined methods.

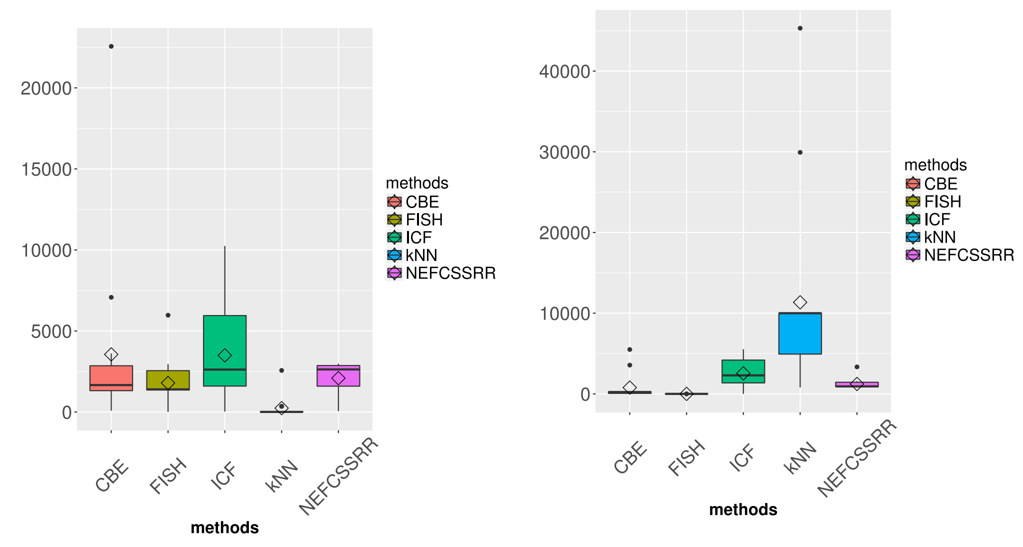
Detailed results on the entire data stream for *spam_data* and *usenet3* benchmarks with respect to obtained prequential accuracies, CPU usage and memory usage are depicted in Figs. 4 and 5.

5.3. Instance selection

Here, we evaluate how IS methods perform in non-stationary environments. As opposed to Section 4.1, in this experiment we have included datasets with both numerical and nominal attributes. In previous experiments [13] instance selectors were shown to be impractical when dealing with medium datasets. Because of that we have discarded those problems with a number of instances $> 100,000$. Additionally, we have created new artificial datasets with a lower number of examples (10,000 instances).

Accuracy displayed by examined methods are given in Table 8. Table 9 shows results on accuracy for the Wilcoxon and Friedman–Holm test, following the same scheme presented in Section 4.1. From these results, we can conclude that:

- The best method on average is the updated kNN without selection (80.49%). The closest competitor (CBE) is five units below kNN. Other online methods, like FISH or ICF, do not respond well to concept drifts.
- No method is statistically better than updated kNN. Although kNN wins in each pairwise comparison in Wilcoxon tests, it only significantly overcomes ($\alpha = 0.05$) FISH and ICF according to Friedman–Holm tests.



(a) Selection time (in seconds)

(b) Reduction rate (in # instances selected)

Fig. 6. Box-plot representation for selection time and reduction (IS).

Table 8
Total test accuracy by method (IS).

	NEFCSSRR	ICF	CBE	FISH	kNN
elecNormNew	67.7652	43.7882	73.8105	60.0455	**84.0815**
powersupply	11.9400	4.2502	12.3800	5.4435	**15.1296**
spambase	81.6779	39.3827	**97.5440**	95.9139	95.1532
spam_data	88.8782	25.6113	91.1197	77.3488	**93.9833**
spam_nominal	**100.0000**	**100.0000**	**100.0000**	**100.0000**	**100.0000**
usenet1	**56.6667**	54.5333	54.0667	55.2000	56.4667
usenet2	61.2000	63.4667	48.4000	**69.8000**	68.2000
usenet_recurrent	**100.0000**	**100.0000**	**100.0000**	**100.0000**	**100.0000**
blips	90.8900	34.1300	94.2300	31.3200	**97.1800**
sudden_drift	76.5300	61.7300	74.2300	60.8700	**82.6600**
gradual_drift	68.2700	52.3600	74.4500	52.0200	**81.2700**
gradual_recurring_drift	87.5800	28.9400	92.6500	28.8400	**96.3300**
incremental_fast	65.8900	51.7700	68.4000	55.8300	**77.2800**
incremental_slow	72.4300	50.9800	68.7000	56.5800	**79.1000**
MEAN	73.5513	50.7816	74.9986	60.6580	**80.4882**

Table 9
Wilcoxon test results and average rankings of methods (Friedman Procedure & Adjusted *p*-value with Holm's Test) for accuracy.

Algorithms	Accuracy		Ranking	p_{Holm}
	+	±		
kNN	4	4	18.2143	–
CBE	2	3	26.1429	0.321710
NEFCSSRR	2	3	29	0.321710
FISH	0	1	47.5	0.000421
ICF	0	1	56.6429	0.000002

- Selection methods make decisions about the relevance or difficulty of a given instance without knowing the future state of the stream. It is normal that the no-selection option always performs better than others. It only depends on the amount of noise introduced by each problem and not by other factors like redundancy.

Regarding reduction and time, Fig. 3 depicts the distribution for both variables. From these plots, we can claim that CBE can be considered as the most accurate solution, and it also offers the highest reduction rates, at the cost of increased time complexity. NEFCSSRR also represents an interesting option as this method shows precise, and performs faster than CBE. The outstanding reduction rate of FISH is explained because it normally selects the kNN for each new example. This fact also explains its poor outcome on accuracy.

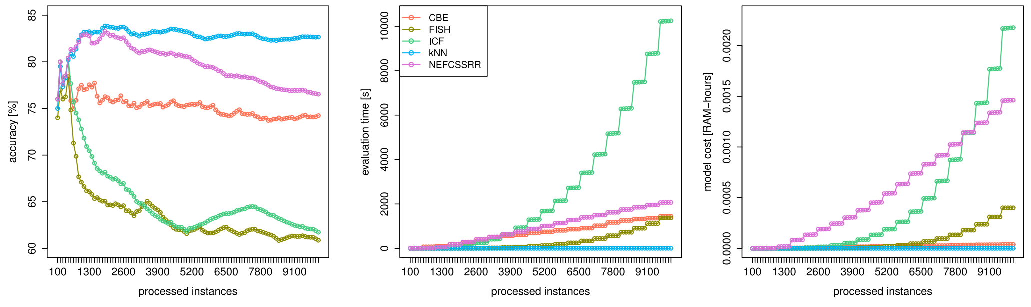
Detailed results on the entire data stream for *sudden_drift* and *gradual_drift* benchmarks with respect to obtained prequential accuracies, CPU usage and memory usage are depicted in Figs. 7 and 8.

Table 10
Classification test accuracy after discretization.

	PiD	IDA	OC	Naïve Bayes
airlines	63.0057	64.1563	**65.0723**	64.5504
powersupply	2.9237	13.5793	11.2938	**16.1087**
elecNormNew	71.9522	**76.6905**	74.0731	73.3625
spambase	**98.0439**	97.8700	97.6744	82.8081
kddcup_10	**99.1474**	98.4644	98.1404	97.1908
poker-lsn	55.0335	59.4337	58.5465	**59.5528**
covtypeNorm	**66.6306**	62.7235	64.2254	60.5208
blips	**74.5680**	66.4494	64.2148	60.9060
sudden_drift	65.7736	81.3168	77.8808	**83.8144**
gradual_drift_med	60.8404	82.8908	80.1032	**84.7000**
gradual_recurring_drift	**65.1678**	58.5250	58.5612	56.7450
incremental_fast	73.9900	75.6472	75.6036	**76.3642**
incremental_slow	65.6074	76.9186	75.4316	**78.0688**
MEAN	66.3603	**70.3589**	69.2939	68.8225

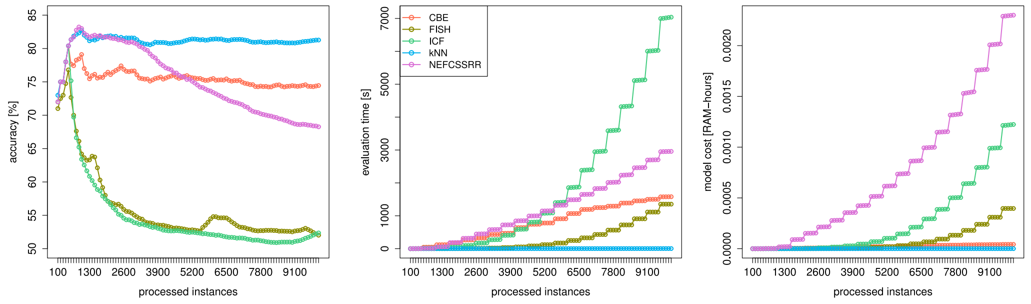
5.4. Discretization

To evaluate the ability of supervised discretizers to reduce the continuous feature space, we propose a new study with three discretization methods for data streams. NB and Incremental Discretization Algorithm (IDA) [11] have been elected as benchmark to assess the quality of supervised discretization schemes. The first one employs a gaussian estimation method, whereas the second one employs an unsupervised scheme based on quantile-estimation. In this experiment, only datasets with at least one numerical attribute have been considered. Email-based dataset used in Section 4.1 are thus discarded.



(a) Prequential accuracy. (b) CPU usage. (c) Memory usage.

Fig. 7. Plots of prequential accuracy (in %), CPU processing time (in s.) and memory usage (in RAM-hours) over the data stream progress (processed instances) for instance selection methods on *sudden_drift* benchmark.



(a) Prequential accuracy. (b) CPU usage. (c) Memory usage.

Fig. 8. Plots of prequential accuracy (in %), CPU processing time (in s.) and memory usage (in RAM-hours) over the data stream progress (processed instances) for instance selection methods on *gradual_drift* benchmark.

Table 11
Wilcoxon test results and average rankings of methods (Friedman Procedure & Adjusted p-value with Holm's Test) for accuracy.

Algorithms	Accuracy		Ranking	p_{Holm}
	+	±		
IDA	1	3	21.7692	–
OC	0	2	26	0.905821
Naïve Bayes	0	3	26.2308	0.905821
PiD	0	3	32	0.255677

Tables 10 and 11 contain test accuracy results for NB classification with and without explicit discretization. From these results, we can conclude the following statements:

- The most accurate method (in average) is IDA, an unsupervised method based on quantile-estimation and a sampling approach. However, its results are pretty close to those obtained by OC and NB. OC also outperforms the base solution, but with smaller margin than presented by IDA.
- According to the Wilcoxon test, we can statistically assert that IDA is only better than OC. Nevertheless this claim is rejected by Friedman's procedure, with a *p*-value far from the standard acceptance thresholds: 0.9 or 0.95. Although some improvement can be achieved by using supervised discretization, it can be deemed as superfluous and likely suboptimal.
- PiD represents the worst choice in this framework. Yet, it is specially remarkable that PiD is able to obtain the best accuracy mark in 5/13 datasets, with an outstanding mark in the blip dataset. This fact can be explained by the high number of parameters to be tuned in PiD and the high dependency on their values. Among the list of parameters, a global minimum and the maximum value need to be defined for the whole set of features, which is unfeasible in streaming environments. This parameter is essential as determines the expansion rate for new intervals, thus it may be possible to tailor it specifically for some datasets.

Apart from the previous deficiencies, Fig. 9a shows a high time-complexity of OC, as a result of a high number of data structures (binary tree, several queues, etc.) to be managed. IDA holds a similar time performance to NB.

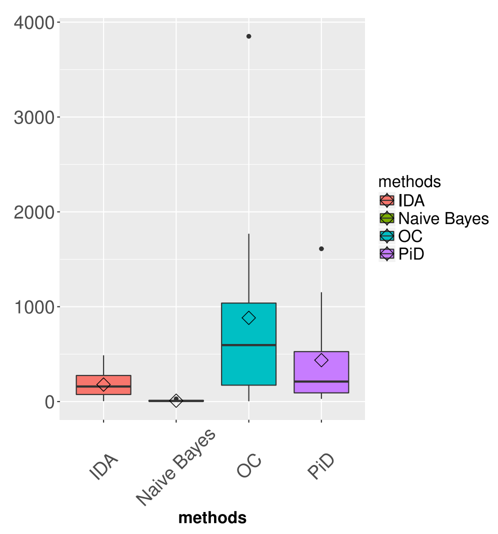
Fig. 9b illustrates the reduction performed by each method, represented as number of intervals generated per method. In this case, OC obtains the simplest solutions thanks to the control performed by χ^2. IDA defines the number of intervals before launching any process and PiD's inaccuracy is explained by a huge number of intervals generated initially ($\approx$ 500 per feature), as well as during the splitting process. Please notice that the subsequent merging process launched by the second layer is just not able to efficiently reduce such many input intervals.

As discussed before, evolving intervals sharply affect streaming classification since new and deleted intervals normally imply dramatic changes in the learning process. New models and techniques with a better interaction between discretization and classification must be designed if we want to transform online discretization into a truly useful tool for data analytics.

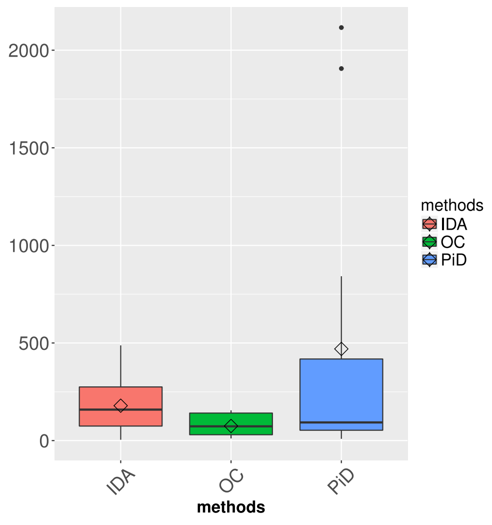
Detailed results on the entire data stream for *sudden_drift* and *gradual_drift* benchmarks with respect to obtained prequential accuracies, CPU usage and memory usage are depicted in Figs. 10 and 11.

6. Data preprocessing for data stream mining: lessons learned and future directions

In this section we will discuss observations made on the basis of the presented survey of existing preprocessing methods for data streams, as well as the accompanying experimental study. Then, we will outline open challenges and future directions in this field.

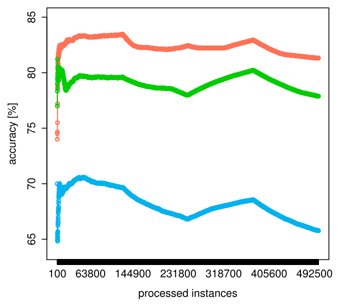


(a) Selection time (in seconds)

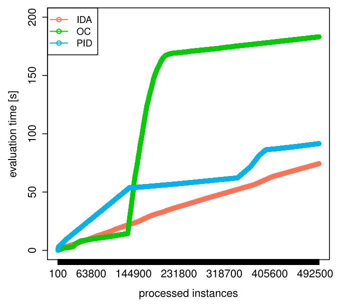


(b) Reduction rate (in # intervals generated)

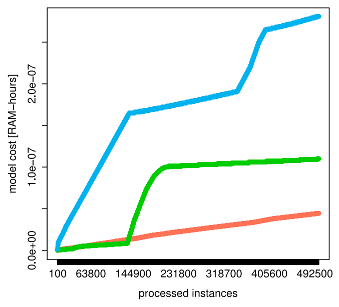
Fig. 9. Box-plot representation for discretization time and reduction (discretization).



(a) Prequential accuracy.

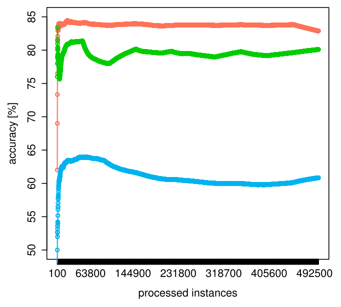


(b) CPU usage.

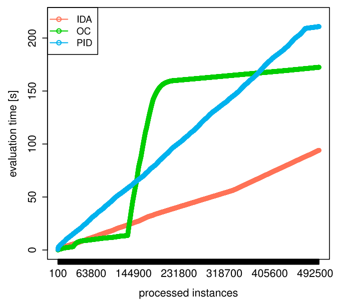


(c) Memory usage.

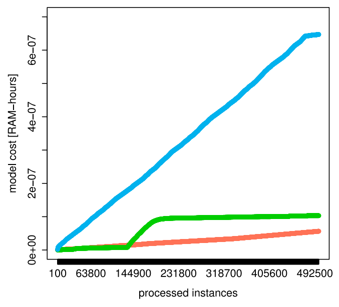
Fig. 10. Plots of prequential accuracy (in %), CPU processing time (in s.) and memory usage (in RAM-hours) over the data stream progress (processed instances) for discretization methods on *sudden_drift* benchmark.



(a) Prequential accuracy.



(b) CPU usage.



(c) Memory usage.

Fig. 11. Plots of prequential accuracy (in %), CPU processing time (in s.) and memory usage (in RAM-hours) over the data stream progress (processed instances) for discretization methods on *gradual_drift* benchmark.

6.1. *Lessons learned*

Some important outcomes and guidelines can be inferred from the study, which we enumerate below:

- A wide range of phenomenons specific to data stream mining, ranging from concept-evolution to dynamic feature space, directly affects the features describing incoming instances. DXMiner is the only system that address all these problem through a combined strategy based on an information-based FS and an unsupervised selection method.
- As expected FS does not improve accuracy results presented by the option with the full set of features. Nevertheless FS solutions are able to yield simpler solutions with similar predictive performance, which is of crucial importance to stream mining frameworks. SU, the selector included in DXMiner, can be elected as the best method for FS because of its outstanding results in accuracy and its low complexity.

- Competence-based methods for IS tend to maintain case-bases polished in the highest degree, free of noise and redundancy. However they are characterized by a very high computational complexity. In distance-based solutions, this overhead is even bigger, while not being balanced by any gains in overall accuracy. In general, all instance selectors have shown an unfavorable behavior with respect to time and memory requirements, thus preventing their meaningful applications in high-speed data stream mining.
- CBE can be elected as best option for IS in terms of precision and reduction. NEFCSSRR also presents itself as an interesting option, with similar results to CBE, however requiring more computational resources.
- When selecting preprocessing methods for data stream mining we must consider not only the obtained accuracy, but also the computational costs that are associated with this method. As our study clearly showed some of the considered methods are characterized by bottlenecks in either CPU or memory usage, thus making them unsuitable for high-speed data streams.

6.2. Challenges and future directions

Here we outline the main challenges that should be addressed by the research community in order to obtain a meaningful progress in the area of preprocessing techniques for data stream problems:

- A scarce number of online and supervised discretizers have been proposed in the literature so far. Most of current methods are unsupervised techniques based on quantiles, using an adaptation strategy with smooth shifts in intervals' definition and the previous definition of intervals. Adding class information to the discretization process would allow to accommodate for local drifts, where properties of only some class changes. Additionally, we envision the potential of ensemble learning that will allow to use various discretization intervals to allow for training a diverse set of classifiers.
- Current online discretizers have shown to perform poorly as their adjustments tend to be more abrupt than those yielded by quantile-based techniques (see Section 4.3). However, this problem has been compensated by the inclusion of class information in the discretization process. Abrupt tweaks and labelings are two major concerns that must be addressed by further developments in this area. This shows that there is a need for combining discretization with active learning solutions. This would allow for selective labeling of only these samples that yield highest probability of influencing the intervals' definitions.
- No pure wrapper-based solutions have been proposed for online problems yet. Efficient implementations of these methods may be challenging die to their increased computational cost, but this may be compensated by the inherent discriminative ability of online learners and their adaptiveness to drifts. One potential solution would be to combine filter and wrapper approaches in order to reduce the number of times the more costly method will be used and to allow for continuous classification even during the wrapper computation. Another potential solution lies in using high-performance solutions based on GPU or distributed computing to reduce the computational load connected with this approach [136].
- There is a need for further research on feature and instance selection methods that can directly address the problem of concept drift. One way of approaching this would be to combine instance selection approaches with drift detection module that could directly influence the usability of prototypes. Whenever a strong drift is being detected, one may discard the previous prototypes and use only the incoming objects. After stream stabilizes, the instance selection can be repeated to adapt to the current concept. Another potential solution is to have weighted prototypes, where weight would reflect how long time ago they were created and how useful they are to mining current state of the stream. This would allow to smoothly forget outdated prototypes, while keeping in memory the ones still useful. Local drifts that occur only within a subset of classes should also be considered. In such a case only selected prototypes must be modified in the areas of drift presence. This would require class-based prototype pruning methods and a method to overlook the influence of these drifting prototypes on stationary classes.
- There is a need for further developing preprocessing methods characterized by a low computational requirements that would allow for a real-time decision making when dealing with big and high-speed data streams [137]. In case of data stream mining one must always balance the obtained accuracy with the amount of time spent on the computations. Therefore, developing approximate solutions with stopping criteria could be beneficial, especially in cases of sudden changes.
- There exist no solutions that directly take into account the possibility of recurrent concept drifts. Therefore, it seems promising to develop preprocessing methods that could accommodate the fact that previously used set of features / instances / discrete bins may become useful once again in the future. Simplest way of approaching this would be to create a secondary buffer storing these items for a certain amount of time, allowing to reuse them when necessary. To avoid unacceptable memory requirements this buffer should be flushed after a certain period of time with no action.
- There is a need to develop data preprocessing methods for more complex data stream types. Such techniques are crucial for imbalanced [138,139], multi-label [140] and multi-instance [141] problems and should be extended into the streaming framework.

7. Concluding remarks

We have presented a thorough survey of data reduction methods applied to data stream mining. Basic concepts, existing works, and present and future challenges have been analyzed in this work. Based on a number of relevant characteristics, we have proposed a simple, yet useful taxonomy of current developments in the online data preprocessing.

Most relevant methods have also been analyzed empirically through a conscious experimental framework, which includes a long and diverse list of artificial and real datasets with different types of drift. A statistical analysis based on non-parametric tests have been conveyed to support the resulting conclusions.

Concluding this work, we can claim that data preprocessing for data streams is still in its early days. New and more sophisticated methods that deal with previously unsolved challenges need to be designed in the years to follow. Great progress has been made in instance and feature selection, but other tasks like discretization remains yet to be properly addressed.

Acknowledgments

This work is supported by the Spanish National Research Project TIN2014-57251-P, the Foundation BBVA project 75/2016 BigDaPTOOLS, the Andalusian Research Plan P11-TIC-7765, and the Polish National Science Center under the grant no. DEC-2013/09/B/ST6/02264. S. Ramírez-Gallego holds a FPU scholarship from the Spanish Ministry of Education and Science (FPU13/00047).

