UNIVERSIDAD DE GRANADA

BIG DATA ENSEMBLES FOR CLASSIFICATION AND SMART DATA EXTRACTION

DOCTORAL DISSERTATION
presented to obtain the
DOCTOR OF PHILOSOPHY DEGREE
in the
INFORMATION AND COMMUNICATION TECHNOLOGY PROGRAM
by

Diego Jesús García Gil

PhD Advisors

Francisco Herrera Triguero & Salvador García López

COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE DEPARTMENT

Granada, January 2020

Editor: Universidad de Granada. Tesis Doctorales
Autor: Diego Jesús García Gil
ISBN: 978-84-1306-501-4
URI: http://hdl.handle.net/10481/62290

This doctoral thesis has been supported by the Spanish National Research Projects TIN-2014-57251-P and TIN-2017-89517-P, and the Project BigDaP-TOOLS - Ayudas Fundación BBVA a Equipos de Investigación Científica 2016.

A mi familia.

Agradecimientos

Si bien esta tesis doctoral solo tiene un nombre en la portada, un proyecto de tal calibre no es posible llevarlo a cabo en solitario. Se necesita del apoyo de todas aquellas personas que, de una forma u otra, con intención o sin saberlo, ayudan, apoyan, motivan y animan a que proyectos como este lleguen a buen fin.

En primer lugar, me gustaría dedicar esta tesis a mi familia, la cual no ha dejado de animarme y apoyarme en ningún momento durante todo este tiempo. En especial a mi padre, Jacinto, por enseñarme a luchar y tirar para adelante pase lo que pase, a mi madre, M^{a} Carmen, por su ternura y cariño, y a mi hermana, M^{a} Paz, por su alocada alegría. También va dedicada a mis abuelos, Diego, Gabriela y José, los cuales me han enseñado de dónde provengo, y a mi abuela M^{a} Paz, que desde allá donde esté me guía y hace el camino más fácil. Por último, una dedicación especial a mi pareja, Ana, que aunque se sumara a esta aventura estando ya empezada, se ha convertido en parte fundamental de que haya llegado a buen puerto. Gracias por estar ahí y compartir mis alegrías y penas dándome ánimo y fuerza (y galletas) cuando más lo he necesitado, has hecho este proceso mucho más fácil.

De esta etapa académica, me gustaría agradecer a mis directores de tesis, Francisco Herrera y Salvador García, por la confianza y el tiempo invertido en mí en forma de reuniones, correos, revisiones, y un largo etcétera. También agradecer a todos los compañeros doctorandos que me han acompañado en esta etapa: Jesús, Sergio, Jacinto, Anabel, Elena, José Ángel, etc. A Carlos, compañero de despacho y amigo desde que entramos juntos al D1-6 por primera vez, y al que le deseo lo mejor en su nueva etapa como papá. A Francisco Luque, por haber compartido conmigo frustraciones y alegrías. Y en especial a Francisco J. Baldán, que desde que lo conocí se ha convertido en alguien fundamental en mi vida. Gracias por tantos y tantos momentos de felicidad, agobio, nervios y consejos mutuos (y todos los que están por venir) tanto dentro como fuera de la universidad.

Todo esto no hubiera sido posible sin la ayuda y guía de todos los doctorandos “veteranos” en su momento, que nos enseñaron que por largo que parezca el proceso, al final merece la pena. En particular a Sergio R., paisano y guía cuando he necesitado ayuda. Y por supuesto agradecer a todos los “senior” del grupo por haber creado una gran familia.

Special thanks to Dr. Xiong and his research group for the amazing treatment and the support received. I would like to also thank Johan for the priceless hospitality during our stay at Västerås. Agradecer también a Miguel y Laura, por hacer que tres meses en un país desconocido pasaran como unos pocos días en casa de unos amigos.

Por último, agradecer a Fran y Juanja, amigos que me conocen desde hace años y que por muy lejos que estemos, y sin importar el tiempo que pasemos sin vernos, sabemos que estamos ahí para lo que haga falta.

GRACIAS A TODOS/AS

Table of Contents

Acronyms

AUC	Area Under the Curve
DM	Data Mining
ENN-BD	Edited Nearest Neighbor for Big Data
GM	Geometric Mean
HME-BD	Homogeneous Ensemble for Big Data
HTE-BD	Heterogeneous Ensemble for Big Data
ICE_BD	Imbalanced Classification Ensemble for Big Data
IEF-BD	Iterative Ensemble Filter for Big Data
IR	Imbalance Ratio
KDD	Knowledge Discovery in Databases
KNN	K-Nearest Neighbors
ML	Machine Learning
MR	MapReduce
PCA	Principal Components Analysis
PCARDE	Principal Components Analysis Random Discretization Ensemble
RD	Random Discretization
RDD	Resilient Distributed Dataset
ROS	Random OverSampling
RP	Random Projection
RPRDE	Random Projection Random Discretization Ensemble
RUS	Random UnderSampling
SMOTE	Synthetic Minority Oversampling Technique

Chapter I

PhD Dissertation

1 Introduction

Big Data analytics has a simple purpose, to get to know people and our environment in the best possible way. We have come to a moment in time when everything in the world has grown to such a limits, where companies, governments and researches have to go one step further and look for new sources of information, new sources of knowledge which will enable us to develop and adapt further [CCS12].

Companies, governments and researchers are constantly looking for information they have not been interested in so far, but as of today, this information is the only way to take this next step. All this information comes from a myriad of sources, both structured and unstructured. Nevertheless, both companies and researchers are facing new challenges coping with the *Volume*, *Velocity*, *Veracity*, and *Value* (among many other V's) that characterize this new paradigm [RGFG$^+$18].

Recently, Smart Data has been introduced, aiming to filter out, or amend, the imperfections and to highlight the valuable data. Smart Data aims to transform raw data into quality data that can lead to knowledge [LGGRG$^+$20]. It converts Big Data into useful information that is accurate. A challenge that becomes even trickier is the management of the quality of the data in Big Data environments. Advanced Big Data modeling and analytics are indispensable for discovering the underlying structure from retrieved data in order to acquire Smart Data.

The main concept of Big Data is that these huge amounts of information will enable data mining (DM) algorithms to achieve better and more accurate models than ever before, in a timely manner. Classic DM algorithms are not prepared to handle these new requirements in terms of data size and performance. This will impede the application of the *Knowledge Discovery in Databases* (KDD) process [WZWD13]. KDD process can be defined as the non-trivial process of identifying valid, potentially useful and understandable patterns in the data. The key aspect of the KDD process is the steps into which it is divided:

1. **Problem specification**: sets the desired target of the discovery and the kind of knowledge you want to extract.
2. **Data extraction**: analyze the most important sources of information, and integrate all that sources into a single piece, known as dataset.
3. **Data preprocessing**: reduces, cleans and fixes the data in order to obtain a quality dataset for further stages.
4. **Data mining**: extracts patterns and/or models from the preprocessed data.
5. **Data interpretation and evaluation**: analyzes the extracted knowledge and presents the results in a user friendly shape.

All these different phases constitute the KDD process. However, they have received different attention because of its importance and weight in the KDD process. The objective of DM is to find repetitive patterns, tendencies o rules that explain the data in a particular context [WFHP16]. Depending on the type of the targeted results, DM techniques can be classified in predictive methods - that make predictions about future or unknown events, and descriptive models - that identify different relationships in the data.

Moreover, attending to the type of the target variable, whether it is defined or not, we can distinguish two different groups:

- **Supervised learning**: the objective is to predict the value of the target variable for new instances by the definition of the relation between input variables and the target variable. Two different families can be distinguished:
  - *Classification* [DHS12]: the target variable is a discrete value, and the different set of possible outcomes (namely classes or labels) are known. For example, red, green or blue in simple color classification.
  - *Regression* [CM07]: the domain of the target variable is continuous. For example, forecasting the amount of rain.
- **Unsupervised learning**: the target variable is undefined. The aim is to discover implicit relations in the data. They can be separated in two different groups:
  - *Clustering* [Har75]: creates groups of similar instances (intra-cluster distance), with the largest separation among groups as possible (inter-cluster distance).
  - *Association* [AIS93]: discover common relations between variables.

This thesis is focused on supervised classification tasks. Although DM is seen as the most important task of the KDD process, it ultimately depends on the quality of the data. This data can be affected by numerous negative factors such as noise, missing values, or inconsistencies in data size, among many others. A well-known principle in data science is the *garbage in - garbage out* principle: no matter how good your data or model are, as long as one is bad, the results will be poor [GLH15, LGGRG$^+$20]. Data preprocessing is a crucial step that enables DM algorithms to find more useful patterns and with better quality [ZWM14]. In spite of being frequently in the background, data preprocessing is the most time and effort consuming task of the KDD process [Pyl99].

With the increasing size of the generated and stored data, traditional DM and data preprocessing techniques are facing a serious challenge: being able to process Big Data effectively and efficiently. Distributed computing has been widely used by data scientists even before the advent of Big Data phenomenon for speeding up the KDD process [RMBG18]. However, it has become mandatory in Big Data environments. This not only requires to adapt existing algorithms, but also to propose new ones, born from and to handle Big Data problems.

MapReduce (MR) is one of the first distributed computing paradigm that enabled the generation and processing of Big Data datasets in an automatic and distributed fashion [DG04]. MR has become the reference in distributed computing paradigms because of its simplicity and fault tolerance. By implementing two functions, *Map* and *Reduce*, users are able to process large amounts of data without worrying about technical aspects, such as data partitioning, failure recovery or job communication.

Apache Hadoop is the most popular and widely used implementation of the MR paradigm [Whi12, LKRH15]. However, despite becoming the reference in performance, Apache Hadoop had some limitations that have hinder its application in some contexts. The lack of in-memory processing, or the low iterative performance are some examples [Lin13]. Recently, a novel framework focused on speed and easy of use have taken Apache Hadoop spot in the Big Data science community. This framework, named Apache Spark, solves Apache Hadoop limitations by providing in-memory computing [HKZ$^+$15]. This is achieved thanks to a novel data structure, called Resilient Distributed Datasets (RDD) [ZCD$^+$12]. They are able to reuse data partitions, and to recover them in case of failure. Since Apache Hadoop was first released, more and more frameworks have emerged focusing on different Big Data aspects. Apache Flink is a recent Apache project designed for distributed stream and batch data processing [HK19]. It tries to fill the "online" gap left by Apache Spark, which employs a mini-batch streaming processing instead of a pure streaming approach.

Ensembles are methods that combine a set of base classifiers to make predictions [Die00a]. These methods have been gaining more and more attention because of their excellent performance, and their ability to correct errors across many base classifiers. The main idea of ensembles is that learning a set of basic classifier models, with some differences between them, will create a better global model. Diversity is the key concept of ensembles. With small changes in input data or base classifiers, diverse base models are created and better ensembles are obtained. The application of data preprocessing techniques in ensembles is a natural approach, since they can help to create the required level of diversity.

Novel Big Data frameworks, like Apache Spark or Apache Flink, include distributed ML libraries. However, only classic classification and data preprocessing methods are included, such as decision trees or data scalers. As stated earlier, ensembles are the one of the best performing methods in DM. If we attend to distributed ensembles for classification, only golden standards are included, such as Random Forest [Bre01]. Given the excellent performance of ensembles in classification problems, new distributed and scalable ensemble methods are required, designed specifically for tackling Big Data classification tasks effectively. Since ensembles are strongly related with data preprocessing, Big Data ensembles based on data preprocessing should be closely studied.

Most DM techniques and algorithms assume that the data is accurate. However, data extracted from the real world is far from perfection. In any KDD process the value of extracted knowledge is directly related to the quality of the data used. Big Data problems also follow the same dictate. A common problem affecting data quality is the presence of noise, particularly in classification problems, where label noise refers to the incorrect labeling of training instances, and is known to be a very disruptive feature of data. It can be caused by faults in data acquisition, transmission and/or storage. Noise disrupts the boundaries of the classes, harming the DM process and its performance. It also alters the posterior interpretability of the model, as well as the conclusions drawn from it [ZW04]. Big Data problems pose a new challenge in terms of quality data due to the massive accumulation of data. This Big Data scenario also brings new problems to classic data preprocessing algorithms, as they are not prepared for working with such amounts of data, and these algorithms are key to move from Big Data to high quality Smart Data.

The negative impact on learning associated to the imbalanced proportion of classes has exploded lately with the exponential grow of "cheap" data. Many real world problems present scarce number of instances in one class, whereas in others their cardinality is several factors greater. Moreover, underrepresented classes are typically those that contain the concept of interest. Because of this, its correct classification poses a challenge. This problem is known as imbalanced data classification, and it's gaining lots of attention in the last years [FGG$^+$18]. The current techniques that treat imbalanced Big Data problems are focused on obtaining fast, scalable and parallel sampling techniques following the standard MR procedure. These generate local balanced solutions in each *Map*, which are eventually combined into a final set. Nevertheless, this divide-and-conquer strategy entails several problems, such as small disjuncts, data lack, etc. Algorithms able to efficiently balance and classify imbalanced Big Data are required for tackling this problem.