References

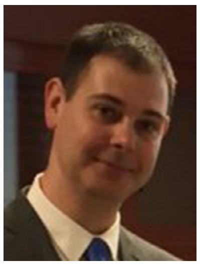
[1] S. García, J. Luengo, F. Herrera, Data Preprocessing in Data Mining, Springer, 2015.
[2] S. García, J. Luengo, F. Herrera, Tutorial on practical tips of the most influential data preprocessing algorithms in data mining, Knowl. Based Syst. 98 (2016) 1–29.
[3] D. Pyle, Data Preparation for Data Mining, Morgan Kaufmann Publishers Inc., 1999.
[4] V. Mayer-Schnberger, K. Cukier, Big Data: A Revolution That Will Transform How We Live, Work and Think., 2013.
[5] S. García, S. Ramírez-Gallego, J. Luengo, J.M. Benítez, F. Herrera, Big data preprocessing: methods and prospects, Big Data Anal. 1 (1) (2016) 9.
[6] J.a. Gama, Knowledge Discovery from Data Streams, Chapman & Hall/CRC, 2010.
[7] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Comput. Surv. 46 (4) (2014) 44:1–44:37.
[8] I. Zliobaite, B. Gabrys, Adaptive preprocessing for streaming data, IEEE Trans. Knowl. Data Eng. 26 (2) (2014) 309–321.
[9] M.M. Masud, Q. Chen, J. Gao, L. Khan, J. Han, B. Thuraisingham, Classification and novel class detection of data streams in a dynamic feature space, in: Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part II, in: ECML PKDD'10, 2010, pp. 337–352.
[10] J.P. Barddal, H.M. Gomes, F. Enembreck, B. Pfahringer, A. Bifet, On dynamic feature weighting for feature drifting data streams, in: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19–23, 2016, Proceedings, Part II, 2016, pp. 129–144.
[11] G. Webb, Contrary to popular belief incremental discretization can be sound, computationally efficient and extremely useful for streaming data, in: IEEE International Conference on Data Mining (ICDM), 2014, pp. 1031–1036.
[12] V. Bolón-Canedo, N.S.-M. no, A. Alonso-Betanzos, Recent advances and emerging challenges of feature selection in the context of big data, Knowl. Based Syst. 86 (2015) 33–45.
[13] N. Lu, J. Lu, G. Zhang, R.L. de Mantaras, A concept drift-tolerant case-base editing technique, Artif. Intell. 230 (2016) 108–133.
[14] M.M. Gaber, Advances in data stream mining, Wiley Interdisc. Rew.: Data Min. Knowl. Discov. 2 (1) (2012) 79–85.
[15] E. Lughofer, P.P. Angelov, Handling drifts and shifts in on-line data streams with evolving fuzzy systems, Appl. Soft Comput. 11 (2) (2011) 2057–2068.
[16] L.I. Kuncheva, Classifier ensembles for detecting concept change in streaming data: overview and perspectives, in: 2nd Workshop SUEMA 2008 (ECAI 2008), 2008, pp. 5–10.
[17] D. Brzezinski, Block-based and Online Ensembles for Concept-drifting Data Streams, Poznan University of Technology, 2015 Ph.D. thesis.
[18] L.L. Minku, X. Yao, A.P. White, The impact of diversity on online ensemble learning in the presence of concept drift, IEEE Trans. Knowl. Data Eng. 22 (2009) 730–742.
[19] I. Khamassi, M. Sayed-Mouchaweh, M. Hammami, K. Ghédira, Self-adaptive windowing approach for handling complex concept drift, Cogn. Comput. 7 (6) (2015) 772–790, doi:10.1007/s12559-015-9341-0.
[20] J. Gama, P. Medas, G. Castillo, P.P. Rodrigues, Learning with drift detection, in: Advances in Artificial Intelligence - SBIA 2004, 17th Brazilian Symposium on Artificial Intelligence, São Luis, Maranhão, Brazil, 29 - October 1, 2004, Proceedings, 2004, pp. 286–295.
[21] A. Bifet, R. Gavaldà, Learning from time-changing data with adaptive windowing, in: Proceedings of the Seventh SIAM International Conference on Data Mining, April 26–28, 2007, Minneapolis, Minnesota, USA, 2007, pp. 443–448.
[22] P. Sobolewski, M. Woźniak, Concept drift detection and model selection with simulated recurrence and ensembles of statistical detectors, J. Univ. Comput. Sci. 19 (4) (2013) 462–483.
[23] R.M.M. Vallim, R.F. de Mello, Proposal of a new stability concept to detect changes in unsupervised data streams, Expert Syst. Appl. 41 (16) (2014) 7350–7360.
[24] B.I.F. Maciel, S.G.T. de Carvalho Santos, R.S.M. de Barros, A lightweight concept drift detection ensemble, in: 27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, 9–11, 2015, 2015, pp. 1061–1068.
[25] M. Woźniak, P. Ksieniewicz, B. Cyganek, K. Walkowiak, Ensembles of heterogeneous concept drift detectors - experimental study, in: Computer Information Systems and Industrial Management - 15th IFIP TC8 International Conference, CISIM 2016, Vilnius, Lithuania, 14–16, 2016, Proceedings, 2016, pp. 538–549.
[26] G. Hulten, L. Spencer, P.M. Domingos, Mining time-changing data streams, in: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, 26–29, 2001, 2001, pp. 97–106.
[27] J. Shan, J. Luo, G. Ni, Z. Wu, W. Duan, CVS: fast cardinality estimation for large-scale data streams over sliding windows, Neurocomputing 194 (2016) 107–116.
[28] B. Krawczyk, M. Woźniak, One-class classifiers with incremental learning and forgetting for data streams with concept drift, Soft Comput. 19 (12) (2015) 3387–3400.
[29] L. Du, Q. Song, X. Jia, Detecting concept drift: an information entropy based method using an adaptive sliding window, Intell. Data Anal. 18 (3) (2014) 337–364.
[30] O. Mimran, A. Even, Data stream mining with multiple sliding windows for continuous prediction, in: 22st European Conference on Information Systems, ECIS 2014, Tel Aviv, Israel, 9–11, 2014, 2014.
[31] P. Domingos, G. Hulten, Mining high-speed data streams, in: I. Parsa, R. Ramakrishnan, S. Stolfo (Eds.), Proceedings of the ACM Sixth International Conference on Knowledge Discovery and Data Mining, ACM Press, Boston, USA, 2000, pp. 71–80.
[32] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, F.E. Alsaadi, A survey of deep neural network architectures and their applications, Neurocomputing 234 (2017) 11–26.
[33] W.M. Czarnecki, J. Tabor, Online extreme entropy machines for streams classification and active learning, in: Proceedings of the 9th International Conference on Computer Recognition Systems CORES 2015, Wroclaw, Poland, 25–27 May 2015, 2015, pp. 371–381.
[34] B. Lakshminarayanan, D.M. Roy, Y.W. Teh, Mondrian forests: efficient online random forests, in: Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, 8–13 2014, Montreal, Quebec, Canada, 2014, pp. 3140–3148.
[35] M. Woźniak, Application of combined classifiers to data stream classification, in: Computer Information Systems and Industrial Management - 12th IFIP TC8 International Conference, CISIM 2013, Krakow, Poland, 25–27, 2013. Proceedings, 2013, pp. 13–23.
[36] M. Woźniak, M. Graña, E. Corchado, A survey of multiple classifier systems as hybrid systems, Inf. Fusion 16 (2014) 3–17.
[37] R. Elwell, R. Polikar, Incremental learning of concept drift in nonstationary environments, IEEE Trans. Neural Netw. 22 (10) (2011) 1517–1531.
[38] Y. Sun, K. Tang, L.L. Minku, S. Wang, X. Yao, Online ensemble learning of data streams with gradually evolved classes, IEEE Trans. Knowl. Data Eng. 28 (6) (2016) 1532–1545.
[39] G. Song, Y. Ye, H. Zhang, X. Xu, R.Y. Lau, F. Liu, Dynamic clustering forest: an ensemble framework to efficiently classify textual data stream with concept drift, Inf. Sci. 357 (2016) 125–143.
[40] L. Canzian, Y. Zhang, M. van der Schaar, Ensemble of distributed learners for online classification of dynamic data streams, IEEE Trans. Signal Inf. Process. Netw. 1 (3) (2015) 180–194.
[41] L.L. Minku, X. Yao, DDD: a new ensemble approach for dealing with concept drift, IEEE Trans. Knowl. Data Eng. 24 (4) (2012) 619–633.
[42] L.L. Minku, A.P. White, X. Yao, The impact of diversity on online ensemble learning in the presence of concept drift, IEEE Trans. Knowl. Data Eng. 22 (5) (2010) 730–742.
[43] A. Bifet, G. Holmes, B. Pfahringer, Leveraging bagging for evolving data streams, in: Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Barcelona, Spain, 20–24, 2010, Proceedings, Part I, 2010, pp. 135–150.
[44] D. Brzezinski, J. Stefanowski, Combining block-based and online methods in learning ensembles from concept drifting data streams, Inf. Sci. 265 (2014) 50–67.
[45] N. Japkowicz, M. Shah, Evaluating learning algorithms: a classification perspective, Cambridge University Press, 2011.
[46] A. Shaker, E. Hüllermeier, Recovery analysis for adaptive learning from non-stationary data streams: experimental design and case study, Neurocomputing 150 (2015) 250–264.
[47] J. Gama, R. Sebastião, P.P. Rodrigues, On evaluating stream learning algorithms, Mach. Learn. 90 (3) (2013) 317–346.
[48] D. Brzezinski, J. Stefanowski, Prequential AUC for classifier evaluation and drift detection in evolving data streams, in: New Frontiers in Mining Complex Patterns - Third International Workshop, NFMCP 2014, Held in Conjunction with ECML-PKDD 2014, Nancy, France, 19, 2014, Revised Selected Papers, 2014, pp. 87–101.
[49] M. Salehi, C. Leckie, J.C. Bezdek, T. Vaithianathan, X. Zhang, Fast memory efficient local outlier detection in data streams, IEEE Trans. Knowl. Data Eng. 28 (12) (2016) 3246–3260.
[50] I. Zliobaite, M. Budka, F.T. Stahl, Towards cost-sensitive adaptation: When is it worth updating your predictive model? Neurocomputing 150 (2015) 240–249.
[51] A. Bifet, G.D.F. Morales, J. Read, G. Holmes, B. Pfahringer, Efficient online evaluation of big data stream classifiers, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, 10–13, 2015, 2015, pp. 59–68.
[52] M. Woźniak, P. Ksieniewicz, B. Cyganek, A. Kasprzak, K. Walkowiak, Active learning classification of drifted streaming data, in: International Conference on Computational Science 2016, ICCS 2016, 6–8 June 2016, San Diego, California, USA, 2016, pp. 1724–1733.
[53] I. Zliobaite, A. Bifet, B. Pfahringer, G. Holmes, Active learning with drifting streaming data, IEEE Trans. Neural Netw. Learn. Syst. 25 (1) (2014) 27–39.
[54] Y. Dong, N. Japkowicz, Threaded ensembles of supervised and unsupervised neural networks for stream learning, in: Advances in Artificial Intelligence - 29th Canadian Conference on Artificial Intelligence, Canadian AI 2016, Victoria, BC, Canada, May 31 - 3, 2016. Proceedings, 2016, pp. 304–315.
[55] M.J. Hosseini, A. Gholipour, H. Beigy, An ensemble of cluster-based classifiers for semi-supervised classification of non-stationary data streams, Knowl. Inf. Syst. 46 (3) (2016) 567–597.

[56] B.S. Parker, L. Khan, Detecting and tracking concept class drift and emergence in non-stationary fast data streams, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 25–30, 2015, Austin, Texas, USA., 2015, pp. 2908–2913.
[57] P. Sobolewski, M. Woźniak, Ldcnet: minimizing the cost of supervision for various types of concept drift, in: Proceedings of the 2013 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments, CIDUE 2013, IEEE Symposium Series on Computational Intelligence (SSCI), 16–19 April 2013, Singapore, 2013, pp. 68–75.
[58] G. Shikkenawis, S.K. Mitra, 2D orthogonal locality preserving projection for image denoising, IEEE Trans. Image Process. 25 (1) (2016) 262–273.
[59] A.A. Mohamad AL-Shiha, W. Woo, S. Dlay, Multi-linear neighborhood preserving projection for face recognition, Pattern Recogn. 47 (2) (2014) 544–555.
[60] H. Zhang, Q.M. Jonathan Wu, T.W.S. Chow, M. Zhao, A two-dimensional neighborhood preserving projection for appearance-based face recognition, Pattern Recogn. 45 (5) (2012) 1866–1876.
[61] G. Doquire, M. Verleysen, Feature selection with missing data using mutual information estimators, Neurocomputing 90 (2012) 3–11.
[62] V. Lopez, I. Triguero, C.J. Carmona, S. Garcia, F. Herrera, Addressing imbalanced classification with instance generation techniques: ipade-id, Neurocomputing 126 (2014) 15–28.
[63] A. Ferreira, M. Figueiredo, Incremental filter and wrapper approaches for feature discretization, Neurocomputing 123 (2014) 60–74.
[64] Y. Yang, G.I. Webb, Discretization for Naive–Bayes learning: managing discretization bias and variance, Mach. Learn. 74 (1) (2009) 39–74.
[65] H.-W. Hu, Y.-L. Chen, K. Tang, A dynamic discretization approach for constructing decision trees with a continuous label, IEEE Trans. Knowl. Data Eng. 21 (11) (2009) 1505–1514.
[66] A. Cano, D.T. Nguyen, S. Ventura, K.J. Cios, ur-caim: improved CAIM discretization for unbalanced and balanced data, Soft Comput. 20 (1) (2016) 173–188.
[67] A. Cano, J.M. Luna, E.L.G. Galindo, S. Ventura, LAIM discretization for multi--label data, Inf. Sci. 330 (2016b) 370–384.
[68] X. Wu, K. Yu, H. Wang, W. Ding, Online streaming feature selection, in: Proceedings of the 27th International Conference on Machine Learning (ICML-10), 2010, pp. 1159–1166.
[69] S. Eskandari, M. Javidi, Online streaming feature selection using rough sets, Int. J. Approx. Reason. 69 (C) (2016) 35–57.
[70] I. Katakis, G. Tsoumakas, I.P. Vlahavas, On the utility of incremental feature selection for the classification of textual data streams, in: Advances in Informatics, 10th Panhellenic Conference on Informatics, PCI 2005, Volos, Greece, November 11–13, 2005, Proceedings, 2005, pp. 338–348.
[71] J.P. Barddal, H.M. Gomes, F. Enembreck, A survey on feature drift adaptation, in: IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI), 2015, pp. 1053–1060.
[72] H.-L. Nguyen, Y.-K. Woon, W.-K. Ng, L. Wan, Heterogeneous ensemble for feature drifts in data streams, in: Proceedings of the 16th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II, in: PAKDD'12, 2012, pp. 1–12.
[73] V.R. Carvalho, W.W. Cohen, Single-pass online learning: performance, voting schemes and online feature selection, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, in: KDD '06, 2006, pp. 548–553.
[74] J. Gomes, M. Gaber, P. Sousa, E. Menasalvas, Mining recurring concepts in a dynamic feature space, IEEE Trans. Neural Netw. Learn. Syst. 25 (1) (2014) 95–110.
[75] X. Wu, K. Yu, W. Ding, H. Wang, X. Zhu, Online feature selection with streaming features, IEEE Trans. Pattern Anal. Mach. Intell. 35 (5) (2013) 1178–1192.
[76] S.C.H. Hoi, J. Wang, P. Zhao, R. Jin, Online feature selection for mining big data, in: Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, BigMine 2012, Beijing, China, 12, 2012, 2012, pp. 93–100.
[77] J. Wang, P. Zhao, S. Hoi, R. Jin, Online feature selection and its applications, IEEE Trans. Knowl. Data Eng. 26 (3) (2014) 698–710.
[78] J. Wang, M. Wang, P. Li, L. Liu, Z. Zhao, X. Hu, X. Wu, Online feature selection with group structure analysis, IEEE Trans. Knowl. Data Eng. 27 (11) (2015) 3029–3041.
[79] H. Li, X. Wu, Z. Li, W. Ding, Online group feature selection from feature streams, in: Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, 14–18, 2013, Bellevue, Washington, USA., 2013.
[80] J. Yan, B. Zhang, N. Liu, S. Yan, Q. Cheng, W. Fan, Q. Yang, W. Xi, Z. Chen, Effective and efficient dimensionality reduction for large-scale and streaming data preprocessing, IEEE Trans. Knowl. Data Eng. 18 (2) (2006) 320–333.
[81] Y. Tadeuchi, R. Oshima, K. Nishida, K. Yamauchi, T. Omori, Quick online feature selection method for regression -a feature selection method inspired by human behavior-, in: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Montréal, Canada, 7–10 2007, 2007, pp. 1895–1900.
[82] Y. Cai, Y. Sun, J. Li, S. Goodison, Online feature selection algorithm with bayesian l1 regularization, in: Advances in Knowledge Discovery and Data Mining, 13th Pacific-Asia Conference, PAKDD 2009, Bangkok, Thailand, 27–30, 2009, Proceedings, 2009, pp. 401–413.
[83] K. Ooi, T. Ninomiya, Efficient online feature selection based on l1-regularized logistic regression, in: ICAART 2013 - Proceedings of the 5th International Conference on Agents and Artificial Intelligence, Volume 2, Barcelona, Spain, 15–18, 2013, 2013, pp. 277–282.
[84] W. Fan, N. Bouguila, Online learning of a dirichlet process mixture of generalized dirichlet distributions for simultaneous clustering and localized feature selection, in: Proceedings of the 4th Asian Conference on Machine Learning, ACML 2012, Singapore, Singapore, 4–6, 2012, 2012, pp. 113–128.
[85] W. Fan, N. Bouguila, Online variational learning of generalized dirichlet mixture models with feature selection, Neurocomputing 126 (2014) 166–179.
[86] O. Amayri, N. Bouguila, On online high-dimensional spherical data clustering and feature selection, Eng. Appl. AI 26 (4) (2013) 1386–1398.
[87] Z. Yao, W. Liu, Extracting robust distribution using adaptive gaussian mixture model and online feature selection, Neurocomputing 101 (2013) 258–274.
[88] H. Yang, M.R. Lyu, I. King, Efficient online learning for multitask feature selection, Trans. Knowl. Discov. Data 7 (2) (2013) 6.
[89] K. Yu, X. Wu, W. Ding, J. Pei, Towards scalable and accurate online feature selection for big data, in: 2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, 14–17, 2014, 2014, pp. 660–669.
[90] A. Roy, Automated online feature selection and learning from high-dimensional streaming data using an ensemble of kohonen neurons, in: 2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12–17, 2015, 2015, pp. 1–8.
[91] H. Yang, R. Fujimaki, Y. Kusumura, J. Liu, Online feature selection: a limited-memory substitution algorithm and its asynchronous parallel variation, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17, 2016, 2016, pp. 1945–1954.
[92] M. Hammoodi, F.T. Stahl, M. Tennant, Towards online concept drift detection with feature selection for data stream classification, in: ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016), 2016, pp. 1549–1550.
[93] S. Eskandari, M.M. Javidi, Online streaming feature selection using rough sets, Int. J. Approx. Reason. 69 (2016) 35–57.
[94] V. Bolón-Canedo, D. Fernández-Francos, D. Peteiro-Barral, A. Alonso-Betanzos, B. Guijarro-Berdiñas, N. Sánchez-Maroño, A unified pipeline for online feature selection and classification, Expert Syst. Appl. 55 (2016) 532–545.
[95] Y. Yeh, C. Hsu, Online selection of tracking features using adaboost, IEEE Trans. Circuits Syst. Video Technol. 19 (3) (2009) 442–446.
[96] J. Yang, K. Zhang, Q. Liu, Robust object tracking by online fisher discrimination boosting feature selection, Comput. Vis. Image Underst. 153 (2016) 100–108.
[97] K. Yu, W. Ding, X. Wu, LOFS: a library of online streaming feature selection, Knowl.-Based Syst. 113 (2016) 1–3.
[98] I. Jolliffe, Principal Component Analysis, Springer Verlag, 1986.
[99] J. Nie, W. Kotlowski, M.K. Warmuth, Online PCA with optimal regret, J. Mach. Learn. Res. 17 (173) (2016) 1–49.
[100] P. Jain, C. Jin, S.M. Kakade, P. Netrapalli, A. Sidford, Streaming PCA: matching matrix bernstein and near-optimal finite sample guarantees for oja's algorithm, in: Proceedings of the 29th Conference on Learning Theory, COLT 2016, New York, USA, June 23–26, 2016, 2016, pp. 1147–1164.
[101] E. Hazan, S. Kale, M.K. Warmuth, On-line variance minimization in O(n^2) per trial, in: Proceedings of the 23rd Annual Conference on Learning Theory, in: COLT '10, 2010, pp. 314–315.
[102] A.A. Joseph, T. Tokumoto, S. Ozawa, Online feature extraction based on accelerated kernel principal component analysis for data stream, Evol. Syst. 7 (1) (2016) 15–27.
[103] M. Ghashami, D.J. Perry, J.M. Phillips, Streaming kernel principal component analysis, in: Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, 9–11, 2016, 2016, pp. 1365–1374.
[104] L.I. Kuncheva, W.J. Faithfull, PCA feature extraction for change detection in multidimensional unlabelled streaming data, in: Proceedings of the 21st International Conference on Pattern Recognition, ICPR 2012, Tsukuba, Japan, 11–15, 2012, 2012, pp. 1140–1143.
[105] A.A. Qahtan, B. Alharbi, S. Wang, X. Zhang, A pca-based change detection framework for multidimensional data streams: change detection in multidimensional data streams, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, 10–13, 2015, 2015, pp. 935–944.
[106] A. Allahyar, H.S. Yazdi, Online discriminative component analysis feature extraction from stream data with domain knowledge, Intell. Data Anal. 18 (5) (2014) 927–951.
[107] F. Sheikholeslami, D. Berberidis, G.B. Giannakis, Kernel-based low-rank feature extraction on a budget for big data streams, in: 2015 IEEE Global Conference on Signal and Information Processing, GlobalSIP 2015, Orlando, FL, USA, 14–16, 2015, 2015, pp. 928–932.
[108] W. Li, J. Yang, J. Zhang, Uncertain canonical correlation analysis for multi-view feature extraction from uncertain data streams, Neurocomputing 149 (2015) 1337–1347.
[109] T.M. Cover, P.E. Hart, Nearest neighbor pattern classification, IEEE Trans. Inf. Theory 13 (1) (1967) 21–27.
[110] S. García, J. Derrac, J. Cano, F. Herrera, Prototype selection for nearest neighbor classification: taxonomy and empirical study, IEEE Trans. Pattern Anal. Mach. Intell. 34 (3) (2012) 417–435.
[111] D.W. Aha, D. Kibler, M.K. Albert, Instance-based learning algorithms, Mach. Learn. 6 (1) (1991) 37–66.
[112] M. Salganicoff, Density-adaptive learning and forgetting, in: Machine Learning Proceedings 1993, Morgan Kaufmann, 1993, pp. 276–283.

[113] R. Klinkenberg, Learning drifting concepts: example selection vs. example weighting, Intell. Data Anal. 8 (3) (2004) 281–300.
[114] M. Salganicoff, Tolerating concept and sampling shift in lazy learning using prediction error context switching, Artif. Intell. Rev. 11 (1) (1997) 133–155.
[115] J. Beringer, E. Hüllermeier, Efficient instance-based learning on data streams, Intell. Data Anal. 11 (6) (2007) 627–650.
[116] H. Brighton, C. Mellish, Advances in instance selection for instance-based learning algorithms, Data Min. Knowl. Discov. 6 (2) (2002) 153–172.
[117] I. Tomek, Two modifications of CNN, IEEE Trans. Syst., Man, Cybern. 6 (11) (1976) 769–772.
[118] S.J. Delany, P. Cunningham, A. Tsymbal, L. Coyle, A case-based technique for tracking concept drift in spam filtering, Knowl. Based Syst. 18 (45) (2005) 187–195.
[119] B. Smyth, M.T. Keane, Remembering to forget: a competence-preserving case deletion policy for case-based reasoning systems, in: Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1, in: IJCAI'95, 1995, pp. 377–382.
[120] A. Shaker, E. Hüllermeier, Iblstreams: a system for instance-based classification and regression on data streams, Evolv. Syst. 3 (4) (2012) 235–249.
[121] I. Žliobaitė, Combining similarity in time and space for training set formation under concept drift, Intell. Data Anal. 15 (4) (2011) 589–611.
[122] L. Zhao, L. Wang, Q. Xu, Data stream classification with artificial endocrine system, Appl. Intell. 37 (3) (2012) 390–404.
[123] K.B. Dyer, R. Capo, R. Polikar, Compose: a semisupervised learning framework for initially labeled nonstationary streaming data, IEEE Trans. Neural Netw. Learn. Syst. 25 (1) (2014) 12–26.
[124] D. Mena-Torres, J.S. Aguilar-Ruiz, A similarity-based approach for data stream classification, Expert Syst. Appl. 41 (9) (2014) 4224–4234.
[125] Y. Ben-Haim, E. Tom-Tov, A streaming parallel decision tree algorithm, J. Mach. Learn. Res. 11 (2010) 849–872.
[126] A. Gupta, F.X. Zane, Counting inversions in lists, in: Proceedings of the 14th Annual ACM-SIAM Symp. on Discrete Algorithms, 2003, pp. 253–254.
[127] S. Guha, A. McGregor, Stream order and order statistics: quantile estimation in random-order streams, SIAM J. Comput. 38 (5) (2009) 2044–2059.
[128] J. Lu, Y. Yang, G.I. Webb, Incremental discretization for Naïve-bayes classifier, in: Proceedings of the Second International Conference on Advanced Data Mining and Applications, in: ADMA'06, 2006, pp. 223–238.
[129] J. Gama, C. Pinto, Discretization from data streams: applications to histograms and data mining, in: Proceedings of the 2006 ACM Symposium on Applied Computing, in: SAC '06, 2006, pp. 662–667.
[130] P. Lehtinen, M. Saarela, T. Elomaa, Online ChiMerge Algorithm, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 199–216.
[131] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: massive online analysis, J. Mach. Learn. Res. 11 (2010) 1601–1604.
[132] M. Lichman, UCI machine learning repository, 2013, [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.
[133] A. Bifet, R. Kirkby, Data stream mining: a practical approach, Technical Report, The University of Waikato, 2009.
[134] S. García, A. Fernández, J. Luengo, F. Herrera, A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability, Soft Comput. 13 (10) (2009) 959–977.
[135] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm Evolut. Comput. 1 (1) (2011) 3–18.
[136] A. Cano, A. Zafra, S. Ventura, Solving classification problems using genetic programming algorithms on gpus, in: Hybrid Artificial Intelligence Systems, 5th International Conference, HAIS 2010, San Sebastián, Spain, 23–25, 2010. Proceedings, Part II, 2010, pp. 17–26.
[137] S. García, S. Ramírez-Gallego, J. Luengo, J.M. Benítez, F. Herrera, Big data preprocessing: methods and prospects, Big Data Anal. 1 (1) (2016) 9. URL http://dx.doi.org/10.1186/s41044-016-0014-0.
[138] B. Krawczyk, Learning from imbalanced data: open challenges and future directions, Progr. Artif. Intell. 5 (4) (2016) 221–232.
[139] V. López, A. Fernández, S. García, V. Palade, F. Herrera, An insight into classification with imbalanced data: empirical results and current trends on using data intrinsic characteristics, Inf. Sci. 250 (2013) 113–141.
[140] F. Herrera, F. Charte, A.J. Rivera, M.J. del Jesús, Multilabel Classification - Problem Analysis, Metrics and Techniques, Springer, 2016.
[141] F. Herrera, S. Ventura, R. Bello, C. Cornelis, A. Zafra, D.S. Tarragó, S. Vluymans, Multiple Instance Learning - Foundations and Algorithms, Springer, 2016.

Sergio Ramıirez-Gallego received the M.Sc. degree in Computer Science in 2012 from the University of Jaén, Spain. He is currently a Ph.D. student at the Department of Computer Science and Artificial Intelligence, University of Granada, Spain. His research interests include data mining, data preprocessing, big data and cloud computing.

Bartosz Krawczyk is an assistant professor in the Department of Computer Science, Virginia Commonwealth University, Richmond VA, USA, where he heads the Machine Learning and Stream Mining Lab. He obtained his MSc and PhD degrees from Wroclaw University of Science and Technology, Wroclaw, Poland, in 2012 and 2015 respectively. His research is focused on machine learning, data streams, ensemble learning, class imbalance, one-class classifiers, and interdisciplinary applications of these methods. He has authored 35+ international journal papers and 80+ contributions to conferences. Dr Krawczyk was awarded with numerous prestigious awards for his scientific achievements like IEEE Richard Merwin Scholarship and IEEE Outstanding Leadership Award among others. He served as a Guest Editor in four journal special issues and as a chair of ten special session and workshops. He is a member of Program Committee for over 40 international conferences and a reviewer for 30 journals.


Salvador Garcıa received the M.Sc. and Ph.D. degrees in Computer Science from the University of Granada, Granada, Spain, in 2004 and 2008, respectively. He is currently an Associate Professor in the Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain. He has published more than 45 papers in international journals. As edited activities, he has co-edited two special issues in international journals on different Data Mining topics and is a member of the editorial board of the Information Fusion journal. He is a co-author of the book entitled "Data Preprocessing in Data Mining" published in Springer. His research interests include data mining, data preprocessing, data complexity, imbalanced learning, semi-supervised learning, statistical inference, evolutionary algorithms and biometrics.

Michal Wozniak is a professor of computer science at the Department of Systems and Computer Networks, Wroclaw University of Science and Technology, Poland. He received M.Sc. degree in biomedical engineering from the Wroclaw University of Technology in 1992, and Ph.D. and D.Sc. (habilitation) degrees in computer science in 1996 and 2007, respectively, from the same university. In 2015 he was nominated as the professor by President of Poland. His research focuses on compound classification methods, hybrid artificial intelligence and medical informatics. Prof. Wozniak has published over 260 papers and three books. His recent one Hybrid classifiers: Method of Data, Knowledge, and Data Hybridization was published by Springer in 2014. He has been involved in research projects related to the above-mentioned topics and has been a consultant of several commercial projects for well-known Polish companies and public administration. Prof. Wozniak is a senior member of the IEEE.

Francisco Herrera (SM'15) received his M.Sc. in Mathematics in 1988 and Ph.D. in Mathematics in 1991, both from the University of Granada, Spain. He is currently a Professor in the Department of Computer Science and Artificial Intelligence at the University of Granada.
He has been the supervisor of 40 Ph.D. students. He has published more than 300 journal papers that have received more than 49,000 citations (Scholar Google, H-index 112). He is coauthor of the books "Genetic Fuzzy Systems" (World Scientific, 2001) and "Data Preprocessing in Data Mining" (Springer, 2015), "The 2-tuple Linguistic Model. Computing with Words in Decision Making" (Springer, 2015), "Multilabel Classification. Problem analysis, metrics and techniques" (Springer, 2016), "Multiple Instance Learning. Foundations and Algorithms"(Springer, 2016). He currently acts as Editor in Chief of the international journals "Information Fusion" (Elsevier) and "Progress in Artificial Intelligence (Springer). He acts as editorial member of a dozen of journals.
He received the following honors and awards: ECCAI Fellow 2009, IFSA Fellow 2013, 2010 Spanish National Award on Computer Science ARITMEL to the "Spanish Engineer on Computer Science", International Cajastur "Mamdani" Prize for Soft Computing (Fourth Edition, 2010), IEEE Transactions on Fuzzy System Outstanding 2008 and 2012 Paper Award (bestowed in 2011 and 2015 respectively), 2011 Lotfi A. Zadeh Prize Best paper Award of the International Fuzzy Systems Association, 2013 AEPIA Award to a scientific career in Artificial Intelligence, and 2014 XV Andalucía Research Prize Maimónides (by the regional government of Andalucía).
His current research interests include among others, soft computing (including fuzzy modeling and evolutionary algorithms), information fusion, decision making, biometric, data preprocessing, data science and big data.

2.2 Nearest Neighbor Classification for High-Speed Big Data Streams Using Spark

- S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, J. M. Benítez and F. Herrera, Nearest Neighbor Classification for High-Speed Big Data Streams Using Spark. IEEE Transactions on Systems, Man, and Cybernetics: Systems 47 (10) (2017) 2727–2739, doi: 10.1109/TSMC.2017.2700889.
  - Status: **Published**.
  - Impact Factor (JCR 2016): 2.350.
  - Subject Category: Computer Science, Cybernetics. Ranking 7 / 22 (**Q2**).

Nearest Neighbor Classification for High-Speed Big Data Streams Using Spark

Sergio Ramírez-Gallego, Bartosz Krawczyk, Salvador García, Michał Woźniak, *Senior Member, IEEE*, José Manuel Benítez, *Member, IEEE*, and Francisco Herrera, *Senior Member, IEEE*

***Abstract*—Mining massive and high-speed data streams among the main contemporary challenges in machine learning. This calls for methods displaying a high computational efficacy, with ability to continuously update their structure and handle ever-arriving big number of instances. In this paper, we present a new incremental and distributed classifier based on the popular nearest neighbor algorithm, adapted to such a demanding scenario. This method, implemented in Apache Spark, includes a distributed metric-space ordering to perform faster searches. Additionally, we propose an efficient incremental instance selection method for massive data streams that continuously update and remove outdated examples from the case-base. This alleviates the high computational requirements of the original classifier, thus making it suitable for the considered problem. Experimental study conducted on a set of real-life massive data streams proves the usefulness of the proposed solution and shows that we are able to provide the first efficient nearest neighbor solution for high-speed big and streaming data.**

***Index Terms*—Apache Spark, big data, data streams, distributed computing, instance reduction, machine learning, nearest neighbor.**

I. Introduction

THE massive volume of information gathered by contemporary systems became omnipresent, as many research activities require collecting increasingly huge amounts of data. For instance, Large Hadron Collider experiments[1] generates 30 petabytes of information per year. Potential applications for massive data analysis techniques could be found in each human activity domain. Enterprises would like to discover interesting client behavior characteristics, e.g., on the basis of sensor or Internet data. Works on personalized medical treatment for individual patients based on his/her clinical records, such as medical history, genomic, cellular, and environmental data may serve as another example.

We are surrounded by enormous volumes of data arriving continuously from different sources. Therefore, one may say that we are living in the *big data era*. Big data is usually characterized by the so-called 5V's (volume, velocity, variety, veracity, and value), describing its massive volume, dynamic nature, diverse forms, different qualities, and usefulness for human beings [1].

In many cases we do not deal with static data collections, but rather with dynamic ones. They arrive in a form of continuous batches of data, known as data streams [2]. In such scenarios, we need not only to manage the volume but also the velocity of data, thus constantly updating and adapting our learning. To add a further difficulty, many modern data sources generate their outputs with very short intervals, thus creating the issue of high-speed data streams [3].

Massive data must be explored efficiently and converted into valuable knowledge which could be used by enterprises (among others) to build their competitive advantage [4]. However, there exist a considerable gap between contemporary processing and storage capacities, which demonstrates that our ability to capture and store data has far outpaced our ability to process and utilize it. Moore's law says that processing capacity double every 18 months, while disk storage capacity doubles every 9 months (storage law) [5]. This leads to creation of the so-called *data tombs*, i.e., volume of data which are stored but never analyzed. Therefore, we have to develop dedicated tools and techniques which are able to mine enormous volumes of incoming data, while additionally taking into consideration that each record may be analyzed only once to reduce the overall computing costs [6]. MapReduce was the first programming paradigm designed to deal with the phenomenon of big data [7]. Recently, a new large-scale processing framework, called Apache Spark [8], [9], is gaining importance in the big data domain due to its good performance in iterative and incremental procedures.

Lazy learning [10] (also called instance-based learning) is considered as one of the simplest, yet most effective schemes in supervised learning [11]. Here, generalization is deferred

Manuscript received August 18, 2016; revised February 6, 2017; accepted April 18, 2017. Date of publication July 26, 2017; date of current version September 15, 2017. The work of S. Ramírez-Gallego, S. García, J. M. Benítez, and F. Herrera was supported in part by the Spanish National Research Project under Grant TIN2014-57251-P and Grant TIN2016-81113-R, and in part by the Andalusian Research Plan under Grant P11-TIC-7765 and Grant P12-TIC-2958. The work of S. Ramírez-Gallego was supported by the FPU Scholarship from the Spanish Ministry of Education and Science under Grant FPU13/00047. This work was supported by the Polish National Science Center under Grant DEC-2013/09/B/ST6/02264. This paper was recommended by Associate Editor J. Wu. *(Corresponding author: Sergio Ramírez-Gallego.)*

S. Ramírez-Gallego, S. García, J. M. Benítez, and F. Herrera are with the Department of Computer Science and Artificial Intelligence, University of Granada, 18071 Granada, Spain (e-mail: sramirez@decsai.ugr.es; salvagl@decsai.ugr.es; j.m.benitez@decsai.ugr.es; herrera@decsai.ugr.es).

B. Krawczyk is with the Department of Computer Science, Virginia Commonwealth University, Richmond, VA 23284 USA (e-mail: bkrawczyk@vcu.edu).

M. Woźniak is with the Department of Computer Science, Wrocław University of Technology, 50-370 Wrocław, Poland (e-mail: michal.wozniak@pwr.edu.pl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2017.2700889

[1]http://home.cern/about/computing

until a query is made to the case-base. However, as distance between every pair of cases must be computed (quadratic complexity), these methods tends to have much slower classification phase than their counterparts. Furthermore, lazy learners, as k-nearest neighbor (k-NN), tend to accumulate instances from data streams, thus leading to using data related to the outdated concepts may for the decision-making process. As of these reasons lazy learning has not been widely used in streaming environments in spite of its attractive properties.

Data reduction techniques may be applied to improve the performance of lazy learners [12]. Concretely, instance selection techniques can be very effective as they reduce the total number of samples stored in the case-base and therefore simplify the underlying search space. Search speed may also be improved by introducing an implicit metric-space ordering in the case-base [13] or through other techniques as locality-sensitive hashing [14].

In this paper, we propose an efficient nearest neighbor solution to classify high-speed and massive data streams using Apache Spark. Our algorithm consists of a distributed case-base and an instance selection method that enhances its performance and effectiveness. A distributed metric tree has been designed to organize the case-base and consequently to speed up the neighbor searches. This distributed tree consists of a top-tree (in the master node) that routes the searches in the first levels and several leaf nodes (in the slaves nodes) that solve the searches in next levels through a completely parallel scheme. Performance is further improved by a distributed edition-based instance selection method, which only inserts correct examples and removes the noisy ones. Up to the best of our knowledge, this is the first lazy learning solution in dealing with large-scale, high-speed, and streaming problems.

The main contributions of this paper are as follows.

1) Efficient and scalable incremental nearest neighbor classification scheme for massive and high-speed data streams.
2) Smart partitioning of the incoming data streams to parallelize the proposed algorithm using Spark environment.
3) Embedded instance selection method with quickly updated hybrid trees.
4) Comprehensive experimental evaluation of the proposed methods.

Experimental results performed using several datasets and configurations show that our proposal outperforms the same model without edition in terms of accuracy. Our method also reduces the time spent in the prediction stage and the memory consumption.

The structure of this paper is as follows. First, the related works about big data analysis, data stream mining, nearest neighbor and instance selection are presented in Section II. Then the proposed solution for Spark architecture is discussed in Section III. The next section (Section IV) includes results of experimental investigations. Finally, Section V concludes this paper.

II. Related Work

This section will provide necessary background on recent advances in mining massive (Section II-A) and streaming datasets (Section II-B), with special focus put on nearest neighbor-based classification approach (Section II-C).

A. Big Data Analytics

Google designed MapReduce [7] in 2003, which is considered as one of the first distributed frameworks for large-scale data processing. MapReduce allows for automatically processing data in an easy and transparent way through a cluster of computers. The user only needs to implement two operators: 1) Map and 2) Reduce. In the Map phase, the system processes key-value pairs read directly from a distributed file system and transform them into another set of pairs (intermediate results). Each node is in charge of reading and transforming a set of pairs from one or more data partitions. In the Reduce phase, the key coincident pairs are sent to the same node and merged to yield the final result through an user-defined function. For further information about MapReduce and others distributed frameworks, please check [6].

Apache Hadoop [15], [16] is an open-source implementation of MapReduce for reliable, scalable, and distributed computing. Despite its popularity and a number of implemented data mining algorithms [17], [18], Hadoop is not suitable for many scenarios, with emphasis on those where there is a need for explicit data reusage. For instance, online, interactive, and/or iterative computing [19] are affected by this problem.

Apache Spark [8], [9] is a distributed computing platform that became one of the most powerful engines developed for the big data scenario. According to its creators, this platform was designed to overcome the limitations of Hadoop. In fact, the Spark engine has shown to perform faster than Hadoop in many cases (up to 100× in memory). Thanks to its in-memory primitives, Spark is able to load data into memory and query it repeatedly, making it suitable for iterative processes (e.g., machine learning algorithms). In Spark, the driver (the main program) controls multiple workers (slaves) and collects results from them, whereas worker nodes read data blocks (partitions) from a distributed file system, perform some computations and save the result to disk.

Resilient distributed dataset (RDD) is the base structure in Spark, on which the distributed operations are performed. A wide variety of operations are offered by RDDs, such as: filtering, mapping, and joining large data. These operations are designed to transform datasets by locally executing tasks within the data partitions, thus maintaining the data locality. Furthermore, RDDs are a versatile tool that allows programmers to preserve intermediate results (in memory and/or disk) in several formats for reusability purposes, as well as customize the partitioning for data placement optimization.

Spark also allows us to use the RDD's API in streaming environments through the transformation of data streams into small batches. Spark Streaming's design enables the same batch code (formed by RDD transformations) to be used in streaming analytics, without a requirement for significant modifications.

For large-scale data mining, several common learning algorithms and statistic utilities were created and packaged into MLlib [20], [21], the machine learning library of Spark.

This library gives support to a number of knowledge discovery tasks such as: classification, optimization, regression, collaborative filtering, clustering, and data preprocessing.

B. Data Stream Mining

Contemporary machine learning problems are often characterized not only by a significant volume of data, but also by its velocity. Instances may arrive continuously in a form of a potentially unbounded data stream [22]. This poses new challenges for learning algorithms, as they must offer adaptation mechanisms for ever-growing dataset, being able to update their structure in accordance with the current state of a stream [23]. Additionally, new constraints must be taken into consideration that are not present or not so important in static scenarios [24]. Learner must have low response and update times, as new objects must be handled as soon as they become available. Too long processing would cause a delay, as stacking arriving objects would only increase with the stream progress. Furthermore, streaming algorithms must assume limited storage space and memory requirements. One cannot store all of objects from a stream, as data volume will continuously expand [25]. Therefore, objects should be discarded after processing and learner must not require an access to previously seen instances.

Data streams are often characterized by a phenomenon called concept drift [26], [27]. It can be defined as a change of characteristic in incoming data over the course of stream processing.

In streaming environments [2], the incoming objects arrive sequentially, thus data streams can be processed in two different operation modes.

1) Chunk (batch), where data arrive in a form of instance blocks or we collect enough instances to form one.
2) Online, where instances arrive one by one and we must process them as soon as they become available.

There are several possible approaches to learning from data streams.

1) Rebuilding the classifier whenever new data becomes available.
2) Using a sliding window approach.
3) Using an incremental or online learner.

The first of discussed approaches is far from being applicable in a real stream mining environment. Training a new model whenever a new set of instances arrive would impose prohibitive computational costs and excessive need for a storage space in order to accommodate the ever-growing size of the training set. Additionally, during the training process the classifier would be unavailable for data processing, which would lead to a significant time delay. These factors force us to design specialized methods that do not suffer from the mentioned limitations.

Sliding window-based classifiers were designed primarily for drifting data streams, as they incorporate the forgetting mechanism in order to discard irrelevant samples and adapt to appearing changes [28]. Recent works in this area incorporate dynamic window size adjustment [29] or usage of multiple windows [30]. However, we focus on stationary data streams for which proper and continuous model update is of greater importance. Therefore, let us discuss in more details the third group of methods.

Incremental [31] and online [32] learners are such classifiers that are able to continuously update their structure or decision boundaries according to incoming new data [33]. Such methods must meet several requirements, such as processing each object only once during the course of training, having strictly limited memory and time consumption, and their training may be stopped at any time with obtained quality not lower than the one from corresponding classifier trained with the same data in a static mode [34]. Main advantages of such methods lie in their fast and flexible adaptation to new data, as they are not rebuilt from a scratch every time a new instances become available. Additionally, once the object has been processed it can be discarded as it will be of no future use for the classifier. This significantly reduce the requirements for memory and storage space. It is worth noticing that some of popular classifiers can work in incremental or online modes, e.g., Naïve Bayes, neural networks, or nearest neighbor methods. There is also a number of classifiers that have been specifically modified to work with changing streams of instances, like concept-adapting decision trees [35] or very fast decision rules [36].

Nearest neighbor algorithms are highly popular in traditional machine learning, as they offer an easy implementation and a high efficiency. However, due to their lazy learning nature and high computational cost they have not gained significant attention in the domain of data stream analysis [37], [38], especially, when instances arriving with high speed are considered. Let us now review the most popular approaches for speeding-up this classifier.

C. Speeding-Up Nearest Neighbor Searches

The k-NN [39] is an intuitive and effective nonparametric model used in many machine learning problems and can be considered as one of top-ten most influential algorithms in data mining [40]. Nevertheless, k-NN is also a time-consuming method that requires all the training instances to be stored in memory, in order to compute the distance measurement between every pair of instances (quadratic complexity). For this reason, a linear search becomes impractical when large-scale problems are faced and/or new examples are constantly introduced to the case-base.

Many techniques have been proposed to alleviate the k-NN search complexity. They range from metric trees (M-trees) [13], which index data through a metric-space ordering; to locally sensitive hashing [14], which map (with high probability) those elements near in the space to the same bins.

M-tree exploit properties such as the triangle inequality to make searches much more efficient in average, skipping a great amount of comparisons. *M-tree* [41] can be considered as one of the most important and simplest data structure in the space-indexing domain. Let n be a single node in the M-tree, while $n.lc$ and $n.rc$ are, respectively, its left and right children. For each iteration, the algorithm finds two representative points (called pivots) $n.lp$ and $n.rp$ and the decision boundary L that

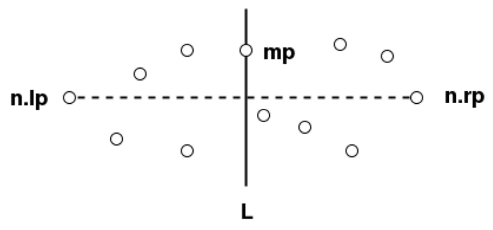


Fig. 1. Partitioning scheme in an M-tree.

goes through the midpoint mp between the set of points to partition. Next, every point to the left of mp is assigned to *n.lc*, and every point to the right to *n.rc* (see Fig. 1). This type of partitioning implies nodes are disjoints and no information is shared between them.

When searching in an M-tree, the algorithm descends through the structure by choosing the nearest node in each level, discarding every node that are not within the searching distance. M-tree may "backtrack" in case that some branches of the tree have remained unpruned. It is done to assure the correctness of the query.

The tedious backtracking process can be skipped if some overlap is allowed between the nodes. *Spill trees* [41] allow overlapping by introducing a new variable called *overlap buffer*. This buffer represents the common area between two overlapping nodes and allows two nodes to have repeated examples. No backtrack is needed in these trees, however, at the cost of introducing potential redundancy. The overlap buffer is estimated by computing the distance (averaged over every instance in the training set) between every example and their nearest neighbors.

The search process in spill trees is thus much more natural, allowing to use *defeatist search* with the aid of the buffer. It uses a direct descend in the tree without backtracking. In practice a combination of spill trees and M-trees can be used making a distinction between overlap (nonbacktracked) and nonoverlap (backtracked) nodes. An example of hybrid structures are *hybrid spill trees* [41].

Most of M-trees are designed to be run sequentially in a single machine and their adaptation to distributed platforms poses a major difficulty. In [42], a distributed version of a metric tree is presented. The authors propose to maintain one top-tree in the master node that route the elements in the first levels. Once the elements have been mapped to the leaves a set of distributed subtrees performs searches in parallel. The idea behind is that the top-tree and the subtrees act like a complete metric tree, but in a fully distributed way. Tornado [43] is another distributed stream processing system that focus on spatio-textual queries. This framework eliminates redundant textual data by using deduplication and fusion of information. In their recent work Maillo *et al.* [44] proposed an efficient k-NN classifier for massive datasets using Apache Spark architecture. The main difference between their proposal and one described here is the nature of analyzed data. Their approach is suitable for massive, yet static datasets and was optimized in order to provide fast calculation of a high number of distances. Our method is suitable for massive, yet streaming data, being able to work in an online mode and with high-speed data streams.

Apart from using special indexing methods, k-NN searches can be speed-up through the application of data preprocessing techniques [12]. These solutions are aimed at reducing the size of datasets in both dimensions (instances and features), while maintaining the original data structure.

Along with feature selection, instance selection is considered as one of the most efficient ways of data reduction. They aim at finding such a reduced dataset that allows to train a model without a performance loss. In many cases, the model can even be more precise due to its simpler structure (Occam's razor principle). Nevertheless, the selection of relevant instances is not a trivial task since a pairwise comparison between each instance must be performed.

Depending on the type of search implemented, instance selection methods could be classified into three categories: 1) condensation (aiming at only retaining boundary points that are close to the borders); 2) edition (aiming at removing noisy boundary points); or 3) hybrid methods (combining the two previous approaches by removing both internal and border points).

Most of instance selection methods described in the literature explicitly or implicitly exploit the k-NN technique to obtain the set of relevant instances [12]. Here, the set of neighbors is being used to decide if the given instance is relevant, redundant, or noisy according to a determined criterion. Classical selection methods as reduced-NN, edited-NN (ENN), or condensed-NN, make use of NN technique to evaluate instances. However, there are many other selection algorithms that use other measures as accuracy, retrieval frequency, or competence. For instance, the iterative case filtering (ICF) method removes those instances whose *reachability* (instances that are correctly solved by a given element) is less than its *coverage* (instances that correctly solved a given element). However, it is worth noticing that ICF launches ENN to remove noisy instances at the beginning.

Relative neighborhood graph edition (RNGE) algorithm [45] is considered as one of the most accurate methods according to the experiments performed in [12]. In RNGE, the neighbors of an instance are determined by a special proximity graph called relative neighborhood graph. Two points are considered as neighbors in the graph if there exist a connecting edge between them. The rule that determines this association is defined as follows: there exist an edge between two given points if there does not exist a third point that is closer to any of them than they are to each other. After building a graph the algorithm removes those instances misclassified by their neighbors (majority voting). RNGE is characterized by a low computational complexity, since the graph can be constructed efficiently in $O(n\log(n))$ time [46].

III. DS-RNGE: Spark-Based Instance Selection Framework for Nearest Neighbor Stream Mining

In this section, we present a lazy learning solution for massive and high-speed data streams. The proposed algorithm (DS-RNGE) consists of a distributed case-base and an instance selection method that enhances its performance and effectiveness. Our case-base is structured using a distributed metric tree, which is entirely maintained in memory to expedite further neighbor queries. Note that our complete scheme is based in-memory primitives from Spark, not only the neighbor queries. The source code of the complete project can be found in: https://github.com/sramirez/spark-IS-streaming.

DS-RNGE proceeds in two phases for each newly arrived batch of data.

1) An edition/update phase aimed at maintaining and enhancing the case-base.
2) A prediction phase that classifies new unlabeled data.

Both phases require fast neighbor queries to accomplish their aims. To deal with this problem, we propose a smart partitioning of the input space in which each subtree queries only a single space partition. This scheme will allow us to parallelize the querying process across the cluster.

A single top-level tree is maintained in the master node to route the elements in the first levels, where the partitioning is still coarse-grained. For each instance, the nearest element in the leaves of the top-tree is returned. The correspondence between leaf nodes and subtrees determines the local tree where the query will be performed (see Section II-C for further details).

As mentioned before, hybrid spill trees use backtracking and redundancy to deal with classification in borders. This implies a cost in time and memory that is far from acceptable in streaming applications. Our idea is to allow classification errors near borders in order to increase the computational efficiency. When the number of elements is much greater than the number of partitions, the number of instances with neighbors in a different partition and the classification error derived from this phenomenon become negligible. Defeatist search is used as reference for our model because of its outstanding performance.

Since an ever-growing and noisy case-base is unacceptable, our approach uses a custom-designed instance selection method. A improved local version of RNGE has been applied to control the insertion and removal of noisy instances. The original method has been redesigned for incremental learning from data streams. For each incoming example, a relative graph is built around the instance and a subset of its neighbors. The local graphs are then used to edit the case-base by deciding what instances should be inserted, removed or left untouched. As every step in this process is performed locally, the communication overhead is negligible.

DS-RNGE manages the following parameters.

1) *nt:* Number of subtrees and number of leaf nodes in the top-tree.
2) *ks:* Number of neighbors used to build the local graphs (instance selection phase).
3) *kp:* Number of neighbors used in the prediction phase.
4) *ro:* Indicates whether removal of examples should be performed or not.

Although edition in our system is guided by a class information, the resulting case-bases can be employed in other learning processes with tangible benefits. For instance, polished case-bases can work with semi-supervised or clustering [47], [48] problems. In general, noise removal should ease learning in other family of classifiers, like decision trees or statistical-based learners. As future work, we will study the effect of case-base edition on other classification methods.

In the following sections, we present the different procedures involved in DS-RNGE. First, we describe the first steps to initialize the distributed case-base (Section III-A). Afterward, the editing/updating process is presented (Section III-B). Here, we present details of insertion and deletion of examples in the tree. Finally, in Section III-C we describe the prediction phase.

A. Initial Partitioning Process

The first step in our system consists of building a distributed metric tree formed by a top-tree (in the master machine) and a set of local trees (in the slave machines). This distributed tree will be queried and updated during next iterations with incoming batches of data. From the first batch we take a sample of *nt* instances to build the main tree. The sampled data should be small enough to fit in a single machine and should maximize the separability between examples to avoid overlapping in the future subtrees. The *nt* parameter is normally set to a value equal to the number of cores in the cluster. By doing so our algorithm is able to fully exploit the maximum level of parallelism in any stage. The routing tree is created following the standard procedure presented in [41], where upper and lower bounds are defined to control the size of nodes.

Once the top-tree is initialized, it is replicated to each machine and one subtree per leaf node is created in the slave machines (see line 7 of the pseudocode). Then, every element in the first batch is inserted in the subtrees by following these steps.

1) For each element, the algorithm searches the nearest leaf node in the top-tree. According to the correspondence between leaf nodes and subtrees we can determine to which subtree each element will be sent. This process is performed in a Map phase.
2) The elements are shuffled to the subtrees according to their keys. Each subtree gets a list of elements to be inserted.
3) For each subtree all received elements are inserted to the tree in a local way. This process is performed in a Reduce phase.

Note that the partitions/subtrees derived from this phase will be maintained during the complete process for reusability purposes, so that only the arriving instances will be moved across the network in each iteration. Algorithm 1 explains this procedure in detail using a MapReduce syntax.

B. Updating Process With Edition

When a new batch of data arrives, we need to start the updating process with edition. This is aimed at inserting new instances, as well as removing those that became redundant over time. At first, the algorithm computes which subtree each element falls into, following the same process described in the previous section. Once all instances are shuffled to subtrees, a local nearest neighbor search for each element is started in corresponding subtrees.