The present thesis addresses different topics: standard classification, noise filtering, and imbalanced data classification. All topics spin around a common denominator, Big Data ensembles for achieving Smart Data. First, a complete study of the most popular Big Data frameworks up to date will be performed, in order to draw the current state-of-the-art in terms of scalability power, and open and future problems. We will design an ensemble method based on data preprocessing for

Big Data classification. Then, we will tackle noise filtering in Big Data classification, an unexplored problem so far. We will design different noise filtering ensemble methods, proving that noise filter in Big Data is a mandatory step that cannot be disregarded. Finally, we will focus on the imbalanced Big Data classification problem, and will propose an ensemble for classifying such data.

Finally, this thesis consists of two different parts: the PhD dissertation and the publications. In the first part, in Section 1 provides the general context of this project. Section 2 describes the main concepts that are used in this thesis. 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 objectives and the methodology followed to develop the ideas proposed are described in Section 4 and Section 5, respectively. In Section 6 is proposed an introduction to publications. In Section 7 the main results of these publications are presented. Finally, Section 8 provides the overall conclusions and open future lines derived from this thesis (Section 9).

The second part of the document consists of the five publications that compose this thesis, organized according to the proposed objectives explained before:

- A comparison on scalability for batch big data processing on Apache Spark and Apache Flink.
- Principal Components Analysis Random Discretization Ensemble for Big Data.
- Enabling Smart Data: Noise Filtering in Big Data Classification.
- From Big to Smart Data: Iterative Ensemble Filter for Noise Filtering in Big Data classification.
- Smart Data based Ensemble for Imbalanced Big Data Classification.

Introducción

La analítica de datos Big Data tiene un propósito simple: conocer a la gente y a nuestro entorno de la mejor forma posible. Hemos llegado a un momento en el que todo ha crecido hasta unos límites que han obligado a las compañías, gobiernos e investigadores a ir un paso más allá y buscar nuevas fuentes de información, nuevas fuentes de conocimiento que nos permita continuar desarrollándonos y adaptándonos [CCS12].

Las compañías, gobiernos e investigadores están constantemente buscando información en la que no se habían interesado antes. A partir de ahora, esta información es la única manera de dar ese siguiente paso. Toda esta información proviene de una gran variedad de fuentes, tanto estructuradas como desestructuradas. Además, las compañías e investigadores están lidiando con los nuevos retos de *Volumen*, *Velocidad*, *Veracidad* y *Valor* (entre otras muchas V's) que caracterizan este nuevo paradigma [RGFG$^+$18].

Recientemente se ha introducido el término Smart Data, con el objetivo de filtrar o arreglar las imperfecciones presentes en los datos, así como resaltar aquellos que son valiosos. Smart Data pretende transformar datos en crudo en datos de calidad que puedan crear nuevo conocimiento [LGGRG$^+$20]. Convierte datos Big Data en información útil y precisa. Un reto que se ha vuelto más difícil aún es la gestión de la calidad de los datos en entornos Big Data. La analítica y modelado avanzado de Big Data son indispensables para el descubrimiento de las estructuras subyacentes de los datos, con el objetivo de obtener Smart Data.

El principal concepto del Big Data es que esa gran cantidad de información permitirá a los algoritmos de minería de datos alcanzar modelos mejores y más precisos que nunca de forma eficiente. Los algoritmos clásicos de minería de datos no están preparados para trabajar con estos nuevos requerimientos de volumen y velocidad. Esto impedirá la aplicación del llamado proceso de descubrimiento de conocimiento a partir de bases de datos (en inglés, KDD) [WZWD13]. El proceso KDD se puede definir como aquel proceso no trivial de identificación de patrones válidos, potencialmente útiles y entendibles en los datos. El aspecto clave del proceso KDD son los pasos en los que está dividido:

1. **Especificación del problema**: establece el objetivo de descubrimiento deseado y el tipo de conocimiento que se pretende extraer.

2. **Extracción de datos**: analiza las fuentes de información más importantes e integra todas esas fuentes en una única pieza, conocida como conjunto de datos.

3. **Preprocesamiento de datos**: reduce, limpia y arregla los datos con el objetivo de obtener un conjunto de datos de calidad para las etapas posteriores.

4. **Minería de datos**: extrae patrones y/o modelos de los datos preprocesados.

5. **Interpretación y evaluación de los datos**: analiza el conocimiento extraído y presenta los resultados de una forma agradable para el usuario.

Todas estas diferentes fases constituyen el proceso KDD. Sin embargo, estas fases han recibido diferente atención por su importancia y peso en dicho proceso. El objetivo de la minería de datos es encontrar patrones repetidos, tendencias o reglas que expliquen los datos en un contexto en particular [WFHP16]. Dependiendo del tipo de resultados objetivo, las técnicas de minería de datos se pueden clasificar en modelos predictivos - hacen predicciones sobre eventos futuros o desconocidos, y modelos descriptivos - identifican diferentes relaciones en los datos.

Dependiendo de si la variable objetivo está definida o no, podemos distinguir dos grupos diferentes:

- **Aprendizaje supervisado**: el objetivo es predecir el valor de la variable objetivo para nuevas instancias, por medio de la definición de la relación entre los datos de entrada y dicha variable objetivo. Podemos distinguir dos familias diferentes:
  - *Clasificación* [DHS12]: la variable objetivo es un valor discreto, y el conjunto de posibles salidas (también llamado clases o etiquetas) es conocido. Por ejemplo, rojo, verde o azul en una clasificación simple de colores.
  - *Regresión* [CM07]: el dominio de la variable objetivo es continuo. Por ejemplo, la predicción de la cantidad de lluvia.
- **Aprendizaje no supervisado**: la variable objetivo no está definida. El objetivo es descubrir relaciones implícitas en los datos. Se pueden dividir en dos grupos diferentes:
  - *Clustering* [Har75]: crea grupos de instancias similares (distancia intra-clúster), con la máxima separación entre grupos posible (distancia inter-clúster).
  - *Asociación* [AIS93]: descubre relaciones comunes entre variables.

Esta tesis está centrada en la tarea de clasificación supervisada. A pesar de que la minería de datos se considera la tarea más importante del proceso KDD, está supeditada a la calidad de los datos. Dichos datos pueden verse afectados por numerosos factores negativos como el ruido, la presencia de valores perdidos, o inconsistencias en el tamaño de los datos, entre otros muchos. Un principio conocido en la ciencia de datos es el de basura a la entrada – basura a la salida: no importa cómo de buenos sean los datos o el modelo, si uno de los dos es de mala calidad, el resultado también lo será [GLH15, LGGRG^{+}20]. El preprocesamiento de datos es una tarea crucial que permite a los algoritmos de minería de datos encontrar patrones más útiles y con mejor calidad [ZWM14]. A pesar de encontrarse con frecuencia en un segundo plano, el preprocesamiento de datos es la tarea que más tiempo y esfuerzo consume del proceso de KDD [Pyl99].

Con el creciente tamaño de los datos que se generan y almacenan, las técnicas tradicionales de minería y preprocesamiento de datos están encontrando un serio desafío: ser capaces de procesar Big Data de forma efectiva y eficiente. Los científicos de datos han usado la computación distribuida con regularidad incluso antes de la llegada del fenómeno Big Data para acelerar el proceso de KDD [RMBG18]. Sin embargo, se ha convertido en obligatoria para entornos Big Data. Esto no solo conlleva la adaptación de los algoritmos existentes, sino también el proponer nuevos, nacidos por y para tratar problemas Big Data.

MapReduce (MR) es uno de los primeros paradigmas de computación distribuida que permitió la generación y procesamiento de conjuntos de datos Big Data de forma automática y distribuida [DG04]. MR se ha convertido en una referencia en computación distribuida por su simplicidad y tolerancia a fallos. Por medio de dos funciones, *Map* y *Reduce*, los usuarios pueden procesar grandes cantidades de datos sin preocuparse por aspectos técnicos, como el particionamiento de datos, la recuperación en caso de fallo, o la comunicación entre tareas.

Apache Hadoop es la implementación del paradigma MR más popular y extendida [Whi12, LKRH15]. Sin embargo, a pesar de haberse convertido en el referente en rendimiento, Apache

Hadoop presenta algunas limitaciones que lastran su aplicación en algunos contextos. La falta de procesamiento en memoria, o el bajo rendimiento en procesos iterativos son algunos ejemplos de ello [Lin13]. Recientemente, una nueva herramienta, centrada en la velocidad y facilidad de uso, ha ocupado el sitio de Apache Hadoop en la comunidad Big Data. Esta herramienta, llamada Apache Spark, solventa las limitaciones de Apache Hadoop por medio de la computación en memoria [HKZ+15]. Esto lo consigue gracias a una novedosa estructura de datos, llamada Resilient Distributed Datasets (RDD) [ZCD+12]. Los RDD tienen la capacidad de reusar particiones de los datos y de recuperarlas en caso de fallo. Desde que Apache Hadoop salió a la luz, ha emergido un número creciente de herramientas centradas en diferentes aspectos del Big Data. Apache Flink es un proyecto Apache reciente, diseñado para el procesamiento distribuido de datos en bloque y flujo continuo [HK19]. Su objetivo es llenar el hueco dejado por Apache Spark en cuanto a procesamiento "online", el cual emplea un procesamiento de mini-bloques en lugar de una aproximación puramente continua.

Los ensembles son métodos que combinan una serie de clasificadores base para hacer predicciones [Die00a]. Estos métodos han ido ganando atención debido a su excelente rendimiento, así como por su habilidad para corregir errores por medio de los diferentes clasificadores base. La idea principal de los métodos de ensemble es que al aprender un conjunto de clasificadores base, con algunas diferencias entre ellos, se creará un mejor modelo global. La diversidad es el concepto clave de los ensembles. Con pequeños cambios en los datos de entrada de los clasificadores base se obtienen modelos base diversos, que resultarán en mejores ensembles. Las técnicas de preprocesamiento de datos tienen una aplicación directa en los ensembles, ya que permiten crear el nivel necesario de diversidad.

Las nuevas herramientas para Big Data, como Apache Spark o Apache Flink, incluyen librerías para aprendizaje automático. Sin embargo, solo incluyen algoritmos clásicos de clasificación y preprocesamiento de datos, como los árboles de decisión o métodos de escalado. Como se ha comentado anteriormente, los ensembles son los métodos con mejor rendimiento en la minería de datos. Si nos centramos en los ensembles distribuidos para clasificación, solo se incluyen algoritmos estándar, como los Random Forest [Bre01]. Dado el excelente rendimiento de los ensembles en problemas de clasificación, son necesarios nuevos métodos de ensemble distribuidos y escalables, diseñados específicamente para abordar problemas de clasificación Big Data de forma eficiente. Ya que los ensembles están altamente relacionados con el preprocesamiento de datos, los ensembles para Big Data basados en preprocesamiento de datos deberán ser estudiados en detenimiento.

La mayoría de las técnicas y algoritmos de minería de datos asumen que los datos son precisos. Sin embargo, los datos extraídos del mundo real están lejos de ser perfectos. En cualquier proceso de KDD, el valor del conocimiento extraído está directamente relacionado con la calidad de los datos empleados. Los problemas Big Data siguen el mismo dictado. Un problema común que afecta a la calidad de los datos es la presencia de ruido, en particular en problemas de clasificación, en los que el ruido de clase hace referencia al incorrecto etiquetado de las instancias de entrenamiento. Este problema es conocido por ser una característica disruptiva de los datos. El ruido en los datos puede ser causado por fallos en la adquisición, transmisión y/o almacenamiento de los datos. También afecta a las fronteras entre clases, lastrando el proceso de minería de datos así como su rendimiento. Además, altera la interpretabilidad del modelo, así como las conclusiones obtenidas de el [ZW04]. Los problemas Big Data suponen un nuevo reto en términos de calidad de los datos dada la acumulación masiva de los mismos. Este escenario Big Data también trae nuevos problemas a los algoritmos clásicos de preprocesamiento de datos, ya que no están preparados para trabajar con tales cantidades de datos, y estos algoritmos con la clave para pasar de Big Data a Smart Data.

El impacto negativo asociado al aprendizaje en presencia de una proporción desbalanceada de clases se ha incrementado últimamente con el crecimiento exponencial de datos “baratos”. Muchos problemas reales presentan un número escaso de instancias para una clase, mientras que las otras tienen una cardinalidad varias veces superior. Además, las clases con una baja representación son frecuentemente aquellas que contienen el concepto de interés. Por ello, su correcta clasificación supone un reto. Este problema es conocido como clasificación de datos desbalanceada, y está ganando mucha atención en los últimos años [FGG+18]. Las actuales técnicas para tratar problemas Big Data desbalanceados están centradas en obtener muestras de los datos balanceadas de forma rápida y escalable siguiendo el procedimiento estándar MR. Esto genera soluciones locales en cada *Map*, las cuales son combinadas eventualmente en un conjunto final. Esta estrategia divide y vencerás conlleva varios problemas como pequeñas disyunciones, falta de datos, etc. Se necesitan algoritmos capaces de balancear y clasificar de forma eficiente datos Big Data desbalanceados.

La presente tesis aborda diferentes temas: clasificación estándar, filtrado de ruido y clasificación desbalanceada. Todos ellos giran en torno a un denominador común: métodos de ensemble en Big Data para obtener Smart Data. En primer lugar se realizará un completo estudio de las herramientas Big Data más populares actualmente, con el objetivo de dibujar el estado del arte en términos de capacidad de escalabilidad, así como problemas actuales y futuros. Diseñaremos un método de ensemble basado en preprocesamiento de datos para clasificación Big Data. Después abordaremos el filtrado de ruido en problemas de clasificación para Big Data, un problema inexplorado hasta ahora. Se diseñarán diferentes métodos de ensemble para el filtrado de ruido, demostrando que el filtrado de ruido en Big Data es un paso fundamental que no puede ser ignorado. Por último nos centraremos en el problema de la clasificación desbalanceada en Big Data, y propondremos un ensemble para la clasificación de esos datos.

Finalmente, esta tesis está formada por dos partes diferenciadas: la tesis doctoral y las publicaciones. En la primera parte, en la Sección 1 se describe el contexto general de este proyecto. En la Sección 2 se definen los principales conceptos que soportan 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 y la metodología empleada para el desarrollo de las ideas propuestas se describen en la Sección 4 y en la Sección 5, respectivamente. En la Sección 6 se propone una introducción a las publicaciones relacionadas con esta tesis. Por otro lado, en la Sección 7 se explican los principales resultados de estas publicaciones. Finalmente, en la Sección 8 se describen las principales conclusiones extraídas, así como las futuras líneas abiertas que deja este extenso trabajo (Sección 9).

La segunda parte del documento consta de las cinco publicaciones que componen esta tesis, organizadas según los objetivos propuestos:

- A comparison on scalability for batch big data processing on Apache Spark and Apache Flink.
- Principal Components Analysis Random Discretization Ensemble for Big Data.
- Enabling Smart Data: Noise Filtering in Big Data Classification.
- From Big to Smart Data: Iterative Ensemble Filter for Noise Filtering in Big Data classification.
- Smart Data based Ensemble for Imbalanced Big Data Classification.

2 Preliminaries

This section presents the main concepts of the topics that have driven this thesis. First, Section 2.1 depicts the Big Data problem as well as describes the novel paradigms, frameworks and tools created for this new concept. In Section 2.2, a couple of lines are devoted to ensemble methods, and their adaptation to Big Data environments. Then, the different data preprocessing family branches and their methods are presented in Section 2.3, with special emphasis in Smart Data extraction. Lastly, Section 2.4 concludes with a description of the imbalanced classification problem.

2.1 Big Data

Vast amounts of information surround us today. Technologies such as the Internet generate data at an exponential rate, thanks to the affordability and great development of storage and network resources. It is predicted that by 2020, the digital universe will be 10 times as big as it was in 2013, totaling an astonishing 44 zettabytes. The current volume of data has exceeded the processing capabilities of classical data mining systems and have created a need for new hardware and software for storing and processing this data [WZWD13]. It is widely accepted that we have entered the Big Data era [LGGRG$^+$20]. Big Data can be defined as the set of technologies that make processing such large amounts of data possible, while most of the classic KDD methods cannot work in a Big Data environment because they were not conceived for it.

Big Data can be defined as a high *Volume* of data, generated at a high *Velocity*, with a high *Veracity*, and a potentially high *Value*. This conforms what is known as the four Big Data V's (among many others) [LGGRG$^+$20]. While the *Volume* and *Velocity* aspects refer to the data generation process and how to capture and store the data, *Veracity* and *Value* aspects deal with the quality and the usefulness of the data. These two last aspects become crucial in any Big Data process, where the extraction of useful and valuable knowledge is strongly influenced by the quality of the used data.

Distributed computing has established itself as the solution for tackling the Big Data problem. It was widely used by data scientists before the advent of Big Data, but with the explosion of data, it has become essential. Many standard and time consuming methods and algorithms have been replaced by their distributed approaches, with the aim of not only accelerating the learning process, but also enabling it. As a result of this rapidly evolving Big Data environment, countless tools, paradigms and techniques have emerged, ready to tackle the Big Data situation [LGGRG$^+$20]. All these tools have one thing in common: bring closer cluster computing to the standard user (engineers and data scientists) by hiding the technical nuances derived from distributed environments.

Among all technologies devoted to deal with the rapidly growing rates of generated data, MR is the seminal paradigm designed by Google in 2004 [DG04]. This paradigm is born from the necessity of processing and/or generating large amounts of data in a distributed and efficient fashion, while minimizing disk access and network use. It also provides fault tolerance, automatic data partition and management, and automatic job-resource scheduling. MR follows a divide and conquer approach for the processing and generation of large datasets with parallel and distributed algorithms on a cluster. The MR model is composed of two phases: *Map* and *Reduce*. The *Map* phase performs a transformation of the data, and the *Reduce* phase performs a summary operation.

The *Map* and *Reduce* functions are defined with respect to an essential data structure, known as (key, value) pairs. To summarize this process, Figure 1 illustrates a typical MR workflow with its *Map* and *Reduce* phases.

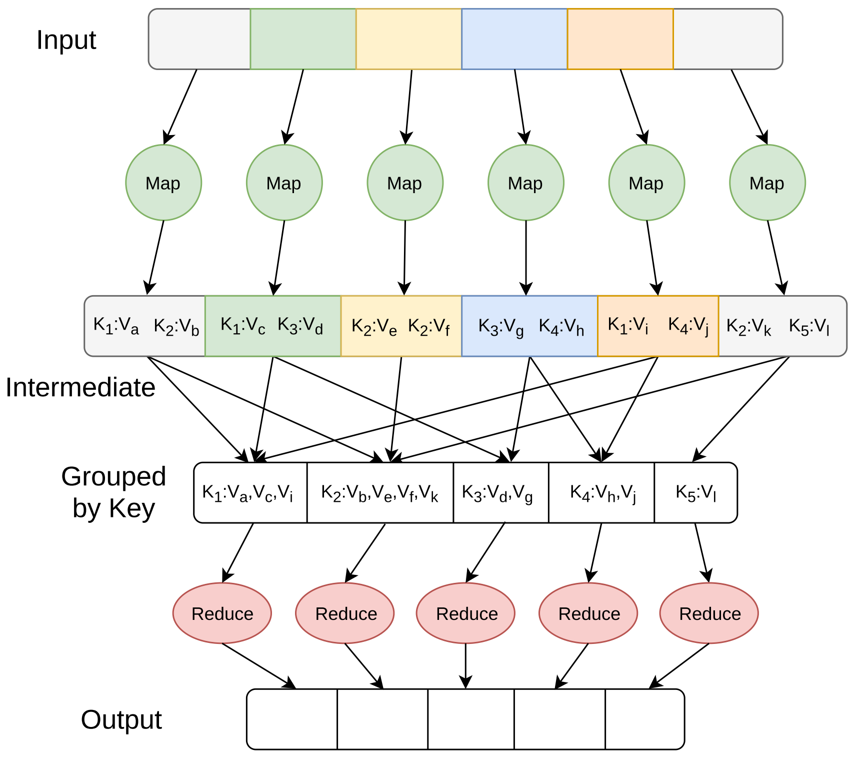

Input
Map
Intermediate
K_1:V_a K_2:V_b K_1:V_c K_3:V_d K_2:V_e K_2:V_f K_3:V_g K_4:V_h K_1:V_i K_4:V_j K_2:V_k K_5:V_l
Grouped by Key
K_1:V_a,V_c,V_i K_2:V_b,V_e,V_f,V_k K_3:V_d,V_g K_4:V_h,V_j K_5:V_l
Reduce
Output

Figure 1: The MR paradigm, K elements represent the keys in the pairs, and V the values.

The MR scheme can be described as follows:

- Before starting the *Map* phase, the master node partitions the data, transforms the instances into a key-value format, and distributes them across the cluster.
- The *Map* function applies a transformation operation to the local (key, value) pairs of each computing node.
- Once the *Map* phase is finished, all pairs belonging the same key are redistributed based on the (key, value) pairs generated in the *Map* phase.
- When all pairs belonging to the same key are in the same computing node, the *Reduce* phase starts. The *Reduce* phase is a summary operation that generates the final values.

Apache Hadoop is the most popular open-source implementation of MapReduce [Whi12]. Although Hadoop is a very extended tool for Big Data processing, it has some important limitations, like intensive disk usage, insufficiency for in-memory computation, poor performance on online and iterative computing, or low inter-communication capacity [Lin13].

Recently, Apache Spark have emerged as an open-source framework for Big Data, focused on speed, ease of use and sophisticated analytics [HKZ$^+$15]. Apache Spark solves Apache Hadoop limitations by in-memory computing, allowing to persist the data in memory for consecutive or iterative processing. Users can load their data into memory and iterate over it repeatedly, making it a suitable tool for ML algorithms. It is built on top of a novel distributed data structure, named RDDs [ZCD$^+$12]. They are a distributed collection of unsorted and immutable data, allowing users to persist them in memory. RDDs can also be tracked and recomputed using a lineage for recovering lost or failed partitions. RDDs are composed of two different types of operations: transformations and actions. The former is applied to each data partition of the RDD, and generates a new RDD. Transformations are lazy operations, meaning that they are not evaluated until an actions is computed on that RDD. Actions trigger all previous transformations of an RDD, and return a value.

As part of the Apache Spark project, a distributed ML library was developed. MLlib project [MBY$^+$16] was born in 2012 as an extra component of Apache Spark. This led to Apache Spark becoming the reference among all other Big Data frameworks for Big Data ML. MLlib includes several out-of-the-box algorithms for alike tasks, such as: classification, clustering, regression, recommendation, and basic data preprocessing. Recently, a new version of MLlib, based on Datasets, has been presented, using the new additions to the Apache Spark framework. Apart from official MLlib API, Spark provides a community package index, known as Spark Packages to collect all open-source algorithms that integrates with MLlib [Pac19].

Regarding Big Data stream processing, Apache Flink have emerged as the reference, solving problems derived from micro-batch models (Apache Spark Streaming) [Fli19]. Apache Flink also supports batch data processing, though it is treated as a special case of streaming. Similarly to Apache Spark, Apache Flink also includes a fault tolerance mechanism to 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 [HK19].

Apache Flink also includes a distributed and stream-oriented ML library, FlinkML. However, it provides few alternatives for some fields in ML, like support vector machines, multiple linear regression for supervised learning, KNN join for unsupervised learning, or scalers and polynomial features for data preprocessing,

Despite the presence of several distributed ML libraries, only golden standard algorithms have been redesigned for distributed Big Data environments. This lack of methods unables reaching Smart Data from raw Big Data. Novel scalable and distributed designs for both classification and data preprocessing, focused on Smart Data extraction, are required to extend and maintain these libraries.

2.2 Ensembles

Ensembles are methods that combine a set of base classifiers to make predictions [Rok10]. In contrary to other classic learners which build one learner from the data, ensembles learn multiple learners and combine them. Ensembles are also referenced in the literature as committee-based learning, or multiple classifier systems [Kun14].

Ensemble methods are designed to increase the global accuracy by learning a set of base classifiers and combining all the decisions to return a single decision or label [Die00b]. These base classifiers are usually simple learners, such as decision trees, neural networks, or other kind of ML algorithm. Depending on the type of the different base learners employed in the construction of the ensemble, we can distinguish between homogeneous ensembles - all base learners are the same type, and heterogeneous ensembles - base learners are of a different kind.

These methods have been attracting increasing attention over the last few years due to their ability to correct errors across many diverse base classifiers. Thanks to the combination of base learners, the generalization capability of ensembles is often much stronger than those base classifiers. Because of this, ensembles are one of the best performing methods in ML nowadays.

With the generated set of base learners comes the decision of which strategy is best for combining them [Zho12]. Rather than just trying to find the best single learner, ensemble methods follow different mechanisms to produce the best combination of base learners:

- **Averaging**: is the most popular method for the combination of numerical outputs. Depending on if all base learners are treated the same, or if they have different weights, different types of averaging are employed:
  - *Simple averaging*: combines the output by averaging the outputs of the different base learners.
  - *Weighted average*: achieves a combined output in cases when base learners have different importance.
- **Voting**: employed for categorical outputs, it is the most popular and fundamental combination method. Different types of voting strategies are available:
  - *Majority voting*: each base classifier votes for one class, and the final decision is the one that receives at least half of the votes.
  - *Plurality voting*: selects the class with the largest number of votes as the decision.
  - *Weighted voting*: assigns a weight to each learner, and gives more power to the stronger classifiers decisions.
  - *Soft voting*: when learners output class probabilities instead of one class label, soft voting generates the combined output by averaging all outputs.
- **Stacking** [Wol92]: a learner is trained to combine the base learners (also known as meta-learner).