Algorithm 1 Initial Partitioning Process

```
1: INPUT: data, nt
2: // data is the input dataset
3: // nt Number of leaf trees to be distributed across the nodes
4: sample = smartSampling(data)
5: topTree = In the master machine, build the top M-tree
   using sample and the standard partitioning procedure
   explained in [41]. It will be replicated to every slave
   machine.
6: For each leaf node in the topTree, one subtree is created in
   a single slave machine. The resulting set of trees (stored as
   an RDD) is partitioned and cached for further processing.
7: mapReduce $e \in data$
8:     Find the nearest leaf node to e in topTree, and outputs
       a tuple with the tree's ID (key) and e (value). (MAP)
9:     The tuple is sent to the correspondent partition and
       attached to the subtree according to its key. (SHUFFLE)
10:    Combine all the elements with the same key (tree ID)
       by inserting them into the local tree. (REDUCE)
11:    Return the updated tree.
12: end mapReduce
```

After obtaining neighbors the instance selector creates groups, where each one is formed by a new element and its neighbors. Then, local RNGE is applied on each group (as explained in Algorithm 3). The idea behind that is to build a local graph around each group and through this graph to decide what kind of action to perform on each element (insertion, removal, or none). New examples can be inserted or not, whereas old examples (neighbors) can be removed or maintained.

Since each graph has only a narrow view of the case-base, the set of neighbors that can be removed has been limited to those that share an edge with the new element. Removal of old examples can be controlled through the binary parameter *ro*. If activated, the removal may cause a drop in the overall accuracy but the prediction process will run faster because of the reduced size. Note that the insertion process is exact in most of cases since the new elements are in the center of the graph and a suitable number of neighbors (a default value of 10) is enough to find their edges. The number of neighbors for graph construction can be controlled through the *ks* parameter. The greater the value of *ks*, the more precise and the slower is the removal.

Once decisions for each element are taken we perform insertions and removals locally in the subtrees in the same reduce phase. Notice that by doing so the neighbor query and the editing process are both performed in the same MapReduce process, thus reducing the communication overhead. The complete editing process is described in Algorithm 2.

Fig. 2 illustrates all the steps involved in DS-RNGE for one training iteration (one batch). The first part shows how the top-tree is built with two examples: $e1$ and $e2$. Once the main tree is built one subtrees per element in the leaves is created in the slave nodes. Every element is also inserted in its local subtree. In the second phase a new element $e3$ arrives at the top-tree. The top-tree routes the search to the first partition, where the element is sent to perform a neighbor search. This search will allow to decide if the element should be inserted or not. Let suppose that the edition method decides the insertion is suitable according to its nearest neighbor (1-NN) (in this case, $e1$). The insertion is then fully local, as the element has been already sent to the correspondent node and partition. The removal process performs the same operations but removing those cases that do not agree with the edition results. In this case, the decision for $e1$ is to remain unchanged.

Algorithm 2 Updating Process With Edition

```
1: INPUT: query, ks, ro
2: // query is the data to be queried
3: // ks represents the number of neighbors to use in the
   instance selection phase.
4: // ro indicates whether to remove old noisy examples or
   not.
5: mapReduce $e \in data$
6:     Find the nearest leaf node to e in topTree and outputs
       a tuple with the tree's ID (key) and e (value). (MAP)
7:     The tuple is sent to the correspondent subtree according
       to its key. (SHUFFLE)
8:     neighbors = the standard M-tree search process is
       launched for each element in its local subtree in order to
       retrieve the ks-neighbors of e. The output will consist of
       a tuple with e (key) and a list of its ks-neighbors (value).
       (REDUCE)
9:     edited = apply the local RNGE algorithm (Algorithm 3)
       to each tuple in neighbors. The output consist of the
       insertion/removal decision for each element.
10:    if $ro == true$ then
11:        Removed old noisy instances in edited from the tree.
12:    end if
13:    Add new correct instances in edited to the tree.
14:    Return the updated tree.
15: end mapReduce
```

Algorithm 3 Local RNGE

```
1: INPUT: e, ne
2: // e incoming example
3: // ne set of neighbors for e
4: Compute the local RNGE graph using e and ne following
   the procedure detailed in [46].
5: Mark e to be added iff most of its graph neighbors agree
   with its class
6: for $en \in ne$ do
7:     if en is a graph neighbor of e and most of en's graph
       neighbors do not agree with its class then
8:         Mark en to be removed
9:     end if
10: end for
```

Within the edition process, local construction of graphs and subsequent filtering is depicted in Fig. 3. In this graphic a new example from class A (dashed point) arrives to a given partition (Algorithm 2). From the set of points in that partition,

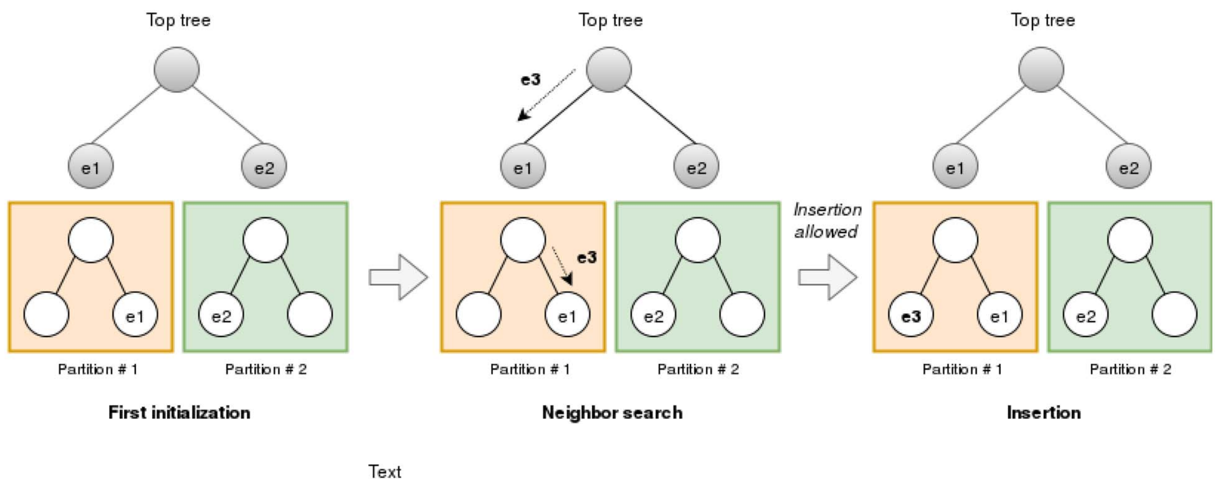


Fig. 2. Flowchart describing initialization, searching, and insertion processes in DS-RNGE.

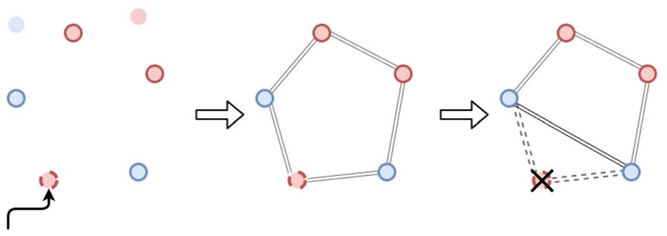


Fig. 3. Local graph edition for each new example. Class A is colored in red and class B in blue.

$ks = 4$-NN (thick circles) are selected from the pool to construct the graph shown in step 2. Note that graphs are built independently from other cases in the partition. Lastly, removal decisions are made according to the connections between neighbors. In our example, the dashed example is not inserted in the case-base since most of its edge-neighbors do not agree with its class. As there are no more agreements between nodes, no additional removals are conducted.

Algorithm 4 Prediction Process

```
1: INPUT: query, kp
2: // query is the data to be queried
3: // kp represents the number of neighbors for predictions.
4: mapReduce e ∈ data
5:     Find the nearest leaf node to e in topTree and outputs a tuple with the tree's ID (key) and e (value). (MAP)
6:     The tuple is sent to the correspondent subtree according to its key. (SHUFFLE)
7:     neighbors = the standard M-tree search process is launched for each element in its local subtree in order to retrieve the ks-neighbors of e. The output will consist of a tuple with e (key) and a list of its ks-neighbors (value). (REDUCE)
8:     For each tuple in neighbors return the most-voted class from the list of neighbors. This value will be the class predicted for the given element.
9: end mapReduce
```

C. Prediction Process

Classification process is an approximate function that is started when new unlabeled data arrive at the system (see Algorithm 4). For each element the algorithm searches for the nearest leaf node in the master node and shuffles the elements to the slave machines. Next, the standard M-tree search process is used to retrieve the *kp*-neighbors of each new element. For each group, formed by a new element and its neighbors, the algorithm predicts the element's class by applying the majority voting scheme to its neighbors. Notice that the query and the prediction are both performed in the same MapReduce phase as in the edition process.

IV. Experimental Study

In order to evaluate the proposed methods, we have designed a thorough experimental study with the following goals in mind.

1) To evaluate the quality and performance of DS-RNGE versus the base model without edition, as well as to check the effect of the batch size on the models (Section IV-B).
2) To check if defeatist search really affects the precision and processing time in tree queries. To do that we perform a comparison between the edited models and the base model without edition (Section IV-C).
3) To validate that our model scales-out correctly by increasing the number of cores available in the cluster (Section IV-D).

A. Experimental Framework

Six large-scale datasets have been used to evaluate the performance and quality of DS-RNGE. Five of them are taken from the UCI Machine Learning Database Repository [49] (poker, susy, hhar,[2] hepmass, and higgs), while another big dataset *ECBDL14*[3] is a highly imbalanced problem

[2]From the Heterogeneity Human Activity Recognition experiment, only the activity recognition dataset was used.

[3]http://cruncher.ncl.ac.uk/bdcomp/

TABLE I
DATASETS USED IN THE EXPERIMENTS. FOR EACH SET, THE NUMBER OF ORIGINAL EXAMPLES (# INST.), THE NUMBER OF UNIQUE EXAMPLES (# UNIQUE), THE TOTAL NUMBER OF ATTRIBUTES (#ATTS.), AND THE NUMBER CLASSES (#CL) ARE SHOWN

Data Set	**# Inst.**	**# Unique**	**#Atts.**	**#Cl.**
poker	1,025,009	1,022,770	10	10
susy	5,000,000	5,000,000	18	2
hhar (25%)	7,535,705	7,535,705	7	7
ecbdl14 (25%)	7,994,298	7,994,298	10	2
hepmass	10,500,000	10,500,000	28	2
higgs	11,000,000	10,721,302	28	2

derived from the data mining competition held under the International Conference GECCO-2014. A random sampling without replacement (25% of the original size) was applied to ECBDL14 and hhar datasets.

In order to transform these static datasets into streams, we randomly partitioned them into equal-sized data batches according to different batch sizes (50 000, 100 000, and 200 000). Before the start of executions, the batches are enqueued. Only one batch per iteration (one second) serves as an input to our system. To prevent the insertion of repeated instances in the M-trees the unique examples from these datasets were extracted and used as the former input for the models. Table I presents the detailed description of the used datasets.

Our evaluation process assumes that DS-RNGE is first tested with the current batch in the queue and then updated with it. This is known as interleaved test-then-train model [50] and allows us to always test our model on unseen examples.

In our experiments three models have been tested. The first one, called *edited*, follows the DS-RNGE scheme presented in Section IV-A, but without allowing the removal of already inserted examples. The second method, called *edited-re*, is another version of DS-RNGE but with removal. And as benchmark method, the same distributed scheme is used but without any type of edition. This scheme, called *orig*, directly inserts elements in the distributed trees and apply the usual prediction process (explained in Algorithm 4). Parameter configuration for all models is described in Table II.

The experiments were performed on a cluster composed of twenty standard nodes (hosting the Spark workers) and one master (hosting the Spark Master) node. The computing nodes have the following features: two processors x Intel Xeon CPU E5-2620 (6 cores/processor, 2.00 GHz, 15 MB cache), 2 TB HDD, and 64 GB RAM. They are connected through a QDR InfiniBand network (40 Gb/s). The following software was used in the experiments: Hadoop 2.5.0-cdh5.3.1 from Cloudera's open-source Apache Hadoop distribution,[4] HDFS replication factor: 2, HDFS default block size: 128 MB, Apache Spark Streaming 1.6, 460 cores (23 CPU cores/node), and 960 RAM GB (48 GB/node). The datasets are hosted in the distributed file system, but they are loaded from memory and cached there before executions start.

[4]http://www.cloudera.com/content/cloudera/en/documentation/cdh5/v5-0-0/CDH5-homepage.html

TABLE II
PARAMETERS OF THE DISTRIBUTED MODELS

Method	Parameters
edited (DS-RNGE)	nt = 420, ks = 10, kp = 1, ro = false
edited-re (DS-RNGE)	nt = 420, ks = 10, kp = 1, ro = true
orig	nt = 420, ks = 0 (no edition), kp = 1, ro = false

TABLE III
AVERAGE RESULTS BY METHOD AND BATCH SIZE (POKER)

Batch Size	Method	Acc.	Tr. time	Cls. time	# Inst. (% red.)	Total time
50,000	edit	**0.5418**	2.47	1.97	330,433 (0.38)	76.65
	edit-re	0.5353	**2.22**	1.93	**81,764 (0.85)**	**71.26**
	orig	0.5351	2.34	**1.69**	537,233 (0.00)	73.61
100,000	edit	**0.5479**	3.67	2.95	369,737 (0.34)	40.05
	edit-re	0.5368	**3.33**	2.90	**86,242 (0.85)**	**38.10**
	orig	0.5380	3.42	**2.67**	563,066 (0.00)	39.77
200,000	edit	**0.5599**	5.52	3.81	445,614 (0.27)	10.82
	edit-re	0.5448	4.85	**3.69**	**92,788 (0.85)**	10.33
	orig	0.5495	**4.48**	4.58	614,467 (0.00)	**9.72**

TABLE IV
AVERAGE RESULTS BY METHOD AND BATCH SIZE (SUSY)

Batch Size	Method	Acc.	Tr. time	Cls. time	# Inst. (% red.)	Total time
50,000	edit	**0.7699**	2.04	1.80	1,967,910 (0.22)	671.381
	edit-re	0.7565	1.80	**1.59**	**1,068,652 (0.58)**	**629.076**
	orig	0.7156	**1.52**	1.78	2,525,658 (0.00)	653.148
100,000	edit	**0.7691**	3.63	3.21	1,999,418 (0.22)	281.50
	edit-re	0.7624	2.60	**2.31**	**1,082,872 (0.58)**	323.40
	orig	0.7160	**1.90**	4.16	2,551,471 (0.00)	**246.67**
200,000	edit	0.7579	5.94	4.75	2,063,179 (0.21)	224.51
	edit-re	**0.7678**	4.96	**3.96**	**1,124,473 (0.57)**	188.68
	orig	0.7164	**2.15**	5.70	2,604,533 (0.00)	**172.17**

Let us now discuss in details the obtained results according to several criteria: the different batch sizes and edition strategies, the search strategy used in M-trees and the scalability of our method.

B. General Comparison: Evaluation of Batch Sizes and Edition Strategies

Detailed results for every dataset, model, and batch size are given in Tables III–VIII. For each combination, several information fields are shown: the batch size, the method used, the average accuracy (Acc.), the average training time (Tr. time), the average classification time (Cls. time), the average number of instances and the percent of reduction in brackets [# Inst. (%)], and the time (in seconds) spent in the whole process. The best result for each column is highlighted in bold.

From these experiments we can state the following conclusions: for every dataset, DS-RNGE without removal (*edit*) is more precise than the other methods, which means that DS-RNGE is useful in enhancing case-bases. DS-RNGE with removal (*edit-re*) only obtains better results than the version without edition (*orig*) in two datasets: 1) *susy* and 2) *ecbdl14*. Competitive results in *ecbdl14* can be explained by the fact that reduction is negligible in this dataset, whereas for *susy*,

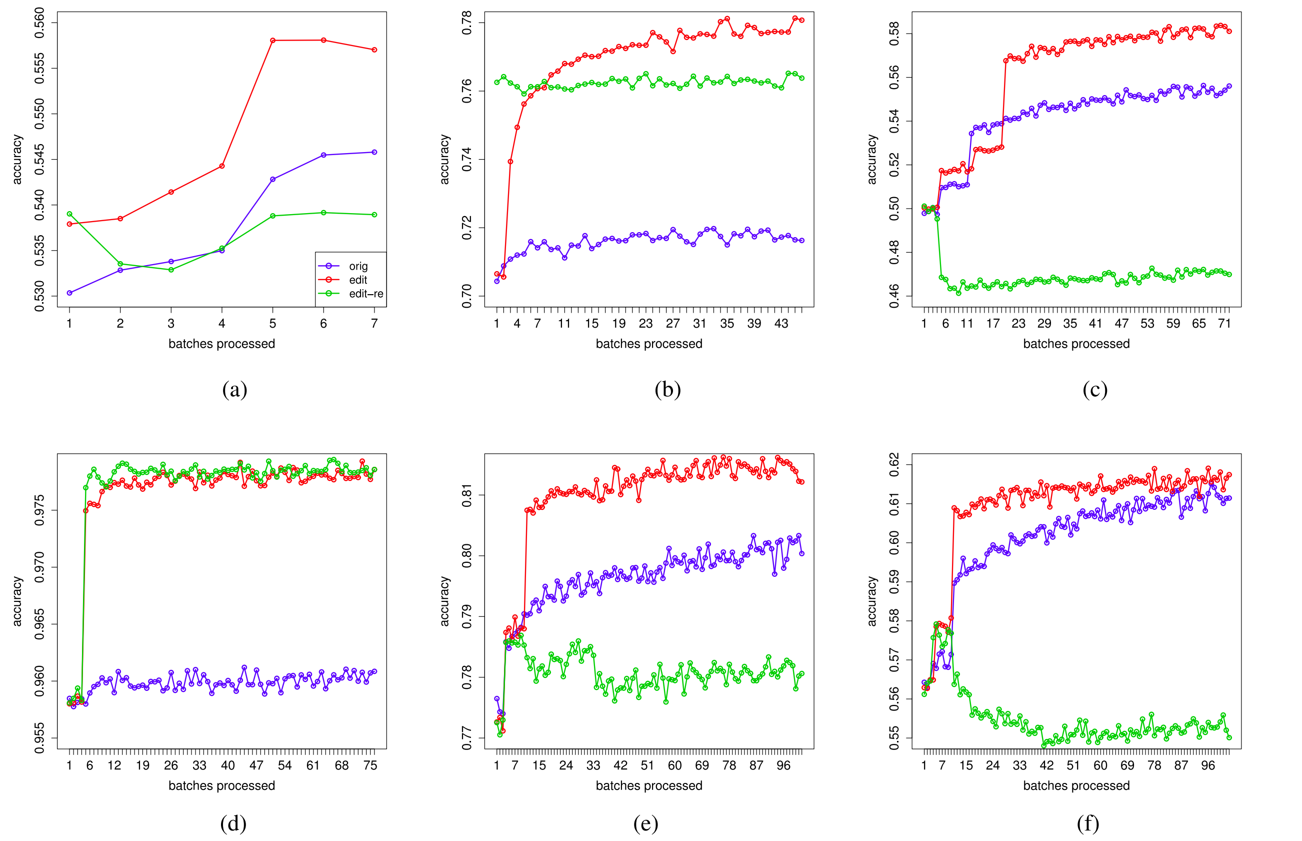


Fig. 4. Accuracies obtained during data stream acquisition according to the number of batches processed. Batch size = 100 000. (a) poker. (b) susy. (c) hhar. (d) ecbdl. (e) hepmass. (f) higgs.

TABLE V
AVERAGE RESULTS BY METHOD AND BATCH SIZE (HHAR)

Batch Size	Method	Acc.	Tr. time	Cls. time	# Inst. (% red.)	Total time
	edit	**0.5595**	1.87	1.56	2,101,474 (0.45)	1,263.19
50,000	edit-re	0.4082	**1.56**	**1.37**	**979,382 (0.74)**	**1,178.99**
	orig	0.5414	1.59	1.77	3,793,085 (0.00)	1,268.03
	edit	**0.5612**	2.81	2.45	2,170,698 (0.43)	617.54
100,000	edit-re	0.4694	2.28	**2.06**	**1,078,336 (0.72)**	**549.50**
	orig	0.5416	**2.05**	2.96	3,818,059 (0.00)	621.15
	edit	**0.5564**	5.29	4.07	2,260,555 (0.42)	364.42
200,000	edit-re	0.4917	4.27	**3.45**	**1,143,478 (0.70)**	310.29
	orig	0.5449	**2.29**	5.13	3,869,666 (0.00)	**307.91**

TABLE VI
AVERAGE RESULTS BY METHOD AND BATCH SIZE (ECBDL14)

Batch Size	Method	Acc.	Tr. time	Cls. time	# Inst. (% red.)	Total time
	edit	0.9769	3.66	2.80	3,935,148 (0.02)	1,854.16
50,000	edit-re	**0.9780**	5.09	3.69	**3,686,600 (0.08)**	2,192.86
	orig	0.9600	**1.67**	**2.69**	4,022,399 (0.00)	**1,573.16**
	edit	0.9767	8.54	6.72	3,960,385 (0.02)	1,384.63
100,000	edit-re	**0.9773**	7.02	5.16	**3,712,289 (0.08)**	1,161.05
	orig	0.9599	**2.00**	**4.25**	4,047,791 (0.00)	**751.88**
	edit	0.9766	9.90	**7.98**	4,011,542 (0.02)	658.41
200,000	edit-re	**0.9785**	13.56	10.11	**3,764,620 (0.08)**	832.44
	orig	0.9600	**2.33**	8.89	4,099,311 (0.00)	**447.95**

TABLE VII
AVERAGE RESULTS BY METHOD AND BATCH SIZE (HEPMASS)

Batch Size	Method	Acc.	Tr. time	Cls. time	# Inst. (% red.)	Total time
	edit	**0.8081**	5.80	5.26	4,273,076 (0.19)	3,781.85
50,000	edit-re	0.7771	5.03	**4.48**	**2,887,314 (0.45)**	**3,432.76**
	orig	0.7957	**1.67**	7.58	5,274,610 (0.00)	3,466.87
	edit	**0.8097**	19.34	18.56	4,316,988 (0.19)	4,222.19
100,000	edit-re	0.7808	9.12	**8.06**	**2,874,770 (0.46)**	**2,171.01**
	orig	0.7963	**2.14**	19.18	5,299,733 (0.00)	2,593.25
	edit	**0.8134**	26.21	25.15	4,392,738 (0.18)	2,471.39
200,000	edit-re	0.7830	15.85	**12.28**	**2,917,591 (0.45)**	**1,394.60**
	orig	0.7968	**2.38**	57.39	5,350,635 (0.00)	2,793.82

it is not clear if it is due to hypo-reduction or other factors. Note that both *susy* and *hepmass* benefits from the same level of reduction, however, reduction is proven to be much more negative in the last dataset. By other factors we mean: incomplete graphs for inserted examples, high dependency between removed instances and their potential incoming neighbors, or high noise presence in data.

The *edit* model is not only precise but also offers highly satisfactory computational time, similar to the original version (the fastest option). In most of the cases, *edit* is characterized by better classification times than *orig* and similar results in total time. When the amount of reduced elements is low, *orig* is faster than *edit* (see Table VI). Nevertheless, when there is a clear reduction (Table VII), *edit* compensates its time-consuming training process with a quicker classification. On the other hand, there is a clear advantage in terms of time on using removal (method *edit-re*) but it fails on accuracy in 4/6 datasets.

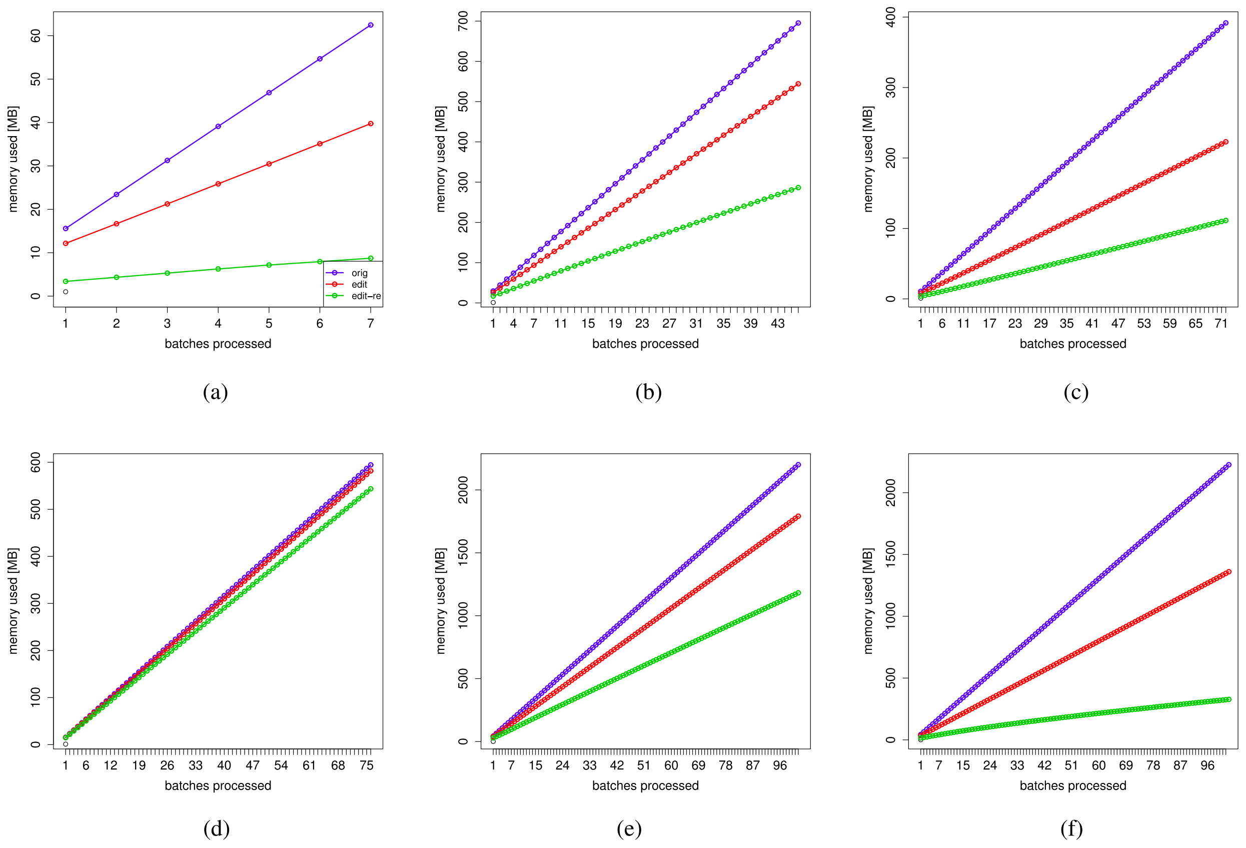


Fig. 5. Memory consumption per batch in megabytes. Batch size = 100 000. (a) poker. (b) susy. (c) hhar. (d) ecbdl. (e) hepmass. (f) higgs.