The key concept of ensembles is diversity [Zho12]. Diversity can be defined as the difference among individual base learners. Ensembles correct errors in classification through learning classifiers that have some differences among them [WGC14]. With diverse classifiers, ensembles will be more robust to noise and outliers, and will achieve better performance. In Figure 2 we can see a graphic representation of the learning process of an ensemble method. This diversity can be introduced following different mechanisms:

- **Data level**: with small changes in input data, diverse classifiers are obtained. It can vary from data sampling techniques, such as bagging [Bre96] or boosting [$\text{FS}^{+}96$], to modifications of the data. This is usually introduced through the application of some data preprocessing techniques.
- **Classifier level**: using different classifiers, or tweaking the parameters of a single classifier can produce diverse classifiers and better ensembles.
- **Hybrid approach**: a combination of data level diversity and classifier level diversity can be employed to obtain highly diverse ensemble methods.

Two different paradigms of ensemble methods can be differentiated depending on how base learners are generated: sequential ensembles and parallel ensembles. The former learns the different models sequentially, whereas in the latter, base learners are generated in a parallel fashion. The idea

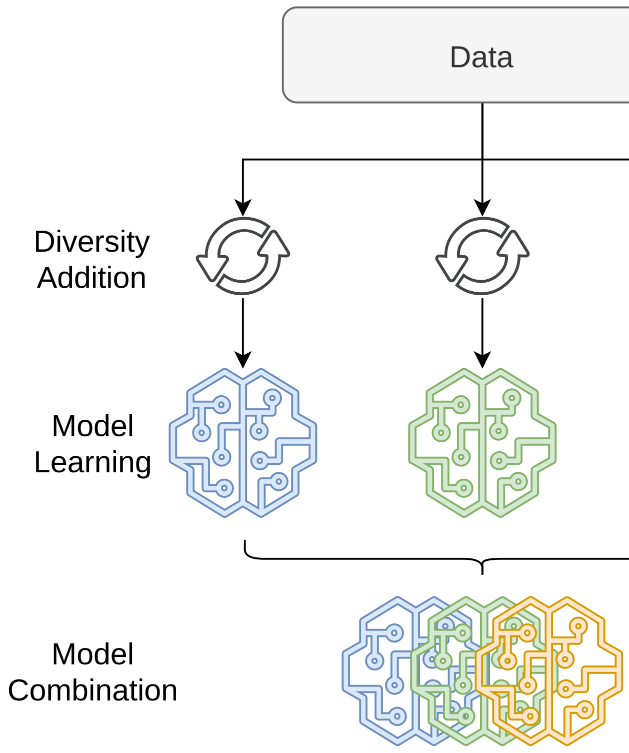


Figure 2: The ensemble learning process introduces diversity on each base model in order to obtain different models for improving the global performance of those base classifiers.

of each of these paradigms is to exploit either dependence or independence among base learners, as two different strategies to create ensembles. The best representative examples of methods for such strategies are boosting and bagging.

Boosting is a general procedure to convert weak learners into strong learners [FS$^+$96]. It trains a set of base learners in a sequential manner, and combines them for prediction. Each learner focus on the misclassifications of the earlier learner by increasing or decreasing weights in order to improve the detection of instances hard to discriminate. The AdaBoost algorithm [FS97] is the most influential boosting algorithm.

Bagging is focused on data level diversity. It samples the data randomly with replacement each iteration of the ensemble [Bre96]. Although base learners are generated from the same data, the introduction of randomness in the learning process can generate some independence among them. This improves the generalization ability of the ensemble [Zho12]. Another benefit is their ability to be adapted to distributed computing, since all base learners are trained from different subsets of the data, they can be processed independently. The most popular ensemble method using this technique is Random Forest [Bre01], which combines bagging with decision trees.

Several ensemble methods have been proposed in the literature since its inception. Classification ensembles such as XGBoost [CG16], LightGBM [KMF$^+$17] or CatBoost [PGV$^+$18] have become some of the best performing methods in ML nowadays. However, their adaptation to Big Data scenarios is still an ongoing process.

If we focus on Big Data ensemble proposals, we can only find classic methods like Random Forest or boosting ensembles available in the most popular ML Big Data libraries. Novel distributed ensemble proposals are needed, ready to tackle Big Data problems.

2.3 Data Preprocessing & Smart Data

In this Big Data era, the lack of human supervision, and the automation in the data acquisition and storing process have led to the acceptance that data will have low quality due to the presence of imperfections, redundancies or inconsistencies, among other pernicious traits. These imperfections can be provoked by sensors failing, anomalous situations, or exogenous factors, among others. Low quality in data can make impossible the later learning process. The set of techniques devoted to tackle those imperfections, and to improve the quality of the data are known as data preprocessing [GLH15]. There are different families of data preprocessing algorithms, being the most widely used the data reduction techniques, imperfect data methods, and imbalance data handling.

Recently, a new term related to the Big Data environment has emerged. Smart Data refers to the challenge of transforming raw data into quality data that can be appropriately exploited to obtain valuable insights [LGGRG$^+$20]. This new concept aims to achieve quality data with *Veracity* and *Value* properties. Therefore, Smart Data is focused on extracting valuable knowledge from data, in the form of a subset (big or not), that contains enough quality for a successful DM process. The impact of Smart Data extraction in industry and academia is two-fold: higher quality DM and reduction of data storage costs.

Data preprocessing is strongly linked to the concept of Smart Data, as it is one of the most important phases of the KDD process. Raw data is likely to contain imperfections, redundancies, or inconsistencies, making it unsuitable for a successful DM process. The goal of data preprocessing is to clean and amend errors in the data, and to improve the performance of the later ML process. This transformation is the difference between “Big” and “Smart” Data, as can be seen in Figure 3.

Among all imperfections affecting data, noise is one of the most disruptive ones. Noise can be defined as an exogenous or external factor that affects the data and corrupts it. As data grows, noise accumulates and algorithmic instability appears, particularly when a massive sample pool has been integrated from heterogeneous sources. Noise disrupts the models obtained and decrements the performance of ML algorithms. Alleviating the effects of noise is a challenging task that requires the correct identification of the corrupted examples in the data.

As stated earlier, noise contained in data entails a critical impact on the models learned from it. Noise sensitive learners aggravate this negative impact. Moreover, data size is a factor that increases the amount of noise in data, since noise accumulates when the number of dimensions and instances increases [FHL14]. We can distinguish two different types of noise in the literature, depending on which element of the data is affecting [ZW04]:

- **Class noise**: also referred as *label noise*, describes wrongly labeled examples. It is conformed by misclassifications (examples with a wrong label), and contradictory examples (instances with different label, but the same input attributes).
- **Attribute noise**: refers to corruptions in data attributes. In spite of being composed of several data problems (missing values or “do not care” values among others), it is mainly focused on erroneous values.

Among the two different types of noise presented, class noise if the most disruptive with the learning process. For this reason, many efforts have been devoted to tackle it. Class noise have different origins, such as errors in the data acquisition process, or subjectivity in the data labeling phase.

Different approaches for dealing with class noise can be found in the literature. These techniques

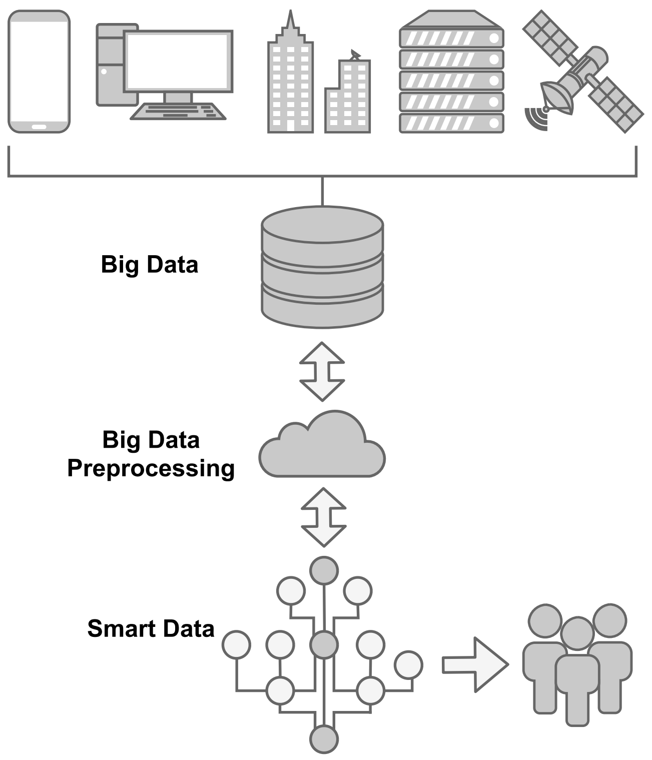


Figure 3: Big Data preprocessing is the key to transform raw Big Data into quality Smart Data.

vary from the creation of noise-robust learners and algorithms, to data preprocessing techniques capable of detect and remove or correct noisy examples. Two different approaches can be distinguished for the noise handling problem [FV14]:

- **Algorithm level**: these methods aim to create robust learning algorithms that are little or not influenced by the presence of noise, pruning strategies to avoid overfitting to noisy instances, or decreasing the importance of noisy instances. Hybrid strategies that combine both approaches have also been proposed, which model the noise and diminish the importance of noise in the learning process.

- **Data level**: also known as *filters*. They aim to remove noisy examples from a dataset, as a previous step to the learning process. We can find ensemble strategies [VVA03], data partitioning approaches, or iterative algorithms [BF99].

The negative effects of noise increase with the size of the data. Data preprocessing methods are also affected by the increasing size and complexity of the data, making them unable to obtain a preprocessed/smart dataset in a reasonable time limit. Although a Big Data scenario is very likely to contain huge amounts of noise, little research has been devoted to tackling noise in Big Data. That is why there is a special need for noise filters in Big Data. While some architectural designs are already proposed in the literature [Zer16], there is no particular algorithm which deals with noise in Big Data classification, nor a comparison of its effect on model generalization abilities or computing times.

2.4 Imbalanced Classification

Among the wide set of problems worsened or even directly provoked by Big Data, the treatment of imbalanced data in binary classification can be highlighted as one of the most common affecting Big datasets [LGGRG$^+$20]. In a supervised classification problem, the existence of a notable difference in the number of instances belonging to different classes is known as imbalanced classification. The class with the lower number of examples is known as the *minority* class. Similarly, the class with the largest number of instances is referred as the *majority* class. Moreover, the classes which are underrepresented are typically those that arouses most interest; therefore, its correct identification becomes primary [FdRCH17].

This problem entails a great challenge to standard ML algorithms, since they are often guided by global search measures weighted in favor of accuracy that constantly overlook this imbalanced situation [FGG$^+$18]. This will led the ML algorithms to ignore the minority class, and treat it as noise, because more general rules that model the majority class will be preferred.

Many efforts have been devoted to tackle imbalanced classification. The different techniques proposed can be divided into three categories:

- **Data level**: these methods modify the data in order to obtain a balanced dataset. This approach makes use of imbalanced data preprocessing techniques.
- **Algorithm level**: it entails the modification of already existing methods for improving the detection of the minority class.
- **Cost-sensitive methods**: they combine above two approaches. They incorporate data level modifications by adding costs to instances, and algorithm level adaptations.

Algorithm level modifications are considered an alternative to data preprocessing (or data level) techniques. Data preprocessing methods modify the data in order to improve the detection of both classes, whereas algorithm modifications alters the classifier learning procedure itself [FGG$^+$18]. This approach involves a deep understanding of the learning process, in order to identify the particular mechanism responsible of the class bias towards the majority class. Many popular ML methods have undergo such modifications, like support vector machines [GANAV14] and their variants [SCZ$^+$14, ML14].

Cost-sensitive methods are a variant of algorithm level modifications. In these methods, a misclassification cost is introduced in order to minimize conditional risk. With the penalization of wrong predictions, the importance of such classes is improved, pushing decision boundaries away from these instances [FGG$^+$18]. Cost-sensitive methods can be differentiated depending on which aspect of the data is affected by them: cost associated with features [ZZL16] or cost associated with classes [KSW15].

Along from these three categories, ensemble methods can be classified into their own category [GFB$^+$11]. Because of their accuracy orientation, ensembles cannot be directly applied to imbalanced datasets, since the base classifiers will ignore the minority class. However, their combination with other techniques that tackles the class imbalance problem, usually data level approaches, can improve ensemble performance. The addition of data level approaches to an ensemble is done by the application of one (or several) data preprocessing method to the data before the application of the ML technique.