TABLE VIII
AVERAGE RESULTS BY BATCH SIZE (HIGGS)

Batch Size	Method	Acc.	Tr. time	Cls. time	# Inst.	Total time
50,000	edit	**0.6027**	5.66	5.02	3,241,957 (0.40)	3,823.60
	edit-re	0.5500	1.91	**1.63**	**864,215 (0.84)**	**2,403.12**
	orig	0.6003	**1.78**	8.90	5,385,153 (0.00)	3,857.13
100,000	edit	**0.6097**	11.25	10.30	3,319,026 (0.38)	3,912.90
	edit-re	0.5549	2.74	**2.46**	**884,123 (0.84)**	**1,069.40**
	orig	0.6015	**2.23**	15.50	5,360,330 (0.00)	2,307.00
200,000	edit	**0.6184**	23.50	21.71	3,434,012 (0.37)	2,202.68
	edit-re	0.5573	5.43	**4.92**	**891,232 (0.84)**	**632.12**
	orig	0.6020	**2.58**	41.08	5,461,312 (0.00)	2,161.83

Two hundred thousand elements per second seems to be the best batch size for all datasets. According to the results, it can be stated that the bigger the batch size, the lower the total time and the higher the average time (both classification and training). A bigger batch size implies that less network communication and map/reduce phases are performed. However, as more data is used for initialization with a bigger batch size the average reduction gets lower and the average time results higher.

To illustrate the progress of accuracy in the streaming process, Fig. 4 depicts the individual accuracy values per batch yielded by all models. In general, we can notice that the *edit* model always improves its accuracy over the time. The *orig* one also shows a positive trend, but in most of cases not outperforming *edit* algorithm. This phenomenon is specially remarkable in *susy* and *ecbdl* cases. The *edit-re* model shows a different behavior depending on the amount of instances reduced. For instance, for *susy* and *ecbdl* datasets *edit-re* responds reasonably well, due to a lower reduction. This, however, is not true for the rest of cases.

Fig. 5 depicts the evolution of memory consumption during the course of stream processing. From all the plots we can draw a similar conclusion: applying DS-RNGE is always beneficial, as it leads to significantly reduced memory usage in every case. Only in case of *ecbdl* dataset the reduction can be viewed as a small one. This makes DS-RNGE highly suitable for processing massive data streams, as it displays a reasonable memory consumption allowing for a real-life and real-time implementations. Therefore, we alleviate the prohibitory requirements of standard nearest neighbor techniques, allowing for a resource-efficient stream mining.

C. Distributed Neighbor Search Comparison: Approximate Versus Accurate

In order to show that defeatist search (without backtracking or buffer) in distributed M-trees can also offer competitive results in terms of accuracy and efficiency, we have performed some experiments comparing DS-RNGE with the distributed model proposed in [42].

The same model proposed in Liu's work has been implemented, using a standard M-trees instead of hybrid spill trees as originally proposed by Liu. This change has been introduced to perform a fair comparison between Liu's model and ours, which is entirely based on M-trees. In the modified Liu's

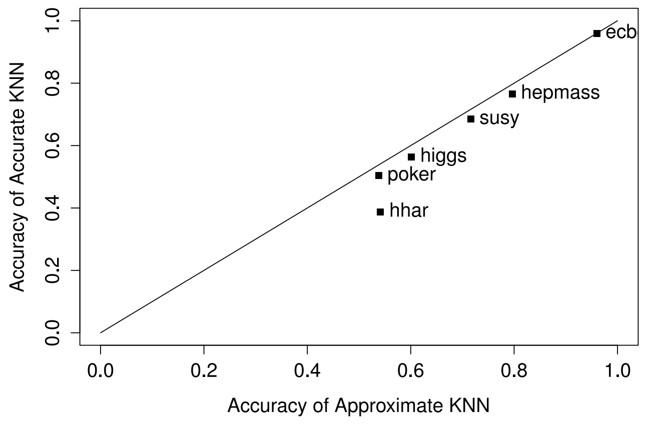


Fig. 6. Distributed neighbor search comparison (approximate versus accurate) in terms of average accuracy (%). Batch size: 200 000 instances/s.

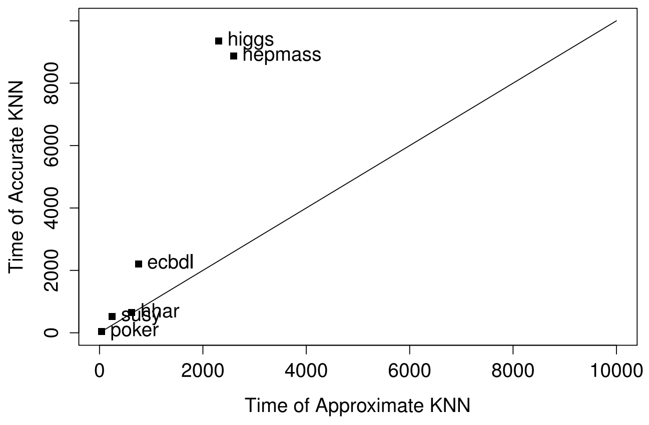


Fig. 7. Distributed neighbor search comparison (approximate versus accurate) in terms of total processing time (seconds). Logarithmic scale.

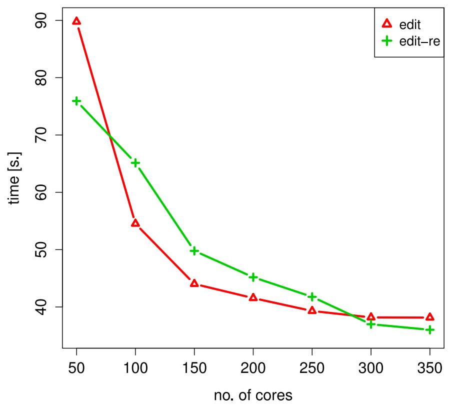


Fig. 8. Scalability study performed by increasing the number of cores available (x-axis). Total time spent by each method (in seconds) is displayed in y-axis.

model, a given node will be explored if the distance between it and the nearest node to the query is in a range, which is determined by the overlap buffer. Liu's model (called *accurate*) can explore more than one branch in its searches, whereas DS-RNGE (called *approximate*) can only explore one.

In Fig. 6, each point compares approximate k-NN with the accurate version on a single dataset in terms of the test accuracy. x-axis represents the accuracy values for approximate k-NN, whereas y-axis for accurate k-NN. Points below $y = x$ line corresponds with datasets for which DS-RNGE performs better. Surprisingly, the results show that DS-RNGE (approximate) is better than the accurate solution for every dataset. These results can be explained by the following fact: in those cases where the neighbors of one element are shared between nodes, it could happened that the "approximate" neighbors better predicts the true class.

Fig. 7 illustrates the same comparison performed in the previous figure, but in terms of efficiency (total time). Here, points above $y = x$ line corresponds with datasets for which DS-RNGE is faster. In this case the results are expected, approximate version always performs faster than the accurate one. Exploring more than one branch is much less efficient than exploring only a single one. In fact some extra map-reduce phases need to be launched in the accurate version in order to merge neighbors from different nodes to obtain the final set of neighbors, which heavily hinders the searching process. This especially affects the two biggest datasets: 1) *higgs* and 2) *hepmass*.

D. Scalability Analysis: Increasing the Number of Cores Available

Fig. 8 shows how the edited models scale-out when the amount of resources available in the cluster is increased. In this experiment the number of cores offered by Spark is gradually increased by 50 in each step using the poker dataset as a reference. A great reduction in time (until 200 cores) can be observed in both versions. From 250 cores, the overhead associated to the distributed scheme (network usage, phase initialization, etc.) starts to equalize the gain obtained by adding additional cores. There is still a time reduction, but the improvement tend to be more stable.

V. Conclusion

In this paper, we have presented DS-RNGE, a nearest neighbor classification solution for processing massive and high-speed data streams using Apache Spark. Up to our knowledge, DS-RNGE is the first lazy learning solution designed for large-scale, high-speed, and streaming problems. Our model organizes the instances by using a distributed metric tree, consisting of a top-level tree that routes the queries to the leaf nodes and a set of distributed subtrees that performs the searches in parallel. DS-RNGE includes an instance selection technique that constantly improves the performance and effectiveness of the learner by only allowing the insertion of correct examples and removing outdated ones. As all phases in DS-RNGE perform the computations locally, our system is able to quickly respond to the continuous stream of data.

The experimental analysis shows that DS-RNGE combines high accuracy with significantly reduced processing time and memory consumption. This allows for an resource-efficient mining of massive dynamic data collections. DS-RNGE without removal overcomes in terms of precision the base model without edition in any case. DS-RNGE yields better time results in the prediction phase, whereas its competitor performs faster in updating the case-base. In general, both algorithms have similar performance, if we measure the total time spent in both phases.

Our future work will concentrate on adding a condensation technique in order to control the ever-growing size of the case-base over time. By removing redundancy, the time cost derived from edition will be alleviated, while at the same time maintaining the original effectiveness. Additionally, we plan extend our approach to drifting data streams and propose time and memory efficient solutions for rebuilding the model as soon as the change occurs. We plan to tackle this challenge by extending our model with drift detection module, as well as by using instance weighting with forgetting to allow for smooth adaptation to changes. Additionally, we envision modifications of our algorithm that will make it suitable for mining massive and imbalanced data streams [51].

References

[1] V. Mayer-Schönberger and K. Cukier, *Big Data: A Revolution That Will Transform How We Live, Work and Think*. London, U.K.: John Murray, 2013.

[2] J. Gama, *Knowledge Discovery From Data Streams*. Boca Raton, FL, USA: Chapman & Hall, 2010.

[3] D. Han, C. G. Giraud-Carrier, and S. Li, "Efficient mining of high-speed uncertain data streams," *Appl. Intell.*, vol. 43, no. 4, pp. 773–785, 2015.

[4] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.

[5] U. Fayyad and R. Uthurusamy, "Evolving data into mining solutions for insights," *Commun. ACM*, vol. 45, no. 8, pp. 28–31, Aug. 2002. [Online]. Available: http://doi.acm.org/10.1145/545151.545174

[6] A. Fernández *et al.*, "Big data with cloud computing: An insight on the computing environment, mapreduce, and programming frameworks," *Wiley Interdiscipl. Rev. Data Min. Knowl. Disc.*, vol. 4, no. 5, pp. 380–409, 2014.

[7] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proc. OSDI*, San Francisco, CA, USA, 2004, pp. 137–150.

[8] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning Spark: Lightning-Fast Big Data Analytics*. Sebastopol, CA, USA: O'Reilly Media, 2015.

[9] Apache Spark: Lightning-Fast Cluster Computing. (2017). *Apache Spark*. [Online]. Accessed on Jan. 2017. [Online]. Available: https://spark.apache.org/

[10] D. Aha, *Lazy Learning*. Dordrecht, The Netherlands, Kluwer, 1997.

[11] C. C. Aggarwal, *Data Mining: The Textbook*. Cham, Switzerland: Springer, 2015.

[12] S. García, J. Luengo, and F. Herrera, *Data Preprocessing in Data Mining*. Cham, Switzerland: Springer, 2014.

[13] H. Samet, *Foundations of Multidimensional and Metric Data Structures* (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling). San Francisco, CA, USA: Morgan Kaufmann, 2005.

[14] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. 25th Int. Conf. Very Large Data Bases (VLDB)*, Edinburgh, U.K., 1999, pp. 518–529.

[15] T. White, *Hadoop, the Definitive Guide*. Sebastopol, CA, USA: O'Reilly Media, 2012.

[16] Apache Hadoop Project. (2017). *Apache Hadoop*. [Online]. Accessed on Jan. 2017. [Online]. Available: http://hadoop.apache.org/

[17] W.-P. Ding, C.-T. Lin, M. Prasad, S.-B. Chen, and Z.-J. Guan, "Attribute equilibrium dominance reduction accelerator (DCCAEDR) based on distributed coevolutionary cloud and its application in medical records," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 3, pp. 384–400, Mar. 2016.

[18] Y. Xun, J. Zhang, and X. Qin, "FiDoop: Parallel mining of frequent itemsets using MapReduce," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 3, pp. 313–325, Mar. 2016.

[19] J. Lin, "Mapreduce is good enough? If all you have is a hammer, throw away everything that's not a nail!" *Big Data*, vol. 1, no. 1, pp. 28–37, 2012.

[20] X. Meng *et al.*, "Mllib: Machine learning in apache spark," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, 2016.

[21] A. Spark. *Machine Learning Library (MLlib) for Spark*. Accessed on Jan. 2017. [Online]. Available: http://spark.apache.org/docs/latest/mllib-guide.html

[22] M. M. Gaber, "Advances in data stream mining," *Wiley Interdiscipl. Rev. Data Min. Knowl. Disc.*, vol. 2, no. 1, pp. 79–85, 2012.

[23] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: A survey," *Inf. Fusion*, vol. 37, pp. 132–156, Sep. 2017.

[24] A. Bifet, G. D. F. Morales, J. Read, G. Holmes, and B. Pfahringer, "Efficient online evaluation of big data stream classifiers," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, Sydney, NSW, Australia, 2015, pp. 59–68.

[25] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, and F. Herrera, "A survey on data preprocessing for data stream mining: Current status and future directions," *Neurocomputing*, vol. 239, pp. 39–57, May 2017.

[26] J. Gama, I. Žliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surveys*, vol. 46, no. 4, pp. 1–37, 2014.

[27] C. Alippi, D. Liu, D. Zhao, and L. Bu, "Detecting and reacting to changes in sensing units: The active classifier case," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 44, no. 3, pp. 353–362, Mar. 2014.

[28] Z. Pervaiz, A. Ghafoor, and W. G. Aref, "Precision-bounded access control using sliding-window query views for privacy-preserving data streams," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 7, pp. 1992–2004, Jul. 2015.

[29] L. Du, Q. Song, and X. Jia, "Detecting concept drift: An information entropy based method using an adaptive sliding window," *Intell. Data Anal.*, vol. 18, no. 3, pp. 337–364, 2014.

[30] O. Mimran and A. Even, "Data stream mining with multiple sliding windows for continuous prediction," in *Proc. 22nd Eur. Conf. Inf. Syst. (ECIS)*, Tel Aviv, Israel, 2014, pp. 1–15.

[31] J. Read, A. Bifet, B. Pfahringer, and G. Holmes, "Batch-incremental versus instance-incremental learning in dynamic and evolving data," in *Proc. 11th Int. Symp. Adv. Intell. Data Anal. (IDA)*, Helsinki, Finland, 2012, pp. 313–323.

[32] P. M. Domingos and G. Hulten, "A general framework for mining massive data streams," *J. Comput. Graph. Stat.*, vol. 12, no. 4, pp. 945–949, 2003.

[33] M. Woźniak, "A hybrid decision tree training method using data streams," *Knowl. Inf. Syst.*, vol. 29, no. 2, pp. 335–347, 2011.

[34] H. He, S. Chen, K. Li, and X. Xu, "Incremental learning from stream data," *IEEE Trans. Neural Netw.*, vol. 22, no. 12, pp. 1901–1914, Dec. 2011.

[35] G. Hulten, L. Spencer, and P. M. Domingos, "Mining time-changing data streams," in *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, San Francisco, CA, USA, 2001, pp. 97–106.

[36] P. Kosina and J. Gama, "Very fast decision rules for classification in data streams," *Data Min. Knowl. Disc.*, vol. 29, no. 1, pp. 168–202, 2015.

[37] L. I. Kuncheva and J. S. Sánchez, "Nearest neighbour classifiers for streaming data with delayed labelling," in *Proc. 8th IEEE Int. Conf. Data Min. (ICDM)*, Pisa, Italy, 2008, pp. 869–874.

[38] D. Yang, E. A. Rundensteiner, and M. O. Ward, "Mining neighbor-based patterns in data streams," *Inf. Syst.*, vol. 38, no. 3, pp. 331–350, 2013.

[39] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. 13, no. 1, pp. 21–27, Jan. 1967.

[40] X. Wu and V. Kumar, Eds., *The Top Ten Algorithms in Data Mining* (Data Mining and Knowledge Discovery). Boca Raton, FL, USA: Chapman & Hall, 2009.

[41] T. Liu, A. W. Moore, A. G. Gray, and K. Yang, "An investigation of practical approximate nearest neighbor algorithms," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, Vancouver, BC, Canada, 2004, pp. 825–832.

[42] T. Liu, C. Rosenberg, and H. A. Rowley, "Clustering billions of images with large scale nearest neighbor search," in *Proc. IEEE Workshop Appl. Comput. Vis. (WACV)*, Austin, TX, USA, 2007, p. 28.

[43] A. R. Mahmood *et al.*, "Tornado: A distributed spatio-textual stream processing system," in *Proc. 41st Int. Conf. Very Large Data Bases*, vol. 8. pp. 2020–2023, 2015.

[44] J. Maillo, S. Ramírez, I. Triguero, and F. Herrera, "kNN-IS: An iterative spark-based design of the k-nearest neighbors classifier for big data," *Knowl. Based Syst.*, vol. 117, pp. 3–15, Feb. 2017.

[45] J. S. Sánchez, F. Pla, and F. J. Ferri, "Prototype selection for the nearest neighbour rule through proximity graphs," *Pattern Recognit. Lett.*, vol. 18, no. 6, pp. 507–513, 1997.

[46] K. J. Supowit, "The relative neighborhood graph, with an application to minimum spanning trees," *J. ACM*, vol. 30, no. 3, pp. 428–448, 1983.

[47] M. Du, S. Ding, and H. Jia, "Study on density peaks clustering based on k-nearest neighbors and principal component analysis," *Knowl. Based Syst.*, vol. 99, pp. 135–145, May 2016.

[48] H. Jia, S. Ding, and M. Du, "Self-tuning p-spectral clustering based on shared nearest neighbors," *Cogn. Comput.*, vol. 7, no. 5, pp. 622–632, 2015.

[49] K. Bache and M. Lichman. (2013). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml

[50] A. Bifet, G. Holmes, R. Kirby, and B. Pfahringer. (2011). *Data Stream Mining: A Practical Approach*. [Online]. Available: http://moa.cms.waikato.ac.nz/publications/

[51] B. Krawczyk, "Learning from imbalanced data: Open challenges and future directions," *Progr. Artif. Intell.*, vol. 5, no. 4, pp. 221–232, 2016.

Sergio Ramírez-Gallego received the M.Sc. degree in computer science from the University of Jaén, Spain, in 2012. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain.

His current research interests include data mining, data preprocessing, big data, and cloud computing.

Bartosz Krawczyk received the M.Sc. and Ph.D. (both with Distinctions) degrees from the Wroclaw University of Science and Technology, Wroclaw, Poland, in 2012 and 2015, respectively.

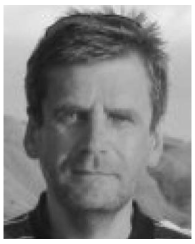
He is an Assistant Professor with the Department of Computer Science, Virginia Commonwealth University, Richmond VA, USA, where he heads the Machine Learning and Stream Mining Laboratory. He has authored over 35 international journal papers and over 80 contributions to conferences. His current research interests include machine learning, data streams, ensemble learning, class imbalance, one-class classifiers, and interdisciplinary applications of these methods.

Dr. Krawczyk was a recipient of numerous prestigious awards for his scientific achievements like IEEE Richard Merwin Scholarship and IEEE Outstanding Leadership Award among others. He served as a Guest Editor in four journal special issues and as the Chair of ten special session and workshops. He is a Program Committee Member for over 40 international conferences and a Reviewer for 30 journals.

Salvador García received the M.Sc. and Ph.D. degrees in computer science from the University of Granada, Granada, Spain, in 2004 and 2008, respectively.

He is currently an Associate Professor with the Department of Computer Science and Artificial Intelligence, University of Granada. He has published over 45 papers in international journals. As edited activities, he has co-edited two special issues in international journals on different Data Mining topics. He has co-authored the book entitled *Data Preprocessing in Data Mining*Ì (Springer). His current research interests include data mining, data preprocessing, data complexity, imbalanced learning, semi-supervised learning, statistical inference, evolutionary algorithms, and biometrics.

Dr. García is an editorial board member of the *Information Fusion* journal.



Michał Woźniak (SM'10) received the M.Sc. degree in biomedical engineering and the Ph.D. and D.Sc. (habilitation) degrees in computer science from the Wrocław University of Technology, Wrocław, Poland, in 1992, 1996, and 2007, respectively.

He is a Professor of computer science with the Department of Systems and Computer Networks, Wrocław University of Science and Technology. His current research interests include compound classification methods, hybrid artificial intelligence, and medical informatics. He has published over 260 papers and three books including the book entitled *Hybrid Classifiers: Method of Data, Knowledge, and Data Hybridization* (Springer, 2014). He has been involved in research projects related to the above-mentioned topics.

Dr. Woźniak was nominated as the Professor by the President of Poland, in 2015. He has been a Consultant of several commercial projects for well-known Polish companies and public administration.

José Manuel Benítez (M'98) received the M.S. and Ph.D. degrees in computer science from the University of Granada, Granada, Spain.

He is currently an Associate Professor with the Department of Computer Science and Artificial Intelligence, University of Granada. He has published in journals such as the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, *Fuzzy Sets and Systems*, the *Journal of Statistical Software*, *Information Sciences*, *Mathware and Soft Computing*, *Neural Networks*, *Applied Intelligence*, the *Journal of Intelligent Information Systems*, *Artificial Intelligence in Medicine*, *Expert Systems with Applications*, *Computer Methods and Programs in Biomedicine*, *Evolutionary Intelligence*, and *Econometric Reviews*. His current research interests include time series analysis and modeling, distributed/parallel computational intelligence, data mining, and statistical learning theory.

Francisco Herrera (SM'15) received the M.Sc. and the Ph.D. degrees from the University of Granada, Granada, Spain, in 1988 and 1991, respectively, both in mathematics.

He is currently a Professor with the Department of Computer Science and Artificial Intelligence, University of Granada. He has been the Supervisor of 38 Ph.D. students. He has published over 300 journal papers that have received over 44 000 Scholar Google Citations with an H-index of 109. He has co-authored books entitled *Genetic Fuzzy Systems* (World Scientific, 2001), *Data Preprocessing in Data Mining* (Springer, 2015), and *The 2-tuple Linguistic Model. Computing With Words in Decision Making* (Springer, 2015), and *Multilabel Classification. Problem Analysis, Metrics and Techniques* (Springer, 2016). His current research interests include soft computing (including fuzzy modeling and evolutionary algorithms), information fusion, decision making, bibliometrics, biometric, data preprocessing, data science, and big data.

Dr. Herrera was a recipient of many awards and honors, such as, the ECCAI Fellow 2009, the IFSA Fellow 2013, the IEEE TRANSACTIONS ON FUZZY SYSTEM Outstanding 2008 and 2012 Paper Award (bestowed in 2011 and 2015), the 2011 Lotfi A. Zadeh Prize Best paper Award of the IFSA Association, and nominated as Highly Cited Researcher in the areas of Engineering and Computer Sciences. He currently acts as the Editor-in-Chief of the international journals *Information Fusion* (Elsevier) and *Progress in Artificial Intelligence* (Springer). He acts as an editorial member of a dozen journals.

2.3 Online Entropy-Based Discretization for Data Streaming Classification

- S. Ramírez-Gallego, S. García, and F. Herrera, Online Entropy-Based Discretization for Data Streaming Classification. Future Generation Computer Systems.
  - Status: **Submitted**.

Manuscript Details

Manuscript number	FGCS_2017_2088
Title	Online Entropy-Based Discretization for Data Streaming Classification
Article type	Full Length Article

Abstract

Data quality is deemed as determinant in the knowledge extraction process. Low-quality data normally imply low-quality models and decisions. Discretization, as part of data preprocessing, is considered one of the most relevant techniques for improving data quality. In static discretization, output intervals are generated at once, and maintained along the whole process. However, many contemporary problems demands rapid approaches capable of self-adapting their discretization schemes to an ever-changing nature. Other major issues for stream-based discretization such as interval definition, labeling or how is implemented the interaction between learning and discretization components are also discussed in this paper. In order to address all the aforementioned problems, we propose a novel, online and self-adaptive discretization solution for streaming classification which aims at reducing the negative impact of fluctuations in evolving intervals. Experiments with a long list of standard streaming datasets and discretizers have demonstrated that our proposal performs significantly more accurately than the other alternatives. In addition, our scheme is able to leverage from class information without incurring in an overweight cost, being ranked as one of the most rapid supervised options.

Keywords	Data stream; Concept drift; Data preprocessing; Data reduction; Discretization; Online learning
Corresponding Author	Sergio Ramírez Gallego
Corresponding Author's Institution	Department of Computer Science and Artificial Intelligence, University of Granada
Order of Authors	Sergio Ramírez Gallego, Salvador Garcia, Francisco Herrera
Suggested reviewers	Leandro Minku, Bartosz Krawczyk

Submission Files Included in this PDF

File Name [File Type]

highligs_FCGS.docx [Highlights]

2017-sramirez-online-v3.pdf [Manuscript File]

biographies_authors.doc [Author Biography]

Submission Files Not Included in this PDF

File Name [File Type]

authors_photos.zip [Author Photo]

To view all the submission files, including those not included in the PDF, click on the manuscript title on your EVISE Homepage, then click 'Download zip file'.

- We propose LOFD, an online, self-adaptive and locally-applied discretization algorithm for streaming classification.
- A key feature of the model is that it smoothly and efficiently adapts its limits minimizing the the negative impact of fluctuations.
- Interval labeling and interaction problems in data streaming are analyzed.
- Close interaction between discretizers and learners is addressed by providing two exhaustive solutions.
- The model is compared to the start-of-the art algorithms from different perspectives, such as real-world problems to different simulated drift scenarios.

Online Entropy-Based Discretization for Data Streaming Classification

S. Ramírez-Gallego[a,*], S. García[a], F. Herrera[a,b]

[a]*Department of Computer Science and Artificial Intelligence, CITIC-UGR, University of Granada, 18071 Granada, Spain.*
[b]*Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia.*

Abstract

Data quality is deemed as determinant in the knowledge extraction process. Low-quality data normally imply low-quality models and decisions. Discretization, as part of data preprocessing, is considered one of the most relevant techniques for improving data quality.

In static discretization, output intervals are generated at once, and maintained along the whole process. However, many contemporary problems demands rapid approaches capable of self-adapting their discretization schemes to an ever-changing nature. Other major issues for stream-based discretization such as interval definition, labeling or how is implemented the interaction between learning and discretization components are also discussed in this paper.

In order to address all the aforementioned problems, we propose a novel, online and self-adaptive discretization solution for streaming classification which aims at reducing the negative impact of fluctuations in evolving intervals. Experiments with a long list of standard streaming datasets and discretizers have demonstrated that our proposal performs significantly more accurately than the other alternatives. In addition, our scheme is able to leverage from class information without incurring in an overweight cost, being ranked as one of the most rapid supervised options.

Keywords: Data stream, Concept drift, Data preprocessing, Data reduction, Discretization, Online learning

1. Introduction

Learning models and subsequent results are highly dependent on the quality of input data. Incorrect decisions can be taken if raw data are not properly cleaned and structured. The data preprocessing task [1, 2] is an essential step

[*]Corresponding author
Email addresses: `sramirez@decsai.ugr.es` (S. Ramírez-Gallego), `salvagl@decsai.ugr.es` (S. García), `herrera@decsai.ugr.es` (F. Herrera)

in data mining, which aims at transforming raw data extracted from databases to polished datasets. This goal is achieved by removing negative factors inherent in data, such as: noise, missing values, meaningless or redundant data. Data reduction is a family of preprocessing techniques that focuses on obtaining a reduced representation of data, at the same maintaining its original structure. This can be done, for example, by selecting the most informative features or instances, or by simplifying the feature space.

Data discretization follows a reduction strategy which converts complex continuous attributes into a finite set of discrete intervals. Discretization has recently become very popular in the data science community [3, 1, 4], mainly due to the need of many learning algorithms for discrete values. For instance, standard implementations of decision rules [5] or Naïve Bayes [6][7] (NB) only admit categorical data in their processes. Even though other methods do not explicitly require discrete values, many of them benefit from simplified spaces [8]. In general, discrete data usually convey faster learning processes and more precise models, thus following the Occam's razor principle.

Standard discretization algorithms require the entire dataset to be in memory as a preliminary requirement. However, an increasing number of current problems in industry (sensors, logs, etc.) output continuous data in form of batches or individual instances (online) [9]. These unbounded and dynamic data [10] (data streams) demand novel learning schemes that not only adapt well, but also that constantly revise their time and memory requirements [11, 12]. The ideal scenario is that in which instances are processed once, and then discarded. Another requirement to face is the likely non-stationary of incoming data (concept drift) [13]. Sudden or abrupt changes in data distribution [14] require outstanding adaptation abilities to follow drifting movements in decision borders.

Online discretization [15] also suffers from concept drift as data distribution is strongly connected with evolving intervals. Ideally, discrete intervals should adapt as smooth as possible to drifts in order to avoid significant drops in accuracy. Also adjustments should not imply complex rebuilding processes, but they should be solved rapidly. Up to date, few supervised approaches for online discretization have been presented in the literature. Despite relevant, these proposals tend to accomplish abrupt and imprecise splits [15], or they are too costly for streaming systems.

How interval labels are defined and labeled by online discretizers, or what type of discrete information is passed to online learners are other open problems that have received even less attention in the literature. Any minor alteration in the meaning and/or the definition of discrete intervals means a certain subsequent drop in learning accuracy. As shown in [15], the standard labeling technique inherited from the static environment is unable to cope with these questions, and it shows a deficient behavior in this new paradigm. Hence novel and improved schemes that explicitly address the interval labeling and interaction problems are required in the streaming field.

The aim of this work is to tackle the previous problems by developing a new solution that smoothly and efficiently adapts to incoming drifts. Our method, henceforth called Local Online Fusion Discretizer (LOFD), mainly relies on highly-informative class statistics to generate accurate intervals at every step. Furthermore, local nature of operations implemented in LOFD offers low response times, thereby making it suitable for high-speed streaming systems. Finally, we detail two alternatives that can be used by online discretizers to effectively improve interaction between the discretization and learning phases. The first approach naturally provides reliable histogram information to some learners, whereas the second one is a renovated version of the standard scheme which is valid for all learners. The improvements introduced here aim at minimizing the drawbacks associated to the dynamic relabeling and interaction phenomenons, described in Section 2.3.

Our approach will be evaluated using a thorough experimental framework, which includes a list of 12 streaming datasets, two online learning algorithms, and the state-of-the-art for online discretization presented in [15]. A thorough analysis based on non-parametric and Bayesian statistical tests is performed to assess quality in the results. Additionally, a study concerning the impact of the novel relabeling approaches, and a case study are also included for illustrative purposes.

The remaining paper is structured as follows. Section 2 introduces the discretization topic from two different perspectives: its formal definition and its adaptation to the online environment. Section 3 describes throughly the solution proposed, highlighting the main contributions introduced to solve the problem. Section 4 presents the results obtained and the subsequent analysis. Finally, some relevant conclusions are provided in Section 5.

2. Background

In this section we detail the discretization problem and some related concepts such as the use of border points as an optimization. Then the problem of discretizing streaming data is presented, as well as a list of related methods presented in the literature. Lastly, the issue of interval definition in the online environment is thoroughly analyzed.

2.1. Discretization: related concepts and ideas

Discretization is a data reduction technique that aims at projecting a set of continuous values into a discrete and finite space [3, 16]. Let D refer to a labeled dataset formed by a set of instances N, a set of attributes M, and a set of classes C. All training instances are labeled with a label from C. A discretization algorithm would generate a set of disjoint intervals for all continuous attribute $A \in M$. The discretization scheme generated I_A consists of a set of cutpoints which define the limits for each interval:

$$I_A = \{\forall g_i \in Dom(A) : g_1 < g_2 < \ldots < g_k\}, \tag{1}$$

where I_A is the discretization scheme for A, and g_1 and g_k, are the inferior and superior limit for A, respectively. Notice that the original scheme considers all distinct points in A at the start, where $k \leq |N|$.

As a preliminary approach to interval labeling, we can associate each interval with the same index as g_{i-1} forming the interval set $I = \{I_{A1}, I_{A2}, \ldots, I_{Ak}\}$, $|I| = k - 1$. Labeling process (also called indexing) will determine how intervals are retrieved in the subsequent learning process. Following the previous description, we can move to define the membership of continuous points to a given interval I_{Aj} as follows:

$$\forall p \in Dom(A), \{\exists j = \{1, 2, \ldots, k\} \mid p \in I_{Aj} \iff g_{j-1} < p \leq g_j\}. \tag{2}$$

For simplification purposes, g_{j-1} value for each attribute can be removed so that intervals are uniquely defined by their g_j. The attribute is upperly bounded by g_k.

Supervised discretization problem (as described above) is a NP-complete [17] optimization problem whose search space consists of all candidate cut points for each $A \in M$, namely, all distinct points in the input dataset considering each attribute independently. This initial space can be simplified by only considering those points on the borders, which are known to be optimal according to several measures in the literature [18]. This improvement will let us to achieve better class separability, as well as significant savings in complexity. For a deeper analysis of border points, please refer to [19]. Formally, a **boundary point** can be defined as any point $y \in A$ between two elements $v, z \in A$ with different classes, such that $v < y < z$, and $\nexists w \in A \mid v < w < z$.

Among the wide set of evaluation measures that benefit from the inclusion of boundaries, those based on entropy are distinguished by its outleading results in discretization. For instance, FUSINTER [20], which integrates quadratic entropy in its evaluations, has proven to be one of the most flexible and competitive discretizers according to [16]. In each iteration, FUSINTER fuses those adjacent intervals whose merging would most improve the aggregated criterion value, defined for each interval as follows:

$$\mu(I_{A\beta}) = \sum_{j=1}^{|C|} \alpha \frac{c_{+j}}{|N|} \Big(\sum_{i=1}^{|C|} \frac{c_i + \lambda}{c_{+j} + |C|\lambda} \Big(1 - \frac{c_i + \lambda}{c_{+j} + |C|\lambda} \Big) \Big) + (1 - \alpha) \frac{|C|\lambda}{c_{+j}}, \tag{3}$$

where c_i represents the number of elements in $I_{A\beta}$ for a given class, c_{+j} the total amount of elements contained in $I_{A\beta}$, and α and λ are two control factors.

2.2. Online discretization for data streams

Data streaming describes the scenario in which instances arrive sequentially in form of unbounded instances or batches [21]. Standard algorithms are not originally designed to cope with unbounded data since they typically assume that the entire training set is available beforehand. New algorithms, capable of constantly updating their structure, are then required [22] in this streaming scenario.

Among the full set of features and problems to be considered in data streaming, we highlight here those more relevant for discretization. Firstly, unbounded streams impose a restricted limit to memory storage. Given that we do not have prior knowledge about the length of streams, it is not possible to just save all incoming objects as the stream might be infinite. Ideally, algorithms should be constrained to a single access to each instance. Afterwards, accessed instances should be removed to meet the memory requirement. Additionally, novel instances must be processed as soon as possible to avoid delays in the prediction and update phases.

Dynamic systems can also be affected by the changes in data distribution introduced by new data [23]. This phenomenon, called concept drift, is well-categorized and described in the literature [14]. Depending on the severity of these modifications, drifts can be classified as: sudden (rapid changes), gradual (smooth changes), incremental (uptrend), recurring (repetitive changes), or blips (sudden peaks).

Several learning strategies have been captured in the literature to overcome the concept drift problem. Explicitly, algorithms can leverage from an external drift detector [24] to detect and rebuild the model whenever a drift appears. On the other hand, learners can hold a self-adaptive strategy based on sliding windows, ensembles [22], or online learners [25] –build the model incrementally with each novel instance–.

The emergence of drifts in dynamic environments poses a major challenge for online discretizers [15], which must adjust their intervals properly over time. Interval adaptation should be as smooth as possible, at the same time reducing as well its time consumption.

Early proposals on online discretization usually follow an unsupervised approach, which defines a preset number of intervals in advance. Some proposals compute quantile points (equal-frequency) in an approximate or exact manner. The quantiles then serve to delimit further intervals. One of the most relevant proposals in quantile-based discretization is the Incremental Discretization Algorithm (IDA) [26] algorithm. IDA employs a reservoir sample to track data distribution and quantiles. Generated intervals are grouped in interval heaps, an efficient data structure to retrieve min-max information in constant time. Equal-width discretization is another alternative that uniquely demands the number of bins the attribute will be split.

In contrast to unsupervised discretizers, class-guided algorithms do not impose a constant number of intervals, but they alternate splits and merges to generate more informative cutpoints [19].Few proposals in the literature address online discretization from a supervised perspective. PiD [27] was the first proposal to leverage from class information in its model based on two layers. The former one produces preliminary cutpoints by summarizing data, whereas the latter one performs class-guided merges on the previous splits. Recurrent updates in the first layer are performed whenever a primal interval is above its size. In counterpart, the second layer uses a parametric approach to merge candidates. PiD has been criticized [26] by several reasons: firstly, the correspondence between layers dilute as time passes (see Section 3); secondly, high skewness in data might provoke a dramatic increment in intervals; and finally, repetitive values might also undermine the performance due to the generation of different intervals with common points.

In [28], the authors developed an online version of ChiMerge (OC) that offers identical results to those claimed by the original proposal. OC relies on a sliding window technique as well as several complex structures to emulate original ChiMerge. Nevertheless, the complexity of the data structures introduced conveys a barely acceptable cost in time and memory requirements.

2.3. Interval definition, labeling and interaction in the streaming scenario.

Evolving nature of discretization in streaming contexts poses a major challenge for the close interaction existing between supervised discretizers and learners. One-step definition in static discretization plainly ignores this problem by assuming no further modifications in the set of cutpoints. However, one-step definition is constantly threatened by the never-ceasing arrival of unseen data in this context. This unpleasant situation not only hinders the discretizer's ability to partition the continuous space, but also the subsequent interaction with the learning operator. As an illustrative example, suppose a logistic regression algorithm relying entirely on literal labels [26] to learn patterns, which obtain deteriorated and imprecise information after each discretization step. At a given cost, the algorithm is forced to discard some already learned patterns, whereas others should be incorporated in order to maintain the algorithm's performance. This section aims at addressing all the aforementioned problems.

In the literature, we can find two alike strategies for **interval labeling** originally designed for static discretization: one based on directly assuming cutpoints as labels (i.e.: values lower than point 2.5), and another one based on literal human-readable terms, usually set by experts (i.e.: < 2.5 is replaced by "low" income). **Interval definition** is usually composed by the set of cut points for each feature. *Cut-point based intervals* represent a quite versatile option as it do not require expert labeling. However, this strategy can be considered as less appropriate for dynamic learning because cut-points constantly move across the feature space. The previous scheme might be replaced by *explicit labels* that would allow points vary freely. Nevertheless, learners that rely in literal labels are known to suffer from a natural drop in accuracy because many learners directly rely on them to generalize. Additionally, new labels appear, and some old ones disappear as a consequence of natural discretization evolution.

Although explicit labeling suffer from definition drift, the *cut-points based strategy* –as defined by [18]– can be directly stated as outdated in streaming contexts. This is mainly due to intervals maintain class consistency by constantly shifting their limits (and hence their labels). To illustrate this problematic, suppose an scenario where all cut points in an scheme I_A in time t are slightly displaced in time $t+1$, for example when a new element is inserted in the lowest bin in IDA. In this case the online discretizer is forced to update the label of each equidistant bins, therefore, a completely new scheme is generated. This abrupt change will cause that some previous information is lost, and the current model remains outdated. The previous issue justifies the adoption of explicit labels to track intervals, and relegates cut points to a secondary role (exclusively for definition).

It is important to distinguish between how the **interval definition and labeling** tasks are accomplished, and how information is transferred between the discretizer phase and the subsequent learning one (**interval interaction**). Most of times labels also act as bridge between phases beyond as a explicit denomination for intervals. However, there exist some special situations where labeling and interaction differ. For instance, the algorithm in [29] relies on cut-points to define its boundaries but provides updated class histograms as interaction unit. From these discrete statistics, it is possible to derive the conditional likelihood that an attribute-value belongs to an interval given its belonging to a class: $P(I_Aj|Class)$. This scheme is called *statistic-based discretization*, and does not require labels explicitly. In this scenario, labeling loses its importance as long as appropriate counts are provided to the classifier. However, this scheme is barely applicable to other algorithms beyond bayesian learners.

Explicit labeling also entails a range of major issues in online learning, such as: abrupt changes in the original definition (label swap, label creation), or constant transfer of instances between bins (instance relabeling). All of them deeply affect the underlying interaction between discrete values and the learning phase, and imply a negative impact on the model performance as shown in [15]. This is specially remarkable in algorithms that uniquely rely on labels, such as linear gradient-based, or rule-based algorithms. Furthermore, not only the meaning is susceptible to alteration, but also the number of intervals is altered. Linear classifiers, for instance, require the number of intervals and their meaning to remain invariant over time [26]. In this paper we propose further improvements to the explicit labeling scheme (Section 3) which enables the application of this approach in streaming contexts. The aim here is to reduce as much as possible the number of intervals and instances affected by relabeling.

In order to illustrate the previous problem, we propose an example where a given interval (numeric label i) is split into two new intervals. This causes the amount of original intervals and labels to augment, and a new scheme to be generated. If the right resulting interval is deemed as the new interval, the new definition forces to interval i to borrow the label from the old interval $i+1$. This process is repeated sequentially from $i+1$ interval till the last one, which is labeled as $|I| = 4$.

Figure 1 depicts a toy example on how the labeling of intervals evolves over time after a split occurs under the standard scheme. The topmost row represents the shape of intervals in time t. After a new cut point appears, interval I_2 is split into two new intervals. It causes that intervals I_2 and I_3 need to re-adapt their labels. In the middle row, the right partition (I_3) will borrow the label from next interval, and the rightmost one will receive a new label I_4. Our alternative (**smooth shift**) is represented by the bottommost row in which only the right split changes. In this case, only one interval (I_4) is affected, whereas the other intervals remains invariant. This scheme will be further explained in Section 3.

Despite current discretizers have solved properly the problem of dynamically discerning between irrelevant and useful intervals in the online context, the

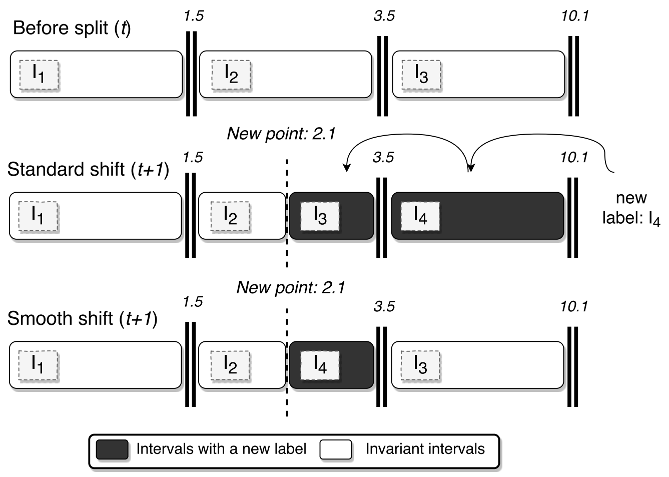


Figure 1: Evolution of intervals before and after a split: a comparison of labeling techniques. *Number in squares corresponds with interval labels, black boxes with the intervals affected by the split, and numbers on vertical lines with the threshold values delimiting the intervals.* The second row represents the standard labeling approach inherited from static discretizers, whereas the third row depicts the smooth labeling scheme.

interval labeling and interaction problems have received little attention in the literature. Only PiD [27] explicitly addresses them by providing a solution that offers free accurate histograms to the attached classifier (NB), similar to that in [29]. As accounting is directly performed by PiD, interval labels do not play a key role anymore, and they can be deemed as disposable. Criticisms to PiD are focused on the fact that the previous scheme does not represent a general solution for standard classifiers, being only valid for those based on histogram-based information.

Other methods in the literature, such as IDA or OC, do not explicitly address the interval definition problem, but they directly assume the standard scheme based on interval labels inherited from static discretization. This scheme normally consists of a set of ordinal discrete values (positive integers), characters, or even cut points. As explained before, this scheme is outdated and entails many drawbacks.

3. LOFD: An Online Entropy-Based Discretizer

In this section we present LOFD, an online, local, supervised, bottom-up discretizer [16] which smoothly adapts its online discretization scheme through a set of accurate histograms. LOFD is a entirely local and self-adaptive that apply local merges and splits whenever a new boundary point appears. By default LOFD relies on the smooth strategy (Section 2.3) to tackle the labeling problem, however, it can be configured to provide likelihood information if required. As evaluation measure, quadratic entropy (Equation 3) is used to evaluate the fitness quality of potential merges.

Firstly, Section 3.1 provides two alike perspectives to address the interval labeling and interaction problems; both included in LOFD. Section 3.2 explains the other features included in LOFD, as well as description about split and merge operations.

3.1. Interval labeling in LOFD: smooth shifting

As mentioned before, online discretizers tend to adopt the standard discretization scheme by default. In this scheme any change in the shape or number of intervals is solved by creating a new scheme whose relationship with the previous one will determine how the prediction model will perform in further steps. The most common idea is to label intervals by attaching an ordinal list of integers to the interval set defined by points. However, plenty of label movements between intervals arise as the training progresses. Our idea is to break the requirement of using ordinal labels, and to replace them by unrelated labels which minimize the number of intervals. In this new scheme, henceforth called **smooth shift**, the splits will be solved by attaching a new label to the minority partition (see Figure 1). In case of merges, the interval will adopt the label from the larger partition in terms of number of instances. In both cases, the remaining intervals do not vary which considerably reduces the impact of evolving discretization.

Beyond providing smoother transitions, “smooth shifting” is completely valid for any classifier that admits categorical variables. Some algorithms (random forest) natively works with categorical variables, whereas others (logistic regression) assumes an implicit order in values that in the case of categorical terms is erroneously imposed. This problem can be easily solved by introducing a binary encoding (one-hot) that equally separates labels in the feature space.

For those learners that can leverage from statistics, LOFD also offers an scheme similar to that presented in PiD (*statistic-based labeling*). Throughout the maintenance of augmented histograms, LOFD provides free likelihood information to NB. In this scenario, intervals are defined and ordered by cut points, and labels are directly ignored. For each incoming test value, the interval bounding the point will be retrieved, and the required information provided. This direct interaction avoids revisiting and swapping interval labels, which makes both the discretization and learning processes more lightweight.

3.2. The LOFD algorithm

In this section we present the strategy implemented by LOFD to adapt its scheme over time, as well as other relevant features and improvements introduced and outlined below:

- **Highly-informative splits**: in online environments, it is complex and costly to track real distribution of points given that algorithms are constrained to certain memory bound. However, most of discretization decisions heavily depend on statistical measures that require accurate information. For example, those based on entropy or mutual information.

Thus, we can assert that the more accurately intervals track distributions, the wiser decisions will be applied.

In LOFD, and for each interval, we build an independent **memory-constrained** histogram that accurately tracks distributions. This model differs from PiD in that the latter one suffers from a weak correspondence between layers which makes the histograms imprecise in most cases. LOFD histograms are only limited by memory requirements though. By imposing size limits, we can adjust the trade-off between performance and accuracy according to our requirements.

- **Bi-directional discretization**: LOFD proposes to consider both splits and merges since both insertions and removals of points are considered and relevant for the streaming scenario. Natural fluctuations in our scheme will be addressed by considering both actions, thus increasing the competitiveness of output solutions. Whereas merges are naturally applied, splits are much more complex since they demand accurately conserved distributions.

- **Extended merges**: local changes in intervals may cause a previous adjacent merge becomes positive. In order to improve the performance of merges, we propose to evaluate all potential combinations among the novel interval and their adjacent intervals (see Algorithm 1). For splits far from the extremes, four intervals must be considered: the two splits and their neighbors.

3.2.1. Main process: instance-level

The main procedure in LOFD is explained here. Firstly, discrete intervals are initialized following the static process defined in FUSINTER [20] (line 6). Discretization is performed on the first batch of elements, formed by *iniTh* instances. From this point, LOFD updates the scheme of intervals in each iteration, and for each single attribute *att*.

For each new single point *val*, LOFD retrieves its bounding interval (*ceiling*) from I_A (line 10), which is internally implemented as a red-black tree[1]. If the point is over the upper feature limit (lines 19-23), LOFD will generate a new interval at this point, being *val* the new maximum for attribute *att*. A merge between the old last interval and the new one is also evaluated by computing the quadratic entropy value for the potential merge. If merged entropy is lower than the sum of parts, the change will be accepted.

If *ceiling* exists, *val* is inserted in the histogram in *ceiling* (also a red-black tree). Using the previous histogram, we check if *val* is a boundary point or not (Section 2.1). If affirmative (lines 14-17), *ceiling* is split into two parts with *val* as midpoint. Afterwards, different merge combinations are evaluated between the resulting intervals, and their neighbors (as will be explained in Section 3.2.2). The resulting set will be inserted in I_A.

[1]`https://en.wikipedia.org/wiki/Red-black_tree`

Finally, each point is added to the timestamped queue to ensure further removals in case of histograms overflow (histogram size $\geq$ *maxHist*). This condition is checked on the entire set of intervals in lines 26-30. If needed, LOFD retrieves points from the queue in ascending order (by age), and removes these points from histograms until enough space for further points is available in interval *int*. This mechanism is mainly used to avoid heavy searches in overpopulated histograms. Further memory control can be programmed by adding a parameter that limit the growth of the timestamped queue. This would help us to avoid memory overflow in scenarios where splits occur continuously.

Algorithm 1 LOFD algorithm

```
 1: INPUT: $D$, initTh, maxHist
 2: // $D$ is the input dataset.
 3: // $initTh$ Number of instances before initializing intervals
 4: // $maxHist$ Maximum number of elements in interval histograms
 5: $I$ = On the first batch ($i = 1 \ldots initTh$), apply the static discretization
    process explained in [20].
 6: for $i = initTh + 1 \rightarrow N$ do
 7:   for $A \in M$ do
 8:     $ceil$ = retrieve the ceiling interval that contains $D_{iA}$
 9:     if $ceil \neq null$ then
10:       $isBound$ = check if $D_{iA}$ is boundary
11:       Insert $D_{iA}$ into $ceil$ and update its criterion
12:       if $isBound == true$ then
13:         $(ceil, new)$ = split $ceil$ into two intervals with $D_{iA}$ as cutpoint
14:         Evaluate local merges between $ceil$, $new$, and the surrounding in-
            tervals until no improvement is achieved.
15:         Insert the resulting set into $I_A$
16:       end if
17:     else
18:       $last$ = Create a new interval on the right side with $D_{iA}$ as upper
          limit
19:       Insert $last$ into $I_A$
20:       Evaluate merge with the old maximum interval
21:     end if
22:   end for
23:   add $D_i$ to the timestamped queue
24:   for $int \in I_A$ do
25:     if $|histogram(int)| > maxHist$ then
26:       Remove old points from the timestamped queue, and subsequently,
          from the local histograms until $|histogram(int)| <= maxHist$. Re-
          move empty intervals.
27:     end if
28:   end for
29: end for
```

Complexity is mainly determined by boundary evaluation $O(|M| \cdot log(maxHist))$, and the split/merge process $O(|M| \cdot maxHist)$ which requires to fetch the whole inner histogram[2]. In either case, the trade-off between performance and effectiveness can ultimately be controlled through $maxHist$. Hence, shorter histograms will imply less accurate decisions, but more agile evaluations.