As stated earlier, data level modifications are carried out by the use of different data preprocessing methods. In the literature, data preprocessing methods for imbalanced classification can be divided into three categories: oversampling, undersampling, and hybrid approaches. The former replicates the minority class until a certain balance is reached. On the contrary undersampling methods delete examples from the majority class until both classes are equal. Finally, hybrid approaches combine the previous two techniques, usually starting with an oversampling the data, followed by an undersampling step that removes samples from both classes, in order to remove noisy instances and improve the performance of the ML technique used.

The most popular and widely used method for oversampling data is known as Random OverSampling (ROS) [BPM04]. ROS randomly replicates examples from the minority class, until the number of instances of both classes is the same. On the other hand, Random UnderSampling (RUS) randomly removes instances from the majority class, until both classes have the same amount of instances [BPM04]. The downside of RUS method, is that it removes information, and may remove key instances for the learning process to distinguish between both classes.

The Synthetic Minority Oversampling TEchnique (SMOTE) is a more advanced oversampling method [FGHC18]. It adds synthetically created instances from the minority class until both classes have the same number of instances. Those new synthetic instances are created by the interpolation of several minority class instances that belong to the same neighborhood. SMOTE calculates the k nearest neighbors of each minority class instance. Then, in the segment that connects every instance with its k closest neighbors, a synthetic instance is randomly created.

Clustering solutions have also been proposed for tackling imbalanced data classification [LTHJ17]. It has been employed effectively for the data imbalance problem as a way to increase the density of points belonging to certain neighborhoods [NPF18]. Clustering methods balance the data by localizing groups of instances belonging to different neighborhoods, and then applying a data sampling technique [Kra16]. These methods can improve the performance of simple data sampling techniques like ROS and RUS, since they will either create or remove data points located in key areas, improving the later ML process. Clustering can also be combined with ensembles to obtain diverse and balanced classifiers, improving the performance of ensemble approaches that uses data sampling [LTHJ17, ZWL+18].

Performance evaluation is a key factor for assessing the classification performance. In binary classification problems, the confusion matrix (shown in Table I.1) collects correctly and incorrectly classified examples from both classes.

Table I.1: Confusion Matrix for Binary Classification Problems

	Positive Prediction	Negative Prediction
Positive class	True Positive (TP)	False Negative (FN)
Negative class	False Positive (FP)	True Negative (TN)

Traditionally, accuracy (Equation I.1) has established as the most extended classification performance metric. However, it is not a valid metric when dealing with imbalanced dataset, since it will bias towards the majority class, and not show the real classification performance. Furthermore, it can lead to wrong conclusions: a dataset with an imbalance ratio (IR) of 9 (9 majority class instances per each minority class example) and a global accuracy of 90% will be classifying all examples as belonging to the majority class.

$$Acc = \frac{TP + TN}{TP + FN + FP + TN} \tag{I.1}$$

The Geometric Mean (GM) is described in Equation I.2. GM metric attempts to maximize both minority and majority classes accuracy at once [BSGR03]. The accuracy of both minority and majority classes is represented by the True Positive Rate (TPR) = $\frac{TP}{TP+FN}$ and True Negative Rate (TNR) = $\frac{TN}{TN+FP}$.

$$GM = \sqrt{TPR * TNR} \tag{I.2}$$

Another popular and extended evaluation metric for imbalanced data classification is the Area Under the Curve (AUC) [HL05]. This metric combines the classification performance of both minority and majority classes It shows the trade-off between the TPR and False Positive Rate (FPR). This metric provides a single measure of a classifier performance, compared against a random classifier.

Due to the automation in data acquisition, Big Data scenarios pose a new challenge in terms of imbalanced data distributions. This Big Data scenario also brings new problems to classic data sampling and balancing methods, since they are not prepared to work with such amounts of data. Moreover, the high data redundancy present in Big Data problems, will hinder the performance of data sampling techniques and, therefore, prevent the transition from Big to Smart Data. Because of their proved performance, proposals focused on creating ensembles using smart and diverse datasets for the classification of imbalanced Big Data are needed for the correct identification of both classes.

3 Justification

From previous sections, we may assert that there is an increasing gap between the storing and processing capabilities of current systems. New tools for processing this data are emerging nowadays. However, we need the proper algorithms, capable of dealing with Big Data problems, for improving or even enabling the ML task. There is an increasing need for scalable and distributed ML proposals.

Ensembles have received little attention within the Big Data environment. Their ability to correct errors through different base learners can be of great help in Big Data problems, where the variety of data can hinder the DM task. Since traditional data preprocessing methods are not able to handle Big Datasets, achieving the required level of diversity has become a challenge due to the complexity and size of Big Data problems.

In order to extend the ML ecosystem, and the classification and data preprocessing based ensembles in Big Data environments, the following major issues should be properly addressed:

- Firstly, it is important to study current technologies for Big Data processing, as well as to learn about the different ML libraries available. This will allow us to decide which platforms suits our necessities better according to our previous requisites.
- In Big Data, there are few ensemble methods devoted to classification tasks. Only classic ensembles have been adapted to Big Data scenarios. The unsuitability of classic data preprocessing methods for Big Data have unabled reaching the required level of diversity for ensembles. There is a need for new ensemble classifiers, designed for Big Data, ready to tackle large-scale problems.
- Data preprocessing techniques have shown their limitations with the ever-growing size of the data. Therefore, the study and design of scalable and distributed methods for improving the quality of such data in order to achieve Smart Data will be needed. We will focus on the most common problem affecting data, the presence of noise. Ensembles based on data partitions have proved to be able to effectively remove noise from normal-sized data. This study will conform the first work in this field for Big Data problems.
- With the acquired knowledge in ensembles for Big Data, it is necessary to go deeper into this topic, and to focus into one problem that hinders the performance of DM methods. Imbalanced classification is a very pernicious problem affecting Big Data. The application of data preprocessing has been extensively employed for this problem for normal-sized data. However, a Smart Data based ensemble focused on imbalanced Big Data classification would be of great interest.

All these issues can be encompassed within the subject of this thesis: The development of scalable ensemble methods for Big Data classification and Smart Data extraction.

4 Objectives

After introducing the main concepts related to this work, we present the main objectives that have driven this thesis. They include the study and analysis of Big Data, classifiers and data preprocessing based ensembles, as well as the development of parallel and distributed ensemble algorithms for Big Data environments (e.g. Apache Spark).

Specifically, the main objective of this thesis is the analysis, design, implementation and evaluation of scalable classifier and data preprocessing ensembles for Big Data and Smart Data extraction. In the following list, we describe each objective individually:

- **To study the current state on scalability of Big Data frameworks**: an analytical and experimental study of current popular Big Data frameworks which will allow us to extend our knowledge about the suitability and performance of current platforms and strategies. The end objective is to provide an introduction of the similarities and differences among Big Data frameworks.

- **To study the current state-of-the-art in Big Data preprocessing and Smart Data**: a theoretical and empirical study of the current state-of-the-art on Big Data preprocessing as a way to achieve Smart Data. This study will uncover all strengths and weaknesses of current developments in these areas. To the best of our knowledge, there is no study in the literature that provides such knowledge.

- **To provide a distributed data preprocessing-based ensemble method for classification**: data preprocessing is known to be capable of achieve the required level of diversity, and to being directly applied to ensemble methods. The goal is to create a data preprocessing-based ensemble method for standard Big Data classification problems. This method will be focused on creating highly diverse data through the use of different data preprocessing techniques.

- **To address the problem of noise in Big Data scenarios**: concretely, the aim here is to prove that noise in Big Data cannot be disregarded, and to provide the first noise filtering algorithms for Big Data scenarios. The resulting noise filtering algorithms will be capable of efficiently remove noisy instances in Big Data environments, and to enable Smart Data. Furthermore, we aim to analyze two different types of noise filters: partition-based, and iterative algorithms.

- **To tackle the imbalanced Big Data classification problem using data preprocessing for enabling Smart Data based ensembles**: after the study of ensembles for standard classification, our objective is to focus on the imbalanced Big Data classification problem. Imbalanced classification is a very common problem when dealing with Big Data. Our aim is to design a scalable Smart Data based ensemble classifier, able to effectively classify imbalanced Big Data datasets.

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 for the research work and experiments will be applied:

1. **Observation**: thorough study of the large-scale ML problem and its specific characteristics, along with the application of ensemble techniques to different stages of the KDD process, as well as the possibilities offered by distributed computing technologies to give a proper solution to this problem.

2. **Hypothesis formulation**: design of new classifier and data preprocessing methods, based on ensembles, for improving the quality of the data and the later DM task. Methods developed must comply with the objectives previously mentioned in order to properly tackle Big Data problems and to achieve Smart Data.

3. **Observation gathering**: retrieving the results obtained by the application of the proposed new methods on real-world Big Data datasets. Both efficiency and accuracy, among other metrics, have to be measured and considered in the designs.

4. **Contrasting the hypothesis**: comparison of the results obtained by the classification and data preprocessing ensemble algorithms in order to analyze the quality of the new proposals. For that purpose, we will rely on scalable ML libraries, such as MLlib of FlinkML. Other proposals in the literature will serve to validate effectiveness and efficiency of our models.

5. **Hypothesis proof or refusal**: Acceptance or rejection and modification, in due case, of the developed techniques as a consequence of the performed experiments and the gathered results. If necessary, the previous steps should be repeated to formulate new hypothesis that can be proven.

6. **Scientific thesis**: Extraction, redaction and acceptance of the conclusions obtained through out the research process. All the proposals and results gathered along the entire process should be synthesized into a memory of the thesis.

6 Summary

This section presents a summary of the publications associated to this thesis. After that, in Section 7 we describe the main results obtained by these proposals. Both the research carried out for this thesis and the associated results are collected into the published journal publications listed below:

- 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 Analytics 2 (1), 1 (2017). DOI: `https://doi.org/10.1186/s41044-016-0020-2`
- D. García-Gil, S. Ramírez-Gallego, S. García, F. Herrera. Principal Components Analysis Random Discretization Ensemble for Big Data. Knowledge-Based Systems 150, 166-174 (2018). DOI: `https://doi.org/10.1016/j.knosys.2018.03.012`
- D. García-Gil, J. Luengo, S. García, F. Herrera. Enabling Smart Data: Noise Filtering in Big Data Classification. Information Sciences 479, 135-152 (2019). DOI: `https://doi.org/10.1016/j.ins.2018.12.002`
- D. García-Gil, F. Luque-Sánchez, J. Luengo, S. García, F. Herrera. From Big to Smart Data: Iterative Ensemble Filter for Noise Filtering in Big Data classification. International Journal of Intelligent Systems 34 (12), 3260-3274 (2019). DOI: `https://doi.org/10.1002/int.22193`
- D. García-Gil, J. Holmberg, S. García, N. Xiong, F. Herrera. Smart Data based Ensemble for Imbalanced Big Data Classification. Submitted.

The remainder of this section is organized following the objectives presented in Section 4 and these publications. First, Section 6.1 provides some insights about the comparison on scalability performed on two Big Data frameworks. Section 6.2 details a data preprocessing ensemble classifier for Big Data. Section 6.3 present the first two proposals in the literature for efficiently and effectively removing noise in Big Data problems. Finally, Section 6.4 describes a data preprocessing ensemble classifier designed to face Big Data imbalanced problems.

6.1 Comparison on Scalability between Big Data Frameworks

With the rise of Big Data, many tools have emerged ready to provide users efficient tools to deal with it. All of these tools have one thing in common, the use of the MR programming paradigm. This paradigm allows to process huge amounts of data using a divide and conquer approach. By means of two simple functions, namely Map and Reduce, any algorithm can be adapted to work with Big Data datasets in a transparent way for the programmer. MR also provides the fundamental fault-tolerance scheme. The most popular open-source framework was Apache Hadoop, however, newer and more efficient implementations of the MR paradigm have emerged recently, tackling Apache Hadoop limitations.

In this work, we performed a thorough comparison between two outstanding Big Data frameworks, Apache Spark and Apache Flink. We divide this study in two parts: (1) a theoretical comparison between both of their engines and ML algorithm implementations, and (2) a practical study on scalability.

We focused on their batch data processing capabilities. Their main differences and analogies are studied in order to outline which are the best scenarios for each platform. Moreover, we studied their corresponding ML libraries, MLlib & ML for Apache Spark (RDD and Dataset based), and FlinkML for Apache Flink.

A practical study on scalability was performed, executing the same ML algorithm on both platforms in order to assess the differences and similarities in performance among them.