3.2.2. Merge and splits: interval-level

Figure 2 depicts a simplified scheme of the **split** process, which occurs whenever a new boundary point is processed. The new boundary point (2.2) introduced causes interval I_2 to be separated into two intervals. Interval I_2 now contains those points from the histogram lower or equal than 2.2, and preserves the same label because it contains more elements than the new interval I_4. Larger intervals keep their original label in order to reduce as much as possible the effect of relabeling in inner points. The new interval receives label I_4 (the next integer unseen), and those points higher than the cutpoint (2.2).

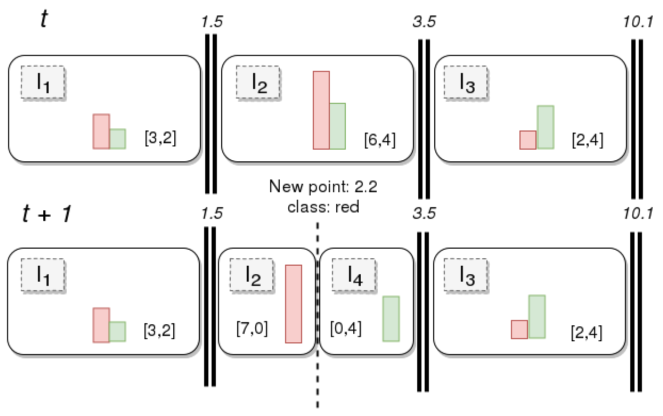


Figure 2: Flowchart describing a split in LOFD (smooth labeling) with three original intervals (first row) and their histograms. A new point (2.2) is introduced, generating a split and a new interval I_4. In LOFD, splits are performed whenever a new boundary point is processed. *Number in squares corresponds with interval labels, number in brackets class distribution, and vertical lines with the cut points considered.*

Whenever a new split/interval is generated, the **merge** process is launched on the divided intervals and their neighbor intervals. Merge process can be deemed as the opposite of split since it basically consists of the fusion of class counters and inner histograms. From the set of potential combinations, that merge action, which more strongly reduces entropy (according Equation 3), will be applied. The previous process will be repeated recursively until no more merges are available or convenient. Note that a merge will be performed iff the

[2]The logarithmic contribution of interval searches has been removed from this formula since $|I_A|$ tends to be negligible (see Section 4).

quadratic entropy of the resulting interval is lower than the sum of both parts. Notice also that merge is responsible of mixed histograms formed by boundary points from different classes in intervals I_1 and I_3.

This procedure avoids recomputing values for intact intervals (the vast majority), so that only one interval will require to re-evaluate its entropy and potential merges in each iteration.

4. Experiments and case study

This section provides a comparative analysis between our proposal and the other state-of-the-art discretizers. As LOFD offers two alternatives for interval labeling, two independent sections are issued here. In Section 4.2, LOFD adopts smooth shifting whereas the rest of discretizers assume standard labeling to interact with NB. In Section 4.3 LOFD and PiD directly interact with NB through histograms. Finally, a case study is included to illustrate the effect of concept drift on evolving discretization intervals.

4.1. Experimental framework

Here we outline all the details related to our experiments, such as: datasets processed, parameters involved and their values, validation scheme, etc. Evaluation has been performed in terms of prediction ability (accuracy), evaluation time (spent on discretizing and prediction), and reduction ability on continuous features (# discrete intervals).

Table 1 shows some basic information about data. Half of the datasets were artificially created by the Massive Online Analysis (MOA) tool [30], ranging from sudden drift to different types of gradual drifts. For more details about the parameter setting accomplished to generate these datasets, please refer to our code repository[3]. The remaining datasets were collected from the official MOA's webpage, except for *kddcup_10* that was processed and generated by Dr. Gama[4].

In order to evaluate the performance of LOFD several state-of-the-art discretizers have been included in this framework for comparison purposes. They range from supervised (OC and PiD) to unsupervised schemes (IDA - window-based version). All described methods have been thoroughly analyzed and categorized in [15]. Gaussian Naïve Bayes (GB) has been elected as the reference classifier for our tests, because of the reasons exposed in Section 2. For the remaining methods, Gaussian estimation is replaced by the discrete intervals generated. Alternatively, Hoeffding Tree (HT) [31] is incorporated to test the discretization effect of our solution on other learning models. Table 2 shows the parameters involved in the experiments, as well as the values set according to the authors' criteria.

[3]`https://github.com/sramirez/MOAReduction`

[4]`http://www.liaad.up.pt/kdus/products/datasets-for-concept-drift`

Table 1: Basic information concerning datasets. For each row, # total instances (#Inst.), total number of attributes (#Atts.) (the number of numerical (#Num.) and nominal (#Nom.)), and the number of output labels (#Cl), are depicted.

Data Set	**#Inst.**	**#Atts.**	**#Num.**	**#Nom.**	**#Cl.**
airlines	539,383	6	3	3	2
elecNormNew	45,311	8	7	1	2
kddcup_10	494,020	41	39	2	2
poker-lsn	829,201	10	5	5	10
covtypeNorm	581,011	54	10	44	7
blips	500,000	20	20	0	4
sudden_drift	500,000	3	3	0	2
gradual_drift	500,000	3	3	0	2
gradual_recurring_drift	500,000	20	20	0	4
incremental_fast	500,000	10	10	0	4
incremental_slow	500,000	10	10	0	4

Table 2: Description of parameters. Default values (in bold) are shown in the first row. Unless specified, these values are common to all methods.

Method	**Parameters**
Discretization	**initial elements = 100, window size = 1 (default)**
Gaussian Bayes, w/o disc. (GB)	–
Gaussian Hoeffding Tree Bayes (10 splits), w/o disc. (HT)	–
Online ChiMerge (OC) [28]	–
Proportional Incremental Disc. (PiD) [27]	$\alpha = 0.75$, initial bins = 500, update layer #2 = 10,000, min/max = 0/1
Local Online Fusion Disc. (LOFD)	max. size by histogram = 10,000, initial elements = 5,000

The evaluation is performed following an standard evaluation technique for online learning, called **interleaved test-then-train** [32]. In this scheme, incoming examples are first evaluated by the current model. Afterwards, the examples are incorporated to the model in the training phase. Note that this technique is more appropriate for streaming environments than hold-out evaluation.

All experiments has been executed in a single commodity machine with the following characteristics: 2 processors Intel Core i7 CPU 930 (4 cores / 8 threads, 2.8 GHz, 8 MB cache), 24 GB DDR2 RAM, 1 TB HDD (3 Gb/s), Ethernet network, CentOS 6.4 (Linux). All algorithms, included our new approach, have been packaged in an extension library for MOA (16.04v)[5]. All the experiments have been launched in MOA.

4.2. Analytical comparative: smooth shift vs. static labeling

This section focuses on studying the effect of LOFD discretizer with smooth labeling in online learning, as well as comparing our solution with other alternatives which utilizes standard labeling.

[5] `http://moa.cms.waikato.ac.nz/moa-extensions/`

Table 3: Classification test accuracy on discretized data (Naïve Bayes)

	PiD	**IDA**	**OC**	**GB**	**LOFD**
airlines	63.0057	64.1563	65.0723	64.5504	**65.0868**
elecNormNew	71.9522	76.6905	74.0731	73.3625	**77.1517**
kddcup_10	99.1474	98.4644	98.1404	97.1908	**99.2901**
poker-lsn	55.0335	59.4337	58.5465	59.5528	**69.3981**
covtypeNorm	66.6306	62.7235	64.2254	60.5208	**69.2387**
blips	74.5680	66.4494	64.2148	60.9060	**76.3668**
sudden_drift	65.7736	81.3168	77.8808	**83.8144**	83.5752
gradual_drift_med	60.8404	82.8908	80.1032	**84.7000**	84.2794
gradual_recurring_drift	65.1678	58.5250	58.5612	56.7450	**67.9446**
incremental_fast	73.9900	75.6472	75.6036	76.3642	**80.7308**
incremental_slow	65.6074	76.9186	75.4316	78.0688	**81.6210**
MEAN	69.2470	73.0197	71.9866	72.3432	**77.6985**

4.2.1. Accuracy results (predictive ability)

Table 3 contains average accuracy results as a summary of the entire learning process performed by Naive Bayes. From these outcomes we can outline the following conclusions:

- There exists a clear advantage on using LOFD against other alternatives. LOFD is on average 5% more precise than its closest competitor (IDA), which means 2.5×10^3 more instances correctly classified[6].
- Up to now, supervised alternatives have generated worse solutions than those presented by unsupervised approaches, despite the former ones leverage from class information. On the contrary, LOFD utilizes this knowledge to overcome the previous problem and thus generates better schemes.
- LOFD overcomes its competitors in 9/11 datasets, with similar results in the other datasets. This fact proves the superiority of LOFD, and its great versatility for both real and artificial datasets, as well as for different type of trends and drifts.

To reaffirm the positive results obtained by LOFD a thorough statistical analysis is performed on accuracy results throughout two non-parametric tests [33]. Table 4 reports the results for Wilcoxon Signed-Ranks Test (1vs1) and Friedman-Holm test (1vsN) with a significance level $\alpha = 0.05$. Both tests assert LOFD clearly outperforms the other options, and no method is close in performance to it (no ties in row). Additionally, p-values derived from Holm's test show to be highly significant (< 0.01).

We also include a Bayesian sign and signed-rank study [34] based on pairwise comparison between methods (in accuracy results). In these tests, a Dirichlet process is assumed over likelihood distributions such that marginals on finite

[6]Considering 5×10^5 instances by dataset on average

Table 4: Wilcoxon pairwise test on accuracy, and ranking of methods generated by Friedman's procedure + adjusted p-value by Holm's Test. *First two columns represents Wilcoxon comparisons, where '+' indicates the number of wins achieved by each algorithm, and '±' the number of wins/ties. The best achievement is highlighted in grease. Third column shows the ranking of methods according to Friedman's test, starting from the control method (topmost). Last column details the adjusted p-values generated by the post hoc Holm's test.*

Algorithms	Accuracy		Ranking	p_{Holm}
	+	±		
LOFD	4	4	9.8182	–
IDA	1	3	29.0909	0.004784
GB	0	3	31.0909	0.003691
OC	0	2	33.4545	0.001620
PiD	0	3	36.5455	0.000365

partitions are Dirichlet distributed. In Bayesian tests, it is assumed two classifiers are practically equivalent if their mean difference of accuracies is within the interval $[-0.01, 0.01]$. This interval defines what is called as region of practical equivalence (rope). Column rope in Table 5 defines the probability of two methods are equal, and corresponds with the area of the posterior within the rope. Column left defines the probability of method A is practically superior to B, whereas column right defines the probability of method B is practically superior to A. Both values corresponds with the area to the left and the right of the rope, respectively. Signed-rank test differs from signed test in that the closed form used to compute the distribution can be replaced by a Monte Carlo sampling of weights, and the latter one does not require a symmetric distribution.

Table 5: Bayesian sign and signed-rank test between LOFD and its competitors. *In each cell, the first number represents the signed probability, and the second one the signed-rank probability.*

Algorithms (LOFD vs. ?)	**left (<<)**	**rope (=)**	**right (>>)**
PiD	1.0—1.0	0.0—0.0	0.0—0.0
IDA	1.0—1.0	0.0—0.0	0.0—0.0
OC	1.0—1.0	0.0—0.0	0.0—0.0
GB	0.8181—0.9973	0.0—0.0	0.1818—0.0002

Table 5 shows that the posterior probability that LOFD is practically superior to the other alternatives. It is close to one for NB, and exactly one for all the discretizers.

Another classifier (HT) is included in the experiments in order to show that LOFD is a versatile, non-exclusive solution for Naïve Bayes. With the inclusion of HT, we have also proven all online classifier susceptible of discretization provided by the MOA platform. Table 6 shows the mean accuracy results following the previous learning procedure but using HT instead of NB. In this scenario, the baseline model (HT with gaussian approximation, 10 splits) stands out as

Table 6: Classification test accuracy on discretized data. Hoeffding tree used as learner.

	PiD	**IDA**	**OC**	**HT**	**LOFD**
airlines	64.3951	64.5158	**65.3619**	65.0784	65.0008
elecNormNew	79.8442	79.8354	70.2132	79.1954	**80.7645**
kddcup_10	**99.8389**	99.7929	99.8368	99.7413	99.5120
poker-lsn	57.9820	69.8381	55.4892	76.0685	**76.1936**
covtypeNorm	77.6671	75.8652	70.1681	80.3119	**81.8190**
blips	73.6652	86.0112	35.7974	**90.9808**	79.3036
sudden_drift	69.5128	82.9856	61.3936	84.8418	**86.7238**
gradual_drift_med	64.6858	84.1394	51.1838	85.5088	**86.5246**
gradual_recurring_drift	68.2206	83.7164	35.6192	**88.3368**	77.8664
incremental_fast	71.1508	78.6526	50.6528	**82.7748**	77.0852
incremental_slow	66.3744	76.7644	50.5308	**83.1052**	70.9906
MEAN	72.1215	80.1924	58.7497	**83.2676**	80.1622

the best method on average. Nevertheless, LOFD represents an interesting option for real data, outperforming its competitors in 3/5 cases. In general, LOFD is advantageous in the whole spectrum of problems (5/11). Finally our solution stands among with IDA as the best discretization alternative on average.

In summary, LOFD has shown to be compatible with other online algorithms beyond NB. Despite HT can be considered as less susceptible to discretization than NB, our solutions also stands as a positive alternative in some problems.

4.2.2. Time results (runtime performance)

It is well-known that supervised approaches tend to be more time-consuming than unsupervised ones, in return they are able to leverage from class information. In streaming environments, it is crucial to control the rapidness of algorithms. Table 7 compares methods in terms of evaluation time (discretization + learning). Unsupervised methods run faster on average than supervised ones, as previously discussed. IDA discretizer is 2 times faster than the closest supervised option (PiD), whereas LOFD (522.42) is ranked as one of the fastest supervised options, behind PiD (508.58). If the throughput rate (time by instance) is computed, we can observe LOFD is able to process each example in approximately 1 millisecond[7], which seems suitable for most of streaming systems.

4.2.3. Number of intervals (model complexity)

Although sometimes irrelevant, the third component to consider here is the number of intervals generated by discretizers. A reduced set of intervals usually imply simpler learning processes, and subsequently, simpler models [16]. However, a reduced number of intervals is typically associated with poor learning capabilities inasmuch as class separability is not fully accomplished. Table 8 depicts information about the simplicity of discretization schemes in terms of number of intervals generated. OC is elected as the best alternative since its

[7]Considering again that the benchmark dataset has 5×10^5 instances.

Table 7: Global time in seconds (discretization + classification)

	PiD	**IDA**	**OC**	**GB**	**LOFD**
airlines	114.62	160.16	595.29	**14.04**	261.48
elecNormNew	28.25	9.17	12.57	**0.67**	16.49
kddcup_10	526.50	158.69	3,850.95	**18.59**	341.49
poker-lsn	129.11	104.30	1,769.26	**11.72**	390.99
covtypeNorm	408.86	275.40	1,694.97	**28.28**	690.43
blips	1,610.87	487.77	1,013.60	**12.02**	780.17
sudden_drift	91.56	74.39	183.15	**2.49**	490.70
gradual_drift_med	210.77	94.08	172.43	**3.63**	672.94
gradual_recurring_drift	1,152.51	429.20	1,038.21	**12.52**	741.98
incremental_fast	986.21	274.86	518.33	**5.68**	853.27
incremental_slow	335.08	246.33	615.29	**6.06**	506.64
MEAN	508.58	210.40	1,042.19	**10.52**	522.42

Table 8: Number of intervals generated by discretizer. *Best value (lowest) by row is highlighted in bold.*

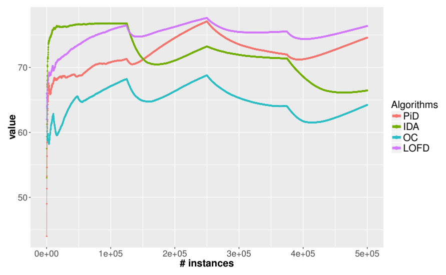
	PiD	**IDA**	**OC**	**LOFD**
airlines	**17**	48	29	39
elecNormNew	81	54	**33**	50
kddcup_10	300	**138**	158	153
poker-lsn	51	55	43	**42**
covtypeNorm	344	330	96	**82**
blips	1,924	126	**120**	552
sudden_drift	22	24	**18**	28
gradual_drift_med	**17**	24	18	30
gradual_recurring_drift	1,829	126	**120**	504
incremental_fast	1,085	66	60	**55**
incremental_slow	313	66	**60**	75
MEAN	543.91	96.09	**68.64**	146.36

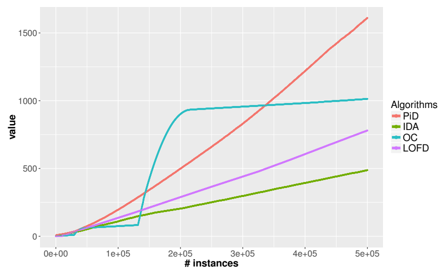
schemes typically consists of few intervals. Nevertheless, in this scenario, simple solutions do not correspond with accurate solutions (see OC results in Table 4). Although LOFD solutions can be deemed as much more complex (more intervals), they lead the accuracy ranking in return.

4.2.4. Case study: drift effect on discretization

Figures 3 (*blips*) and 4 (*poker-lsn*) aim at depicting the effect of drifts on discretizers' performance. In Figure 3, some abrupt peaks can be observed, as well as the LOFD's ability to recover from them properly. This ability allows LOFD outperforms other methods from early stages. Also notice that LOFD is the only algorithm capable of sustaining the accuracy rate after the drift in the midway of the series.

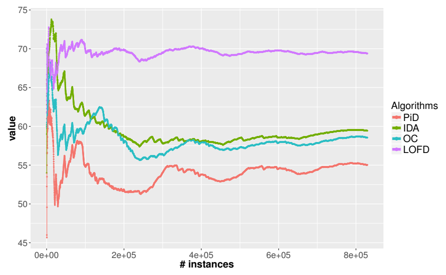
No remarkable drift can be distinguished from Figure 4, however, we can notice that LOFD's accuracy rate is much more competitive and less fluctuating than presented by other methods. Regarding the time plots, LOFD shows an efficiency order close to linear for both problems.

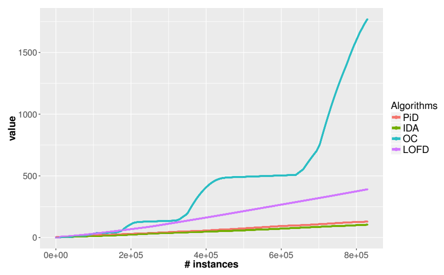




(a) Accuracy (% of correct predictions) (b) CPU time (seconds)

Figure 3: Detailed plots of prequential accuracy, and CPU time over the data stream progress (# instances processed) on *blips*.





(a) Accuracy (% of correct predictions) (b) CPU time (seconds)

Figure 4: Detailed plots of prequential accuracy, and CPU time over the data stream progress (# instances processed) on *poker-lsn*.

4.3. Analytical comparative: statistic-based labeling

Beyond standard labeling, augmented histograms provide enough information for a correct NB-based learning. Labels are not required anymore in this context. This section presents an analysis about how valuable could be the adoption of this scheme in online discretization, and whether the previous scheme is more advantageous for PiD or LOFD. It is noteworthy to remark that the histogram scheme is the default strategy presented in [27], and the only one tested in that work.

Table 9 provides evidence of the negative effect of histogram version on PiD (from 69.25% to 60.52% in accuracy), and its bad results compared to LOFD. Specially remarkable is the case of *poker-lsn* where almost all instances are incorrectly predicted. If we focus on this dataset, we can notice several deficiencies in PiD. First of all, if new values are laid out of the range defined by parameters min/max, there will be required several iterations to create the required intervals. Likewise, as no interval is defined for the new overflowed point, PiD will not provide histogram nor likelihoods to NB. Therefore, subsequent predictions will be almost misguided. In *poker-lsn*, the effect is much more astonishing, simply because NB has more options (classes) to choose from. All these evidences, plus those presented in Section 2.2, show that combination “histogram + PiD”

Table 9: Classification test accuracy (%) + total time on discrete data (histogram scheme)

	Time (s)		**Accuracy**	
Datasets	PiD	LOFD	PiD	LOFD
airlines	**108.08**	221.01	53.4763	**64.6136**
elecNormNew	21.18	**13.81**	74.1989	**75.1324**
kddcup_10	514.84	**140.38**	97.9902	**99.2079**
poker-lsn	108.71	**55.26**	0.1117	**61.0778**
covtypeNorm	407.16	**217.69**	**63.1194**	62.9710
blips	1,243.54	**315.70**	70.9794	**72.7270**
sudden_drift	**89.53**	479.93	38.6880	**83.3610**
gradual_drift_med	**180.98**	480.45	51.1544	**84.4526**
gradual_recurring_drift	1,201.22	**351.77**	62.9858	**64.0668**
incremental_fast	868.54	**500.65**	75.6676	**75.9816**
incremental_slow	278.62	**267.95**	**77.3006**	77.2844
MEAN	456.58	**276.78**	60.5157	**74.6251**

does not work properly.

LOFD outperforms PiD in 8/11 datasets, and its mean accuracy (74.63%) across all datasets is even superior to that of each method in Table 3. It shows that standard/smooth labeling seems to perform better than the histogram alternative in terms of accuracy rate.

Regarding total time (discretization + prediction), there is a clear advantage on relieving NB from its competence of histogram counting. Transition between discretization and prediction is more direct, and that improvement is reflected in LOFD' time results (almost half time than in Table 7). Time improvement in PiD is less noticeable, but still competitive and relevant.

In summary, standard/smooth labeling contributes to obtain much more precise models, and this is a more versatile strategy that can work with any online classifier as mentioned in Section 3.1. Nevertheless, free information provided by histogram-based discretizers allow classifiers to perform faster predictions.

4.4. Case study: sudden drift scenario

This section illustrates the different discretization solutions offered by the discretizers studied, as well as how they adapt their solutions after the appearance of concept drifts. Figure 5 depicts the solutions generated for the *sudden drift* dataset, before and after ($+1 \times 10^4$ instances later) a drift appears in attribute #2. The drift concretely appears after iteration 3.75×10^5

Among with the cut points limiting intervals (vertical lines), a simplified class histogram of the last 1×10^3 points is included in the figure. Left subplots show the density of points before the drift, where most of points are blue, and skewed to the right. After the drift (right subplots) more red points appear on the left side, thus removing practically the skewness. The expected output here is that more intervals appear in the leftmost part of the histogram in order to follow the trend and thus better separate classes.

As observed in Figure 5, only LOFD generates more cut points to the left of the midpoint after the drift, whereas the rest look practically identical to its previous value. In fact, we can observe the other supervised schemes look quite misguided given that intervals are quite concentrated, thus showing a high level of overlapping (specially those of PiD). Concerning IDA, its discretization solutions fits perfectly an equal-frequency approach as expected. LOFD intervals also look well-distributed, but at the same time they respect and follow the class borders.

5. Concluding Remarks

In this paper we have studied several major issues to be faced by contemporary online discretizers. How discretizers should adapt the intervals or how intervals are labeled and interact with learners are two of the main axes on which further developments should revolved. As a potential solution for the interval labeling and interaction problems, we have proposed and analyzed two opposing strategies. The first one is a renovated solution valid for every online learning system, whereas the other one relies on histograms to provide direct information to bayesian learners, for example. Both alternatives have shown in the experiments their positive effect on the transition between consecutive discretization states.

To solve the adaptation problem we have implemented all the labeling schemes in a novel online discretization algorithm, called LOFD. This discretizer produces self-adaptive and highly-informative discretization schemes, in which precise intervals are supported by updated class statistics. LOFD also presents a high level of responsiveness thanks to the fully local strategy implemented, mainly based on fast interval fusions and splits.

The complex experimental framework performed, with 12 datasets and 3 algorithms, has proven that LOFD is by far the most competitive solution in terms of predictive accuracy. It has also been confirmed by the statistical analysis carried out with a significance level $\alpha \leq 0.01$. LOFD is also ranked as one of the most rapid supervised discretizers. Compared with the other alternatives, which either barely cover the search space or generate too many meaningless intervals, LOFD is able to achieve an excellent trade-off between simple and precise solutions.

Acknowledgements

This work is supported by the Spanish National Research Project TIN2014-57251-P, the Foundation BBVA project 75/2016 BigDaPTOOLS, the Andalusian Research Plan P11-TIC-7765. S. Ramírez-Gallego holds a FPU scholarship from the Spanish Ministry of Education and Science (FPU13/00047).

References

References

[1] S. García, J. Luengo, F. Herrera, Data Preprocessing in Data Mining, Springer, 2015.

[2] S. García, J. Luengo, F. Herrera, Tutorial on practical tips of the most influential data preprocessing algorithms in data mining, Knowledge-Based Systems 98 (2016) 1 – 29.

[3] H. Liu, F. Hussain, C. L. Tan, M. Dash, Discretization: An enabling technique, Data Mining and Knowledge Discovery 6 (4) (2002) 393–423.

[4] S. Ramírez-Gallego, S. García, H. Mouriño Talín, D. Martínez-Rego, V. Bolón-Canedo, A. Alonso-Betanzos, J. M. Benítez, F. Herrera, Data discretization: taxonomy and big data challenge, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 6 (1) (2016) 5–21.

[5] H. Chen, T. Li, C. Luo, S. J. Horng, G. Wang, A rough set-based method for updating decision rules on attribute values; coarsening and refining, IEEE Transactions on Knowledge and Data Engineering 26 (12) (2014) 2886–2899.

[6] Y. Yang, G. I. Webb, Discretization for Naive-Bayes learning: managing discretization bias and variance, Machine Learning 74 (1) (2009) 39–74.

[7] X. Wang, Y. He, D. D. Wang, Non-naive bayesian classifiers for classification problems with continuous attributes, IEEE Transaction on Cybernetics 44 (1) (2014) 21–39.

[8] Q. Wu, D. Bell, M. McGinnity, G. Prasad, G. Qi, X. Huang, Improvement of decision accuracy using discretization of continuous attributes, in: Proceedings of the Third International Conference on Fuzzy Systems and Knowledge Discovery, FSKD'06, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 674–683.

[9] J. Lu, P. Zhao, S. C. H. Hoi, Online passive-aggressive active learning, Machine Learning 103 (2) (2016) 141–183.

[10] J. Gama, Knowledge Discovery from Data Streams, Chapman & Hall/CRC, 2010.

[11] M.-A. Aufaure, R. Chiky, O. Curé, H. Khrouf, G. Kepeklian, From business intelligence to semantic data stream management, Future Generation Computer Systems 63 (Supplement C) (2016) 100 – 107, modeling and Management for Big Data Analytics and Visualization.

[12] S. Ramírez-Gallego, A. Fernández, S. García, M. Chen, F. Herrera, Big data: Tutorial and guidelines on information and process fusion for analytics algorithms with mapreduce, Information Fusion 42 (Supplement C) (2018) 51 – 61.

[13] R. Pears, S. Sakthithasan, Y. S. Koh, Detecting concept change in dynamic data streams, Machine Learning 97 (3) (2014) 259–293.

[14] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Computing Surveys 46 (4) (2014) 44:1–44:37.

[15] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woniak, F. Herrera, A survey on data preprocessing for data stream mining: Current status and future directions, Neurocomputing 239 (2017) 39 – 57.

[16] S. García, J. Luengo, J. A. Sáez, V. López, F. Herrera, A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning, IEEE Transactions on Knowledge and Data Engineering 25 (4) (2013) 734–750.

[17] B. S. Chlebus, S. H. Nguyen, On Finding Optimal Discretizations for Two Attributes, Springer Berlin Heidelberg, Berlin, Heidelberg, 1998, pp. 537–544.

[18] T. Elomaa, J. Rousu, General and efficient multisplitting of numerical attributes, Machine Learning 36 (1999) 201–244.

[19] S. Ramírez-Gallego, S. García, J. M. Benítez, F. Herrera, Multivariate discretization based on evolutionary cut points selection for classification, IEEE Transactions on Cybernetics 46 (3) (2016) 595–608.

[20] D. A. Zighed, S. Rabaséda, R. Rakotomalala, FUSINTER: A method for discretization of continuous attributes, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6 (3) (1998) 307–326.

[21] M. M. Gaber, Advances in data stream mining, Wiley Interdisciplinary Review: Data Mining and Knowledge Discovery 2 (1) (2012) 79–85.

[22] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, M. Woźniak, Ensemble learning for data stream analysis: a survey, Information Fusion 37 (2017) 132 – 156.

[23] M. Tennant, F. T. Stahl, O. Rana, J. B. Gomes, Scalable real-time classification of data streams with concept drift, Future Generation Computer Systems 75 (2017) 187–199.

[24] S. Sakthithasan, R. Pears, A. Bifet, B. Pfahringer, Use of ensembles of fourier spectra in capturing recurrent concepts in data streams, in: 2015 International Joint Conference on Neural Networks IJCNN, 2015, pp. 1–8.

[25] B. Krawczyk, M. Woźniak, One-class classifiers with incremental learning and forgetting for data streams with concept drift, Soft Computing 19 (12) (2015) 3387–3400.

[26] G. Webb, Contrary to popular belief incremental discretization can be sound, computationally efficient and extremely useful for streaming data, in: IEEE International Conference on Data Mining (ICDM), 2014, pp. 1031–1036.

[27] J. Gama, C. Pinto, Discretization from data streams: Applications to histograms and data mining, in: Proceedings of the 2006 ACM Symposium on Applied Computing, SAC '06, 2006, pp. 662–667.

[28] P. Lehtinen, M. Saarela, T. Elomaa, Online ChiMerge Algorithm, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, Ch. Data Mining: Foundations and Intelligent Paradigms: Volume 2: Statistical, Bayesian, Time Series and other Theoretical Aspects, pp. 199–216.

[29] T. Elomaa, P. Lehtinen, Maintaining optimal multi-way splits for numerical attributes in data streams, in: Advances in Knowledge Discovery and Data Mining, 12th Pacific-Asia Conference, PAKDD 2008, Osaka, Japan, May 20-23, 2008 Proceedings, 2008, pp. 544–553.

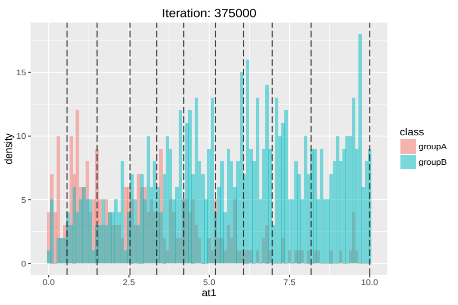
[30] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: massive online analysis, Journal of Machine Learning Research 11 (2010) 1601–1604.

[31] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, in: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '01, 2001, pp. 97–106.

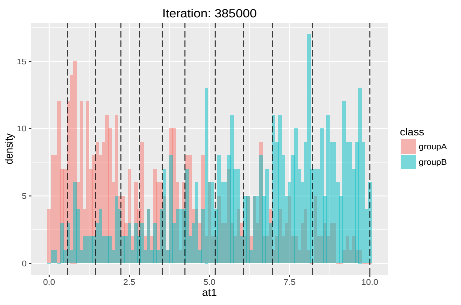
[32] A. Bifet, R. Kirkby, Data stream mining: a practical approach, Tech. rep., The University of Waikato (2009).

[33] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power, Information Sciences 180 (10) (2010) 2044 – 2064, special Issue on Intelligent Distributed Information Systems.

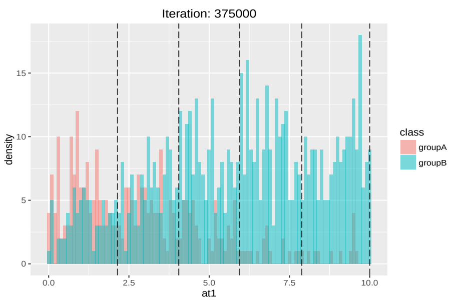
[34] A. Benavoli, G. Corani, F. Mangili, M. Zaffalon, F. Ruggeri, A bayesian wilcoxon signed-rank test based on the dirichlet process, in: Proceedings of the 31th International Conference on Machine Learning, ICML 2014, 21-26, 2014, pp. 1026–1034.



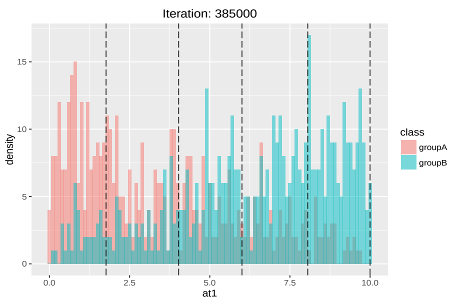
(a) LOFD: density function (iteration: 3.75×10^5)



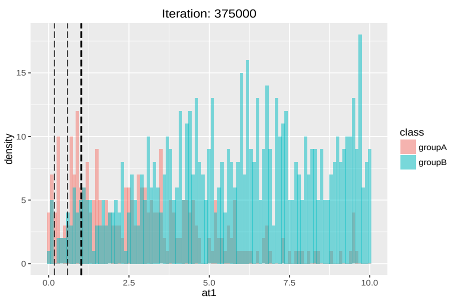
(b) LOFD: density function (iteration: 3.85×10^5)



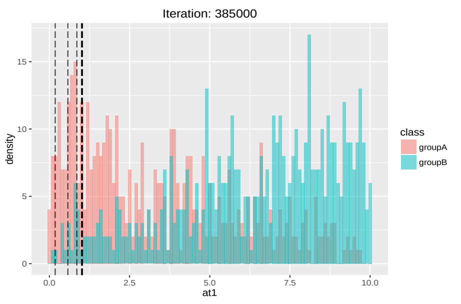
(c) IDA: density function (iteration: 3.75×10^5)



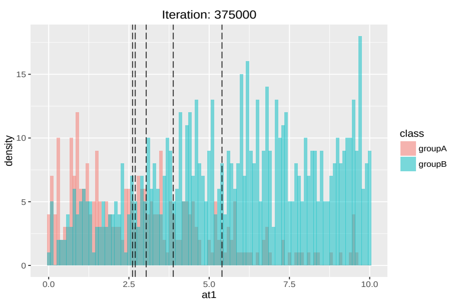
(d) IDA: density function (iteration: 3.85×10^5)



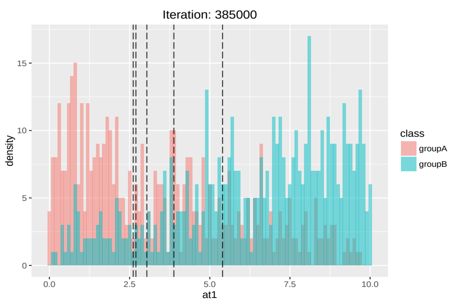
(e) PiD: density function (iteration: 3.75×10^5)



(f) PiD: density function (iteration: 3.85×10^5)



(g) OC: density function (iteration: 3.75×10^5)



(h) OC: density function (iteration: 3.85×10^5)

Figure 5: Density plots before and after a concept drift in attribute #2 (*sudden drift* dataset). *Each row represents a different discretizer, each column the distribution of data before and after the drift, and each vertical line the intervals generated.*

Sergio Ramírez-Gallego received the M.Sc. degree in Computer Science in 2012 from the University of Jaén, Spain. He is currently a Ph.D. student at the Department of Computer Science and Artificial Intelligence, University of Granada, Spain. He has published in journals such as IEEE Transactions on Cybernetics, IEEE Transactions on Systems, Man, and Cybernetics: Systems, Experts Systems with Applications or Neurocomputing. His research interests include data mining, data preprocessing, big data and cloud computing.

Salvador García received the M.Sc. and Ph.D. degrees in Computer Science from the University of Granada, Granada, Spain, in 2004 and 2008, respectively. He is currently an Associate Professor in the Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain. He has published more than 45 papers in international journals. As edited activities, he has co-edited two special issues in international journals on different Data Mining topics and is a member of the editorial board of the Information Fusion journal. He is a co-author of the book entitled "Data Preprocessing in Data Mining" published in Springer. His research interests include data mining, data preprocessing, data complexity, imbalanced learning, semi-supervised learning, statistical inference, evolutionary algorithms and biometrics.

Francisco Herrera (SM'15) received his M.Sc. in Mathematics in 1988 and Ph.D. in Mathematics in 1991, both from the University of Granada, Spain. He is currently a Professor in the Department of Computer Science and Artificial Intelligence at the University of Granada.

He has been the supervisor of 40 Ph.D. students. He has published more than 300 journal papers that have received more than 49000 citations (Scholar Google, H-index 112). He is coauthor of the books "*Genetic Fuzzy Systems*" (World Scientific, 2001) and "*Data Preprocessing in Data Mining*" (Springer, 2015), "*The 2-tuple Linguistic Model. Computing with Words in Decision Making*" (Springer, 2015), "*Multilabel Classification. Problem analysis, metrics and techniques*" (Springer, 2016), "*Multiple Instance Learning. Foundations and Algorithms*"(Springer, 2016).

He currently acts as Editor in Chief of the international journals "Information Fusion" (Elsevier) and "Progress in Artificial Intelligence (Springer). He acts as editorial member of a dozen of journals.

He received the following honors and awards: ECCAI Fellow 2009, IFSA Fellow 2013, 2010 Spanish National Award on Computer Science ARITMEL to the "Spanish Engineer on Computer Science", International Cajastur "Mamdani" Prize for Soft Computing (Fourth Edition, 2010), IEEE Transactions on Fuzzy System Outstanding 2008 and 2012 Paper Award (bestowed in 2011 and 2015 respectively), 2011 Lotfi A. Zadeh Prize Best paper Award of the International Fuzzy Systems Association, 2013 AEPIA Award to a scientific career in Artificial Intelligence, and 2014 XV Andalucía Research Prize Maimónides (by the regional government of Andalucía).

His current research interests include among others, soft computing (including fuzzy modeling and evolutionary algorithms), information fusion, decision making, biometric, data preprocessing, data science and big data.

Bibliography

[Agg15] Aggarwal C. C. (2015) *Data Mining: The Textbook.* Springer Publishing Company, Incorporated.

[AIS93] Agrawal R., Imieliński T., and Swami A. (1993) Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD international conference on Management of data* 22(2): 207–216.

[Alp10] Alpaydin E. (2010) *Introduction to Machine Learning.* The MIT Press, 2nd edition.

[AS94] Agrawal R. and Srikant R. (1994) Fast algorithms for mining association rules. In *Proceedings of 20th International Conference on VLDB*, pp. 487–499.

[BCnAB15] Bolón-Canedo V., no N. S.-M., and Alonso-Betanzos A. (2015) Recent advances and emerging challenges of feature selection in the context of Big Data. *Knowledge-Based Systems* 86(Supplement C): 33 – 45.

[BCSMAB13] Bolón-Canedo V., Sánchez-Maroño N., and Alonso-Betanzos A. (2013) A review of feature selection methods on synthetic data. *Knowledge and Information Systems* 34(3): 483–519.

[BGE15] Barddal J. P., Gomes H. M., and Enembreck F. (2015) A survey on feature drift adaptation. In *IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 1053–1060.

[Bou04] Boullé M. (2004) Khiops: A statistical discretization method of continuous attributes. *Machine Learning* 55: 53–69.

[Brz15] Brzezinski D. (2015) *Block-based and Online Ensembles for Concept-drifting Data Streams.* PhD thesis, Poznan University of Technology.

[BSA15] Bolón-Canedo V., Sánchez-Maroño N., and Alonso-Betanzos A. (2015) *Feature Selection for High-Dimensional Data.* Artificial Intelligence: Foundations, Theory, and Algorithms. Springer.

[CCP13] Chen S.-M., Chang Y.-C., and Pan J.-S. (2013) Fuzzy rules interpolation for sparse fuzzy rule-based systems based on interval type-2 gaussian fuzzy sets and genetic algorithms. *IEEE Transactions on Fuzzy Systems* 21(3): 412–425.

[CH67] Cover T. M. and Hart P. E. (1967) Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* 13(1): 21–27.

[Cis16] Cisco (2016) Cisco global cloud index: Forecast and methodology, 2015-2020. [Online; accessed Januray 2018].

[CLQW03] Chen C.-W., Li Z.-G., Qiao S.-Y., and Wen S.-P. (2003) Study on discretization in rough set based on genetic algorithm. In *Proceedings of the Second International Conference on Machine Learning and Cybernetics (ICMLC)*, pp. 1430–1434.

[CLS+16] Cheng S., Liu B., Shi Y., Jin Y., and Li B. (2016) Evolutionary computation and Big Data: Key challenges and future directions. In Tan Y. and Shi Y. (Eds.) *Data Mining and Big Data: First International Conference (DMBD'16)*, pp. 3–14. Springer International Publishing.

[CM98] Cherkassky V. S. and Mulier F. (1998) *Learning from Data: Concepts, Theory, and Methods*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

[CS14] Chandrashekar G. and Sahin F. (2014) A survey on feature selection methods. *Computers and Electrical Engineering* 40(1): 16 – 28.

[CS17] Czolombitko M. and Stepaniuk J. (2017) Scalable maximal discernibility discretization for Big Data. In Polkowski L., Yao Y., Artiemjew P., Ciucci D., Liu D., Slezak D., and Zielosko B. (Eds.) *Rough Sets: International Joint Conference, IJCRS 2017*, pp. 644–654. Springer International Publishing.

[DG08] Dean J. and Ghemawat S. (2008) Mapreduce: Simplified data processing on large clusters. *Communications of the ACM* 51(1): 107–113.

[DGH10] Derrac J., García S., and Herrera F. (2010) A survey on evolutionary instance selection and generation. *Internation Journal of Applied Metaheuristic Computing* 1(1): 60–92.

[DH00] Domingos P. and Hulten G. (2000) Mining High-Speed Data Streams. In Parsa I., Ramakrishnan R., and Stolfo S. (Eds.) *Proceedings of the ACM Sixth International Conference on Knowledge Discovery and Data Mining*, pp. 71–80. ACM Press, Boston, USA.

[DHS00] Duda R., Hart P., and Stork D. (2000) *Pattern Classification*. Wiley-Interscience, John Wiley & Sons, Southern Gate, Chichester, West Sussex, England, 2nd edition.

[ES03] Eiben A. E. and Smith J. E. (2003) *Introduction to Evolutionary Computing*. Natural Computing. Springer-Verlag.

[Esh91] Eshelman L. J. (1991) The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. volumen 1 of *Foundations of Genetic Algorithms*, pp. 265 – 283.

[FdRL+14] Fernández A., del Río S., López V., Bawakid A., del Jesús M. J., Benítez J. M., and Herrera F. (2014) Big Data with Cloud Computing: an insight on the computing environment, MapReduce, and programming frameworks. *Wiley Interdisciplinary Rewiews: Data Mining and Knowledge Discovery* 4(5): 380–409.

[FFBS17] Franc V., Fikar O., Bartos K., and Sofka M. (2017) Learning data discretization via convex optimization. *Machine Learning* , in press.

[FI93] Fayyad U. M. and Irani K. B. (1993) Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1022–1029.

[FILn07] Flores J. L., Inza I., and Larrañaga P. (2007) Wrapper discretization by means of estimation of distribution algorithms. *Intelligent Data Analysis* 11(5): 525–545.

[Fre02] Freitas A. A. (2002) *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag New York, Inc.

[Gab12] Gaber M. M. (2012) Advances in data stream mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2(1): 79–85.

[Gam10] Gama J. (2010) *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC.

[GDCH12] García S., Derrac J., Cano J. R., and Herrera F. (2012) Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34(3): 417–435.

[GIM99] Gionis A., Indyk P., and Motwani R. (1999) Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pp. 518–529.

[GLBH17] Gutiérrez P. D., Lastra M., Benítez J. M., and Herrera F. (2017) SMOTE-GPU: Big data preprocessing on commodity hardware for imbalanced classification. *Progress in Artificial Intelligence* 6(4): 347–354.

[GLH15] García S., Luengo J., and Herrera F. (2015) *Data Preprocessing in Data Mining*. Springer.

[GLS$^+$13] García S., Luengo J., Sáez J. A., López V., and Herrera F. (2013) A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering* 25(4): 734–750.

[GMCR04] Gama J., Medas P., Castillo G., and Rodrigues P. P. (2004) Learning with drift detection. In *Advances in Artificial Intelligence - SBIA 2004, 17th Brazilian Symposium on Artificial Intelligence*, pp. 286–295.

[Gol89] Goldberg D. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[GZB$^+$14] Gama J., Zliobaite I., Bifet A., Pechenizkiy M., and Bouchachia A. (2014) A survey on concept drift adaptation. *ACM Computing Surveys* 46(4): 44:1–44:37.

[Har75] Hartigan J. A. (1975) *Clustering Algorithms*. John Wiley & Sons, Inc., New York, NY, USA.

[HKP11] Han J., Kamber M., and Pei J. (2011) *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition.

[HKZ$^+$15] Hamstra M., Karau H., Zaharia M., Konwinski A., and Wendell P. (2015) *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, Incorporated.

[HSD01] Hulten G., Spencer L., and Domingos P. M. (2001) Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 97–106.

[HTH06] He Z., Tian S., and Huang H. (2006) EMVD-BDC: An evolutionary multivariate discretization approach for association rules. *Journal of Computational Information Systems* 2(4): 1343–1350.

[Jol02] Jolliffe I. T. (2002) *Principal Component Analysis.* Springer, 2nd edition.

[KMG$^+$17] Krawczyk B., Minku L. L., Gama J., Stefanowski J., and Woźniak M. (2017) Ensemble learning for data stream analysis: A survey. *Information Fusion* 37: 132 – 156.

[KŠRŠ97] Kononenko I., Šimec E., and Robnik-Šikonja M. (1997) Overcoming the myopia of inductive learning algorithms with RELIEFF. *Applied Intelligence* 7(1): 39–55.

[Kun08] Kuncheva L. I. (2008) Classifier ensembles for detecting concept change in streaming data: Overview and perspectives. In *2nd Workshop SUEMA 2008 (ECAI 2008)*, pp. 5–10.

[LA11] Lughofer E. and Angelov P. P. (2011) Handling drifts and shifts in on-line data streams with evolving fuzzy systems. *Applied Soft Computing* 11(2): 2057–2068.

[Lan01] Laney D. (2001) 3d data management: Controlling data volume, velocity and variety. [Online; accessed Januray 2018].

[LGP15] Leyva E., Gonzlez A., and Prez R. (2015) Three new instance selection methods based on local sets: A comparative study with several approaches from a bi-objective perspective. *Pattern Recognition* 48(4): 1519–1533.

[LHTD02] Liu H., Hussain F., Tan C. L., and Dash M. (2002) Discretization: An enabling technique. *Data Mining and Knowledge Discovery* 6(4): 393–423.

[Lin12] Lin J. (2012) Mapreduce is good enough? if all you have is a hammer, throw away everything that's not a nail! *Big Data* 1: 28–37.

[LLZdM16] Lu N., Lu J., Zhang G., and de Mantaras R. L. (2016) A concept drift-tolerant case-base editing technique. *Artificial Intelligence* 230: 108 – 133.

[LMGY04] Liu T., Moore A. W., Gray A. G., and Yang K. (2004) An investigation of practical approximate nearest neighbor algorithms. In *Advances in Neural Information Processing Systems NIPS*, pp. 825–832.

[LS97] Liu H. and Setiono R. (1997) Feature selection via discretization. *IEEE Transactions on Knowledge and Data Engineering* 9: 642–645.

[LWL$^+$17] Liu W., Wang Z., Liu X., Zeng N., Liu Y., and Alsaadi F. E. (2017) A survey of deep neural network architectures and their applications. *Neurocomputing* 234: 11 – 26.

[MBY$^+$16] Meng X., Bradley J., Yavuz B., Sparks E., Venkataraman S., Liu D., Freeman J., Tsai D., Amde M., Owen S., Xin D., Xin R., Franklin M. J., Zadeh R., Zaharia M., and Talwalkar A. (2016) Mllib: Machine learning in apache spark. *Journal of Machine Learning Research* 17(34): 1–7.

[MCG$^+$10] Masud M. M., Chen Q., Gao J., Khan L., Han J., and Thuraisingham B. (2010) Classification and novel class detection of data streams in a dynamic feature space. In *Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part II*, ECML PKDD'10, pp. 337–352.

[MSC13] Mayer-Schnberger V. and Cukier K. (2013) *Big Data: A Revolution That Will Transform How We Live, Work and Think. Viktor Mayer-Schnberger*. John Murray Publishers.

[NMH$^+$14] Nuij W., Milea V., Hogenboom F., Frasincar F., and Kaymak U. (2014) An automated framework for incorporating news into stock trading strategies. *IEEE Transactions on Knowledge and Data Engineering* 26(4): 823–835.

[NZD11] Ning A., Zhang X., and Duan W. (2011) DHMM speech recognition algorithm based on immune particle swarm vector quantization. In *Proceedings of the Third International Conference on Artificial Intelligence and Computational Intelligence - Volume Part III*, AICI'11, pp. 420–427.

[Pyl99] Pyle D. (1999) *Data Preparation for Data Mining*. Morgan Kaufmann Publishers Inc.

[Qui93] Quinlan J. R. (1993) *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc.

[Sam05] Samet H. (2005) *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc.

[Sim08] Simon D. (2008) Biogeography-based optimization. *IEEE Transactions on Evolutionary Computation* 12(6): 702–713.

[TDGH12] Triguero I., Derrac J., García S., and Herrera F. (2012) A taxonomy and experimental study on prototype generation for nearest neighbor classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42(1): 86–100.

[vdMPvdH09] van der Maaten L. J. P., Postma E. O., and van den Herik H. J. (2009) Dimensionality Reduction: A Comparative Review. Technical report, Tilburg University.

[WBM$^+$06] Wu Q., Bell D., McGinnity M., Prasad G., Qi G., and Huang X. (2006) Improvement of decision accuracy using discretization of continuous attributes. In *Proceedings of the Third International Conference on Fuzzy Systems and Knowledge Discovery*, FSKD'06, pp. 674–683. Springer-Verlag, Berlin, Heidelberg.

[Web14] Webb G. (2014) Contrary to popular belief incremental discretization can be sound, computationally efficient and extremely useful for streaming data. In *IEEE International Conference on Data Mining (ICDM)*, pp. 1031–1036.

[WFH11] Witten I. H., Frank E., and Hall M. A. (2011) *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann.

[WGC14] Woźniak M., Graña M., and Corchado E. (2014) A survey of multiple classifier systems as hybrid systems. *Information Fusion* 16: 3–17.

[Whi12] White T. (2012) *Hadoop, The Definitive Guide.* O'Reilly Media, Inc.

[WHW14] Wang X., He Y., and Wang D. D. (2014) Non-naive bayesian classifiers for classification problems with continuous attributes. *IEEE Transactions on Cybernetics* 44(1): 21–39.

[WK09] Wu X. and Kumar V. (Eds.) (2009) *The Top Ten Algorithms in Data Mining.* Data Mining and Knowledge Discovery. Chapman & Hall/CRC.

[WZWD14] Wu X., Zhu X., Wu G.-Q., and Ding W. (2014) Data mining with Big Data. *IEEE Transactions on Knowledge and Data Engineering* 26(1): 97–107.

[YXZ$^+$17] Yin C., Xia L., Zhang S., Sun R., and Wang J. (2017) Improved clustering algorithm based on high-speed network data stream. *Soft Computing* , in press.

[ZCD$^+$12] Zaharia M., Chowdhury M., Das T., Dave A., Ma J., McCauley M., Franklin M. J., Shenker S., and Stoica I. (2012) Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pp. 2–2.

[ZG14] Zliobaite I. and Gabrys B. (2014) Adaptive preprocessing for streaming data. *IEEE Transactions on Knowledge and Data Engineering* 26(2): 309–321.

[ZHJ04] Zhang G., Hu L., and Jin W. (2004) Discretization of continuous attributes in rough set theory and its application. In *Proceedings of the First International Conference on Computational and Information Science*, CIS'04, pp. 1020–1026.

[ZOT14] Zhai Y., Ong Y., and Tsang I. W. (2014) The emerging "big dimensionality". *IEEE Computational Intelligence Magazine* 9(3): 14–26.