The work associated to this part is:

D. García-Gil, S. Ramírez-Gallego, S. García, F. Herrera. Big Data Analytics 2 (1), 1 (2017). A comparison on scalability for batch big data processing on Apache Spark and Apache Flink. DOI: `https://doi.org/10.1186/s41044-016-0020-2`

6.2 Ensemble Classification for Big Data

Classification ensembles are methods that combine a set of base classifiers to obtain better predictions. These base classifiers are basic and simple classifiers, that can correct errors across many iterations. Ensembles require diversity in the data in order to obtain diverse base classifiers. This diversity is usually introduced through data preprocessing. With small changes in input data, diverse decision trees are created and better ensembles are obtained. Despite the importance of classification ensemble methods, few proposals have been made for Big Data problems. Specially scarce is the impact of data preprocessing in Big Data classification ensembles.

In order to fill this gap, we proposed a new ensemble classifier for Big Data using Apache Spark. This method is inspired by Random Projection Random Discretization Ensemble (RPRDE) by Ahmad and Brown [AB13]. It is a highly diverse ensemble method focused on data level diversity composed of two data preprocessing methods, Random Discretization (RD) and Random Projections (RP). RD method discretizes the data in k intervals by randomly selecting $k-1$ instances, and using those values as thresholds for the discretization of each feature. On the other hand, RP projects the data onto a lower d-dimensional subspace. However, Random Projections (RP) [JL84] are known to suffer from a gradual degradation in performance as the projected dimensions drops below $\log k$ [Das00]. Moreover, RP is highly unstable, different RP will lead to radically different results [FM03].

Our main objective was to provide a data preprocessing ensemble, capable of working with huge amounts of data efficiently. Our proposal is denoted by Principal Components Analysis Random Discretization Ensemble (PCARDE). PCARDE tackles RPRDE drawbacks by using a more informative and less randomized method like Principal Components Analysis (PCA) [Jol11]. As stated previously, ensembles require diversity, however, PCA always produces the same results for a fixed number of principal components. In order to achieve the desired level of diversity, we introduced randomization in the number of principal components we select. For each iteration of the ensemble, we select the number of principal components randomly in the interval $[1, m-1]$ (m number of features).

In order to measure the performance of PCARDE, we evaluated it on a conscious experimental framework, composed by five Big Data datasets, different ensemble classifiers with different sizes, a set of non-parametric and Bayesian statistical tests. Additionally, a study concerning the impact of the proposed addition was carried out.

The journal contribution associated to this part is:

D. García-Gil, S. Ramírez-Gallego, S. García, F. Herrera. Principal Components Analysis Random Discretization Ensemble for Big Data. Knowledge-Based Systems 150, 166-174 (2018). DOI: `https://doi.org/10.1016/j.knosys.2018.03.012`

6.3 The Problem of Noise in Big Data

Advanced Big Data modeling and analytics are indispensable for discovering the underlying structure from retrieved data. However, imperfections in the data will disrupt all posterior KDD phases. Among all the problems that may appear in the data, the presence of noise in the dataset is one of the most frequent. In Big Data, the high redundancy of the instances and high dimensional problems pose new challenges to classic noise preprocessing algorithms. While some architectural designs are already proposed in the literature [Zer16], there is no particular algorithm which deals with noise in Big Data classification, nor a comparison of its effect on model generalization abilities or computing times.

In view of this lack of algorithms for noise filtering in Big Data problems, we proposed a framework for Big Data under Apache Spark. This framework is composed of two algorithms based on ensembles of classifiers for removing noisy examples. Both of them are based on a partitioning scheme of the dataset. The former partitions the data using a k-fold strategy and learns a Random Forest, each time leaving out the test partitions. The result are k classifiers which are employed to predict the data. Instances that do not match their prediction are considered as noise and removed. The later ensemble noise filter differs from the first in the classifiers used. It learns three different classifiers: a Random Forest, KNN and a Logistic Regression. Then, each of the k partitions are predicted with the three classifiers. Using a voting strategy instances are either kept, or considered as noise and removed.

The former method of the proposed framework is denoted by Homogeneous Ensemble for Big Data (HME-BD), the later is called Heterogeneous Ensemble for Big Data (HTE-BD).

A simple filtering algorithm based on KNN is also proposed in the framework for comparison purposes. This algorithm, namely ENN-BD, removes noisy instances by comparing the label of each example with its closest neighbor. If the labels are different, the instance is considered as noise and removed.

The extensive empirical study of the framework consists of four Big Data datasets, five different levels of noise, two different classifiers, a study of the number of instances left after the noise filtering process, a comparison of the percentage of correctly removed instances, a computing times comparative, and an analysis based on non-parametric and Bayesian statistical tests.

With the proposed framework serving as a baseline, our next target was to improve the noise filtering ecosystem in Big Data with more powerful algorithms.

Iterative algorithms for noise cleaning have been used extensively form normal-sized problems showing promising results [SGLH16]. Taking into account Apache Spark's capacity for iterative algorithms, we proposed an iterative ensemble filter for noise filtering in Big Data classification. The algorithm proposed, similarly to the previously proposed framework, is based on a partitioning scheme of the data. It performs a k-fold to the data, and learns a Random Forest on each *training* partition. The complete dataset is predicted k times using each of the models learned. All instances will have k predictions. Using a voting strategy, instances are either kept or considered as noise and removed. This process is repeated a number of times using the newly generated dataset in an iterative fashion.

This distributed approach, called Iterative Ensemble Filter for Big Data (IEF-BD), was tested on six Big Data datasets, with five different levels of noise. Additionally, reduction rates and computing times of IEF-BD were tested.

The journal papers associated to this part are:

D. García-Gil, J. Luengo, S. García, F. Herrera. Enabling Smart Data: Noise Filtering in Big Data Classification. Information Sciences 479, 135-152 (2019). DOI: `https://doi.org/10.1016/j.ins.2018.12.002`

D. García-Gil, F. Luque-Sánchez, J. Luengo, S. García, F. Herrera. From Big to Smart Data: Iterative Ensemble Filter for Noise Filtering in Big Data classification. International Journal of Intelligent Systems 34 (12), 3260-3274 (2019). DOI: `https://doi.org/10.1002/int.22193`

6.4 Imbalanced Big Data Classification

Uneven class distribution is present in many of today's problems. In Big Data, the automation in data acquisition have worsened this problem. Many efforts have been devoted to tackle the class imbalanced problem for normal-sized data. Data preprocessing is the most common set of techniques employed for this task. Data sampling solutions balance the dataset through the addition or removal of data points until the classes are balanced to a desired level. In Big Data, there is little research devoted to handle irregular binary class distributions. Given the good performance of data preprocessing when joined with ensemble techniques, a further study in imbalanced ensembles can be of great interest.

Then we proposed a Smart Data based ensemble, capable of dealing with imbalanced binary class problems. This ensemble is based on the same data preprocessing carried out by PCARDE, plus a novel clustering-based data balancing methodology. Hierarchical clustering is employed with a random number of clusters each iteration of the ensemble. Clusters found are individually balanced using ROS technique. This results in a balanced and diverse dataset. Our proposal is denoted by Imbalanced Classification Ensemble for Big Data (ICE_BD).

The empirical study consists of 21 imbalanced Big Data datasets, three data balancing methods, three different classifiers for comparison, and two imbalanced data classification metrics.

The paper associated to this part is:

D. García-Gil, J. Holmberg, S. García, N. Xiong, F. Herrera. Smart Data based Ensemble for Imbalanced Big Data Classification. Submitted.

7 Discussion of Results

The following subsections present the main results and further discussion motivated by the research conducted in this thesis.

7.1 Comparison on Scalability between Big Data Frameworks

This comparison has been devoted to analyze the similarities and differences among two novel Big Data frameworks, Apache Spark and Apache Flink. The aim of this work is to assess the performance these two frameworks, using their corresponding ML libraries for batch Big Data processing.

To obtain well-funded conclusions about these two Big Data frameworks, we divided the comparison into two parts: an internal comparison of their engines, and a experimental comparative of their ML performance.

Regarding the engine-level comparison, the main differences founded are that Apache Spark keeps data in memory across iterations through an explicit caching. However, Apache Spark plans its executions as acyclic graph plans, which implies that it needs to schedule and run the same set of instructions in each iteration. In contrast, Apache Flink implements a thoroughly iterative processing in its engine based on cyclic data flows (one iteration, one schedule). Additionally, it offers delta iterations to leverage operations that only changes part of data.

As stated, we also performed a thoroughly experimental evaluation of the performance of both frameworks. For this comparison, we use the same Big Data dataset for both frameworks, which where running on the same hardware. According to experiments, MLlib performs slightly better than Spark ML. However, there is a big difference among Apache Spark and Apache Flink. Depending on the ML algorithm, time difference between Apache Spark and Apache Flink can go from 2.5x up to 10x times less time for Apache Spark.

7.2 Ensemble Classification for Big Data

In the corresponding section, we have proposed a new and distributed classification ensemble for Big Data inspired by Ahmad and Brown RPRDE ensemble. The aim of this work is provide a data preprocessing ensemble, adapted to Big Data frameworks and able to face Big Data problems efficiently.

Our first approach to improve the dimensionality reduction step of RPRDE was to replace the RP method with a more informed one like χ^2 [LS95]. We use χ^2 for performing feature selection. Like PCA, χ^2 also produces the same results for a fixed number of features. The solution was the same applied to PCA, introduce randomization in the number of features we compute in order to obtain diverse decision trees.

The experimental study carried out to measure the impact of PCA justified the addition of PCA instead of RP, showing the differences in feature selection among χ^2 and our proposal, called PCARDE. This study was divided in two parts: a study of the most important features selected by PCA and χ^2, and an accuracy comparison of a decision tree using PCA, χ^2 and RP with the same increasing number of features selected. The results showed that PCA and χ^2 are selecting very different features among them. This led to big differences in accuracy, where PCA showed to be the best performing method among the three feature selectors tested.

According to the experiments, our proposal is able to achieve better accuracy for every tested dataset, achieving up to 10% more accuracy. PCARDE has shown to be a very stable ensemble classifier, showing little or no improvement when using 10 trees onward. This means that PCARDE achieves the required level of diversity in a short amount of iterations. Moreover, fewer iterations require less computing time, making it a fast and efficient solution. The computational cost of PCA only becomes noticeable in datasets with a large number of features. For datasets with small number of features, PCA can perform faster than the other methods. Positive outcomes were confirmed by a non-parametric statistical test with high confidence.

7.3 The Problem of Noise in Big Data

Here, we have introduced the first methods for dealing with noise in Big Data problems. The first method, called HME-BD, is an homogeneous partition-based ensemble. The second method is an heterogeneous partition-based method, named HTE-BD. The proposed methods follow a partitioning approach, in particular, a k-fold approach is used. The k-fold is performed to the data, and then, using the *training* partition, k models are learned depending on the method. Using these models, the partition left is predicted and instances are either kept or removed according to the comparison among their prediction and the real label.

The large experimental study carried out have drawn different conclusions. Here we highlight some:

- Avoiding the treatment of noise is never the best option and using the appropriate noise filtering technique will provide a significant improvement in accuracy.
- The classifier used after the noise filtering process have to be taken into consideration for choosing the best noise filtering strategy. Decision trees are affected by aggressive filters, since they are able to withstand small amounts of noise while exploiting the clean instances. On the other hand, KNN is very affected by the noisy instances left.
- Different number of partitions for the internal k-fold have shown to have little impact in the global accuracy values and no impact in the number of removed instances. This proves that proposed methods are robust.
- The voting strategy has a huge impact in the number of removed instances by HTE-BD. This is an expected behavior, since conservative strategies will tend to keep more instances.
- The best performing method for both the decision tree and KNN has been HME-BD. A set of non-parametric and Bayesian statistical tests have proved that it performed statistically better than HTE-BD. It is also the most efficient solution in terms of computing times. The use of KNN hinders the computation performance of HTE-BD.

In a further extension we introduced an iterative ensemble filter for Big Data noise filtering, called IEF-BD. It is a noise filter born from the combination of the partitioning strategy of the aforementioned HME-BD, and the iterative nature of popular noise filters for normal-sized problems.

The experimental study carried out on six Big Data datasets proved that IEF-BD not only outperforms HME-BD in terms of accuracy, but also enables the use of noise filters in datasets where HME-BD is not able to improve the baseline accuracy without any data preprocessing. The voting strategy of IEF-BD has a great impact in both the accuracy, and number of instances removed.

Regarding computing times, IEF-BD have shown to be more computationally demanding than HME-BD due the learning of larger models and the prediction of the entire dataset.

Both of these approaches have constituted the first suitable proposals for noise filtering in Big Data domains, showing that data preprocessing in general, and noise filtering in particular, cannot be disregarded, and that a good noise filtering strategy can greatly improve the performance of models learned from such data. Moreover, both of the proposals enable to reach Smart Data from raw and low-quality Big Data.

7.4 Imbalanced Big Data Classification

In this section, we have presented a new ensemble for Big Data imbalanced classification based on clustering and Smart Data based technologies, called ICE_BD. The proposed algorithm follow the same diversity mechanisms introduced with PCARDE. The data balancing phase is carried out by hierarchical clustering for the localization of neighborhoods. Those regions of the data, are balanced using ROS technique. The final result is an ensemble built on diverse, balanced and *smart* data.

The extensive experimental study justifies the combination of RD and PCA preprocessing for creating diverse data, and the proposed hierarchical clustering balancing method. The study was focused on the analysis of performance using specific imbalanced data classification metrics. Concretely, we tested the three classifiers using GM and AUC metrics. We compared ICE_BD against a decision tree, Random Forest and PCARDE, all of them with and without any data balancing technique. For the data balancing techniques, we used the most popular and widely used methods: RUS, ROS and SMOTE.

This experimental study allowed us to show that ICE_BD is a suitable method for imbalanced Big Data classification. It is able to deal with datasets with any IR in a reasonable amount of time. ICE_BD achieved the best results in both GM and AUC. On average, our proposal achieved a 5% better performance on both GM and AUC metrics than its closest competitor. ICE_BD has also proved to be able to handle Big Data datasets in a timely manner.

8 Concluding Remarks

In this thesis, we have addressed several problems, focusing on a common objective: the analysis, design, implementation and evaluation of scalable classifier and data preprocessing ensembles for Big Data and Smart Data extraction.

Our first objective was focused on analyzing and gaining full understanding of the current technologies developed for the Big Data scenario, as well as their similarities and differences in terms of performance. Results showed that one of the frameworks analyzed had an outstanding performance, and thus we were encouraged to shift towards it as the main framework for the developing of the rest of the proposals. Additionally, the existence of an open and public library for this framework, were researches can share their work, is an important feature for contributing to the open-source community.

For the second objective of our research line, a novel ensemble method for Big Data classification is proposed, namely PCARDE. PCARDE is a new scalable and distributed ensemble method based on PCA for the dimensionality reduction step and RD, capable of working with Big Data problems. Experimental results have demonstrated the exceptional stability and prediction accuracy of PCARDE, with respect to other classic classification ensembles.

The third objective was geared towards Smart Data extraction. The problem of noise is a crucial step in transforming raw data into quality Smart Data, especially in Big Data scenarios. Our first approach to this problem constituted the first suitable noise filter in Big Data domains, where the high redundancy of the instances and high dimensional problems pose new challenges to classic noise preprocessing algorithms. We have proposed several noise filtering algorithms, based on the creation of ensembles of classifiers. Different strategies of data partitioning and ensemble classifier combination have led to three different approaches: HME-BD, HTE-BD and ENN-BD. Then, we took a step forward and proposed an iterative ensemble noise filter for noise-cleaning in Big Data classification, IEF-BD. Our proposed noise filter cleans the data in an iterative fashion, using a data partitioning strategy. Experimental outcomes validated the performance of IEF-BD, showing that the proposed technique is able to efficiently detect and remove the noisy instances in Big Data datasets, producing Smart Datasets.

Our last objective, and bearing the previous knowledge in mind, was focused on a special case of classification, which is becoming more common in Big Data scenarios, the imbalanced Big Data classification. A Smart Data based ensemble for dealing with the imbalanced class classification problem in Big Data is presented, called ICE_BD. It combines the use of RD and PCA, with a novel solution for data balancing based on a combination of hierarchical clustering and ROS. Empirical outcomes acknowledge the efficiency when dealing with Big Data imbalanced datasets of this method, proving superior performance on both GM and AUC metrics.

Conclusiones

En esta tesis se han abordado varios problemas, centrados en un objetivo común: el análisis, diseño, implementación y evaluación de métodos de ensemble escalables para clasificación y preprocesamiento de datos para Big Data y extracción de Smart Data.

El primer objetivo estaba centrado en analizar y comprender en profundidad las actuales tecnologías desarrolladas para entornos Big Data, así como sus similaridades y diferencias en términos de rendimiento. Los resultados mostraron que una de estas herramientas analizadas obtuvo un rendimiento excepcional, y esto nos motivó a elegirla como la herramienta principal para el desarrollo del resto de las propuestas. Adicionalmente, la existencia de una librería abierta y pública donde los investigadores pueden compartir su trabajo es una característica importante a tener cuenta a la hora de colaborar con la comunidad de software libre.

Para el segundo objetivo de nuestra línea de investigación, se ha propuesto un novedoso método de ensemble para clasificación de problemas Big Data, llamado PCARDE. PCARDE es un nuevo método de ensemble escalable y distribuido, basado en PCA para la fase de reducción de dimensionalidad, y RD, capaz de abordar problemas Big Data. Los resultados experimentales han demostrado la excepcional estabilidad y precisión de PCARDE a la hora de predecir datos, con respecto a otros métodos de ensemble clásicos.

El tercer objetivo estaba centrado en la extracción de Smart Data. El problema de ruido en los datos es un paso crucial a la hora de transformar datos en crudo en datos de calidad o Smart Data, especialmente en entornos Big Data. Nuestro primer acercamiento a este problema ha supuesto el primer filtro de ruido adaptado a problemas Big Data, donde la alta redundancia y dimensionalidad de los datos plantean un reto a los algoritmos clásicos de preprocesamiento de datos para la limpieza de ruido. Hemos propuesto varios algoritmos de filtrado de ruido, basados en la creación de ensembles de clasificadores. El uso de diferentes estrategias de particionamiento de datos, así como de combinaciones de clasificadores del ensemble, nos ha conducido a tres soluciones diferentes: HME-BD, HTE-BD y ENN-BD. Teniendo esto en cuenta, se ha ido un paso más allá y hemos propuesto un ensemble iterativo para el filtrado de ruido en clasificación de Big Data, llamado IEF-BD. Este filtro de ruido limpia los datos de forma iterativa, usando una estrategia de particionamiento de datos. Los resultados experimentales han validado el buen redimiento de IEF-BD, mostrando que el método propuesto es capaz de detectar y eliminar instancias ruidosas en Big Data de forma eficiente, generando Smart Data.

Nuestro último objetivo, y teniendo en cuenta el conocimiento previo, ha estado centrado en un caso especial de clasificación que se está volviendo más común en escenarios Big Data: la clasificación desbalanceada. Se ha presentado un ensemble basado en Smart Data para tratar el problema de la clasificación con clases desbalanceadas en Big Data. Esta propuesta combina el uso RD y PCA con una novedosa solución para el balanceo de datos, basada en la combinación de clustering jerárquico y ROS. El estudio empírico ha verificado la eficacia de este método cuando trata con problemas Big Data desbalanceados, demostrando un rendimiento superior al resto con respecto a las métricas GM y AUC.

9 Future Work

From the conclusions drawn from this thesis, new and promising lines of research can be proposed. They aim at improving existing models, and to address new problems that are emerging of surely will do from the evolving Big Data scenario.

- **Study of other methods for ensemble diversity**: different methods for adding diversity to ensemble methods are available for normal-sized problems. Concretely, data level diversity through data preprocessing techniques have shown to achieve great performance. Among the extensive list of data preprocessing methods with a random component, only a handful of techniques are available in Big Data. New data preprocessing methods for Big Data problems, focused on creating diverse ensembles can be an interesting topic [MCJ+19].

- **Noise filtering for imbalanced classification**: the presence of noise in imbalanced problems is a challenging task, since the removal of minority class instances is a risky process. Noise filters for imbalanced data classification have been studied for normal-sized data, but not so much for Big Data environments [FGHC18]. Noise filters which prioritize the removal of instances from the majority classes can be an interesting topic for multiclass or imbalanced Big Data problems [RTR18].

- **Removing versus relabeling of noisy instances**: it is stated that the removal of mislabeled instances is more efficient than repairing and/or relabeling them [FV14, LJX19]. However, as we have seen, an excessive removal of instances can cause a serious and irremediable loss of information. Noise measures are devoted to give a score to selected instances in order to decide if such instances have to be either removed, relabeled or maintained. This will lead to a more fine-grained noise removal, and to achieve higher quality data. The study and implementation of novel noise measures or scores in Big Data domains, able to verify with a high degree of confidence if an instance is noisy, and which is the right class, will be of great interest [LSA+18].

- **Smart Data for non-standard supervised problems**: DM is a field subdivided in different areas, such as supervised and unsupervised learning. However, there are many other lesser known learning problems. These problems are known as non-standard supervised problems [CCGH19]. Among all different types of non-standard supervised problems, online learning is gaining lots of attention in the last years. Technologies such as the internet have increased the necessity of data stream processing for analyzing and extracting all valuable insights from their continuously generated data. As stated earlier, for achieving Smart Data, the data needs to be perfected and cleaned. Smart Data extraction in online learning problems is still in its early days. New and more powerful methods, capable of achieving Smart Data from data streaming problems, and to deal with unsolved challenges so far need to be designed [RGKG+17].

- **Noise filtering in Federated Learning**: Federated Learning is a new and distributed ML approach that enables training on a wide set of decentralized data residing on different devices, such as mobile phones [MMR+17]. The main idea is that users download a global model to their devices, and improve it by learning from their own private data. Then, changes are summarized and sent to the cloud, where it is averaged with other users updates to improve the global model [KMA+19]. Federated Learning makes emphasis on mobile and edge device computing, with a special concern in data privacy, and has been successfully combined with

Deep Learning neural networks. The application of noise filtering techniques to Federated Learning problems can have many advantages. Local and private device data can be cleansed using the information provided by the global model. This will ultimately lead to higher quality local data and to take better decisions.

- **Unsupervised anomaly detection in Big Data**: anomaly detection can be defined as the process of identifying unexpected events, or events that differ form the normal behavior. It is often applied on unlabeled data, taking only into consideration the relation among the features [GU16]. There is a long list of real-world applications of unsupervised anomaly detection techniques, such as network intrusion, video surveillance, fraud detection, and also in life science and medical domains. Ensemble approaches for unsupervised anomaly detection have been employed with great success [HZW$^+$19]. With the ever-growing size of data, classic methods for unsupervised anomaly detection have troubles dealing with such amounts of data. Distributed methods, prepared for detecting anomalies on unsupervised real-time Big Data problems can be of great interest.

Chapter II

Publications

1 A comparison on scalability for batch big data processing on Apache Spark and Apache Flink

- D. García-Gil, S. Ramírez-Gallego, S. García, F. Herrera. Big Data Analytics 2 (1), 1 (2017).
  - Status: **Published**.
  - Impact Factor (JCR 2017): Not indexed.

A Comparison on Scalability for Batch Big Data Processing on Apache Spark and Apache Flink

Diego García-Gil[*]
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
djgarcia@decsai.ugr.es

Sergio Ramírez-Gallego
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
sramirez@decsai.ugr.es

Salvador García
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
salvagl@decsai.ugr.es

Francisco Herrera
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
herrera@decsai.ugr.es

Abstract

The large amounts of data have created a need for new frameworks for processing. The MapReduce model is a framework for processing and generating large-scale datasets with parallel and distributed algorithms. Apache Spark is a fast and general engine for large-scale data processing based on the MapReduce model. The main feature of Spark is the in-memory computation. Recently a novel framework called Apache Flink has emerged, focused on distributed stream and batch data processing. In this paper we perform a comparative study on the scalability of these two frameworks using the corresponding Machine Learning libraries for batch data processing. Additionally we analyze the performance of the two Machine Learning libraries that Spark currently has, MLlib and ML. For the experiments, the same algorithms and the same dataset are being used. Experimental results show that Spark MLlib has better perfomance and overall lower runtimes than Flink.

Keywords Big Data · Spark · Flink · MapReduce · Machine Learning

Introduction

With the always growing amount of data, the need for frameworks to store and process this data is increasing. In 2014 IDC predicted that by 2020, the digital universe will be 10 times as big as it was in 2013, totaling an astonishing 44 zettabytes [1]. Big Data is not only a huge amount of data, but a new paradigm and set of technologies that can store and process this data. In this context, a set of new frameworks focused on storing and processing huge volumes of data have emerged.

[*]Corresponding author.

MapReduce [2] and its open-source version Apache Hadoop [3, 4] were the first distributed programming techniques to face Big Data storing and processing. Since then, several distributed tools have emerged as consequence of the spread of Big Data. Apache Spark [5, 6] is one of these new frameworks, designed as a fast and general engine for large-scale data processing based on in-memory computation. Apache Flink [7] is a novel and recent framework for distributed stream and batch data processing that is getting a lot of attention because of its streaming orientation.

Most of these frameworks have their own Machine Learning (ML) library for Big Data processing. The first one was Mahout [8] (as part of Apache Hadoop [3]), followed by MLlib [9] which is part of Spark project [5]. Flink also has its own ML library that, while it is not as powerful or complete as Spark's MLlib, it is starting to include some classic ML algorithms.

In this paper, we present a comparative study between the ML libraries of these two powerful and promising frameworks, Apache Spark and Apache Flink. Our main goal is to show the differences and similarities in performance between these two frameworks for batch data processing. For the experiments, we use two algorithms present in both ML libraries, Support Vector Machines (SVM) and Linear Regression (LR), on the same dataset. Additionally, we have implemented a feature selection algorithm to compare the different functioning of each framework.

Background

In this section, we describe the MapReduce framework and two extensions of it, Apache Spark and Apache Flink.

MapReduce

MapReduce is a framework that has supposed a revolution since Google introduced it in 2003 [2]. This framework processes and generates large datasets in a parallel and distributed way. It is based on the Divide and Conquer algorithm. Briefly explained, the framework splits the input data and distributes it across the cluster, then the same operation is performed on each split in parallel. Finally, the results are aggregated and returned to the master node. The framework manages all the task scheduling, monitoring and re-executing in case of failed tasks.

The MapReduce model is composed of two phases: Map and Reduce. Before the Map operation, the master node splits the dataset and distributes it across the computing nodes. Then the Map operation is performed to every key-value pair to the node local data. This produces a set of intermediate key-value pairs. Once all Map tasks have finished, the results are grouped by key and redistributed so that all pairs belonging to one key are in the same node. Finally, they are processed in parallel.

The Map function takes data structured in <key, value> pairs as input and outputs a set of intermediate <key, value> pairs:

$$Map(< key1, value1 >) \rightarrow list(< key2, value2 >) \tag{1}$$

The result is grouped by key and distributed across the cluster. The Reduce phase applies a function to each list value, producing a single output value:

$$Reduce(< key2, list(value2) >) \rightarrow < key2, value3 > \tag{2}$$

Apache Hadoop [3, 4] has become the most popular open-source framework for large-scale data storing and processing based on the MapReduce model. Despite its popularity and performance, Hadoop presents some important limitations [10]:

- Intensive disk-usage
- Low inter-communication capability
- Inadequacy for in-memory computation
- Poor perfomance for online and iterative computing

Apache Spark

Apache Spark [5, 6] is a framework aimed at performing fast distributed computing on Big Data by using in-memory primitives. This platform allows user programs to load data into memory and query it repeatedly, making it a well suited tool for online and iterative processing (especially for ML algorithms). It was developed motivated by the limitations in the MapReduce/Hadoop paradigm [4, 10], which forces to follow a linear dataflow that make an intensive disk-usage.

Spark is based on distributed data structures called Resilient Distributed Datasets (RDDs) [11]. Operations on RDDs automatically place tasks into partitions, maintaining the locality of persisted data. Beyond this, RDDs are an immutable and versatile tool that let programmers persist intermediate results into memory or disk for re-usability purposes, and customize the partitioning to optimize data placement. RDDs are also fault-tolerant by nature. The lazy operations performed on each RDD are tracked using a ”lineage”, so that each RDD can be reconstructed at any moment in case of data loss.

In addition to Spark Core, some additional projects have been developed to complement the functionality provided by the core. All these sub-projects (built on top of the core) are described in the following:

- Spark SQL: introduces DataFrames, which is a new data structure for structured (and semi-structured) data. DataFrames offers us the possibility of introducing SQL queries in the Spark programs. It provides SQL language support, with command-line interfaces and ODBC/JDBC controllers.
- Spark Streaming: allows us to use the Spark's API in streaming environments by using mini-batches of data which are quickly processed. This design enables the same set of batch code (formed by RDD transformations) to be used in streaming analytics with almost no change. Spark Streaming can work with several data sources like HDFS, Flume or Kafka.
- Machine Learning library (MLlib) [12]: is formed by common learning algorithms and statistic utilities. Among its main functionalities includes: classification, regression, clustering, collaborative filtering, optimization, and dimensionality reduction. This library has been especially designed to simplify ML pipelines in large-scale environments. In the latest versions of Spark, the MLlib library has been divided into two packages, MLlib, build on top of RDDs, and ML, build on top of DataFrames for constructing pipelines.
- Spark GraphX: is the graph processing system in Spark. Thanks to this engine, users can view, transform and join interchangeably both graphs and collections. It also allows expressing the graph computation using the Pregel abstraction [13].

Apache Flink

Apache Flink [7] is a recent open-source framework for distributed stream and batch data processing. It is focused on working with lots of data with very low data latency and high fault tolerance on distributed systems. Flink's core feature is its ability to process data streams in real time.

Apache Flink offers a high 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.

It also supports both stream and batch data processing with his two main APIs: DataStream and DataSet. These APIs are built on top of the underlying stream processing engine.

Apache Flink has four big libraries built on those main APIs:

- Gelly: is the graph processing system in Flink. It contains methods and utilities for the development of graph analysis applications.
- FlinkML: this library aims to provide a set of scalable ML algorithms and an intuitive API. It contains algorithms for supervised learning, unsupervised learning, data preprocessing, recommendation and other utilities.
- Table API and SQL: is a SQL-like expression language for relational stream and batch processing that can be embedded in Flink's data APIs.
- FlinkCEP: is the complex event processing library. It allows to detect complex events patterns in streams.

Although Flink is a new platform, it is constantly evolving with new additions and it has already been adopted as a real-time process framework in many big companies, such as: ResearchData, Bouygues Telecom, Zalando and Otto Group.

Spark vs. Flink: main differences and similarities

In this section, we present the main differences and similarities in the engines of both platforms in order to explain which are the best scenarios for one platform or the other. Afterwards, we highlight the main differences between three ML algorithms implemented in both platforms: Distributed Information Theoretic Feature Selection (DITFS), SVM and LR.

Comparison between engines

The first remarkable difference between both engines lies in the way each tool ingests streams of data. Whereas Flink is a native streaming processing framework that can work on batch data, Spark was originally designed to work with static data through its RDDs. Spark uses micro-batching to deal with streams. This technique divides incoming data and processess small parts one at a time. The main advantage of this scheme is that the structure chosen by Spark, called DStream, is a simple queue of RDDs. This approach allows users to switch between streaming and batch as both have the same API. However, micro-batching may not perform quick enough in systems that requires very low latency. Nevertheless, Flink fits perfectly well in those systems as it natively uses streams for all type of workloads.

Unlike Hadoop MapReduce, Spark and Flink have support for data re-utilization and iterations. Spark keeps data in memory across iterations through an explicit caching. However, Spark plans its executions as acyclic graph plans, which implies that it needs to schedule and run the same set of instructions in each iteration. In contrast, Flink implements a thoroughly iterative processing in its engine based on cyclic data flows (one iteration, one schedule). Additionally, it offers delta iterations to leverage operations that only changes part of data.

Till the advent of Tungsten optimization project, Spark mainly used the JVM's heap memory to manage all its memory [14]. Although it is straightforward solution, it may suffers from overflow memory problems and garbage collect pauses. Thanks to this novel project, these problems started to disappear. Through DataFrames, Spark is now able to to manage its own memory stack and to exploit the memory hierarchy available in modern computers (L1 and L2 CPU caches). Flink's designers, however, had these facts into consideration from the initial point [15]. The Flink team thus proposed to maintain a self-controlled memory stack, with its own type extraction and serialization strategy in binary format. The advantage derived from these tunes are: less memory errors, less garbage collection pressure, and a better space data representation, among others.

About optimization, both frameworks have mechanisms that analyze the code submitted by the user and yields the best pipeline code for a given execution graph. Spark through the DataFrames API and Flink as first citizen. For instance, in Flink a join operation can be planned as a complete shuffling of two sets, or as a broadcast of the smallest one. Spark also offers a manual optimization, which allows the user to control partitioning and memory caching.

The rest of matters about easiness of coding and tuning, variety of operators, etc. have been omitted from this comparison as these factors do not affect the performance of executions.

A thorough comparison between algorithm implementations

Here, we present the implementation details of three ML algorithms implemented in Spark and Flink. Firstly, a feature selection algorithm implemented by us in both platforms is reviewed. Secondly, the native implementation of SVM in both platforms is analyzed. And lastly, the same process is applied for the native implementation of LR.

Distributed Information Theoretic Feature Selection

For comparison purposes, we have implemented in both platforms a feature selection framework based on information theory. This framework was proposed by Brown et al. [16] in order to ensemble multiple information theoretic criteria into a single greedy algorithm. Through some independence assumptions, it allows to transform many criteria as linear combinations of Shannon entropy terms: mutual information (MI) and conditional mutual information (CMI). Some relevant algorithms like minimum Redundancy Maximum Relevance or Information Gain, among others, are included in the framework. The main objective of the algorithm is to assess features based on a simple score, and to select those more relevant according to a ranking. The generic framework proposed by Brown et al. [16] to score features can be formulated as:

$$J = I(X_i;Y) - \beta \sum_{X_j \in S} I(X_j;X_i) + \gamma \sum_{X_j \in S} I(X_j;X_i|Y), \tag{3}$$

where the first term represents the relevance (MI) between the candidate input features X_i and the class Y, the second one the redundancy (MI) between the features already selected (in the set S) and the candidate ones, and the third one the conditional redundancy (CMI) between both sets and the class. γ represents a weight factor for CMI and β the same for MI.

Brown's version was re-designed for a better performance in distributed environments. The main changes accomplished by us are described below:

- Column-wise transformation: most of feature selection methods performs computations by columns. It implies that a previous transformation of data to a columnar format may improve the performance of further computations, for example, when computing relevance or redundancy. Accordingly, the first step in our program is aimed at transforming the original set into columns where each new instance contains the values for each feature and partition in the original set.
- Persistence of important information: some pre-computed data like the transformed input or the initial relevances are cached in memory in order to avoid re-computing them in next phases. As this information is computed once at the start, its persistence can speed up significantly the performance of the algorithm.
- Broadcast of variables: in order to avoid moving transformed data in each iteration, we persist this set and only broadcast those columns (feature) involved in the current iteration. For example, in the first iteration the class feature is broadcasted to compute the initial relevance values in each partition.

In the Flink implementation a bulk iteration process has been used to cope with the greedy process. In the Spark version, the typical iterative process with caching and repeated tasks has been implemented. Flink code can be found in the following GitHub repository: `https://github.com/sramirez/flink-infotheoretic-feature-selection`. The Spark code was gathered into a package and uploaded to the Spark's third-party package repository: `https://spark-packages.org/package/sramirez/spark-infotheoretic-feature-selection`.

Linear Support Vector Machines

Both Spark and Flink implements SVMs classifiers using a linear optimizer. Briefly, the minimization problem to be solved is the following:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} l_i\left(\mathbf{w}^T\mathbf{x}_i\right) \tag{4}$$

where $\mathbf{w}$ is the weight vector, $\mathbf{x}_i \in \mathbb{R}^d$ the data instances, λ the regularization constant, and l_i the convex loss functions. For both versions, the default regularizer is l_2-norm and the loss function is the hinge-loss: $l_i = \max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right)$

The Communication-efficient distributed dual Coordinate Ascent algorithm (CoCoA) [17] and the stochastic dual coordinate ascent (SDCA) algorithms are used in Flink to solve the previously defined minimization problem. CoCoA consists of several iterations of SDCA on each partition, and a final phase of aggregation of partial results. The result is a final gradient state, which is replicated across all nodes and used in further steps.

In Spark a distributed Stochastic Gradient Descent[2] (SGD) solution is adopted [12]. In SGD a sample of data (called mini batches) are used to compute subgradients in each phase. Only the partial results from each worker are sent across the network in order to update the global gradient.

[2]`https://en.wikipedia.org/wiki/Stochastic_gradient_descent`

Linear Regression

Linear least squares is another simple linear method implemented in Spark. Despite it was designed for regression, its output can be adapted for binary classification problems. Linear least squares follows the same minimization formula described for SVMs (see Equation 4) and the same optimization method (based on SGD), however, it uses squared loss (described below) and no regularization method:

$l_i = \frac{1}{2}(\mathbf{w}^T\mathbf{x}_i - y_i)^2$

The Flink version for this algorithm is quite similar to the one created by Spark's developers. It uses SGD to approximate the gradient solutions. However, Flink only offers squared loss whereas Spark offers many alternatives, like hinge or logistic loss.

Experimental Results

This section describes the experiments carried out to show the performance of Spark and Flink using three ML algorithms over the same huge dataset. We carried out the comparative study using SVM, LR and DITFS algorithm.

The dataset used for the experiments is the ECBDL14 dataset. This dataset was used 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. It 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: 2% of positive instances. For this problem, two pre-processing algorithms were applied. First, the Random OverSampling (ROS) algorithm used in [18] was applied in order to replicate the minority class instances from the original dataset until the number of instances for both classes was equalized, summing a total of 65 millions instances. Finally, for DITFS algorithm, the dataset has been discretized using the Minimum Description Length Principle (MDLP) discretizer [19].

The original dataset has been sampled randomly using five differents rates in order to measure the scalability performance of both frameworks: 10%, 30%, 50%, 75% and 100% of the pre-processed dataset is used. Due to a current Flink limitation, we have employed a subset of 150 features of each ECBDL14 dataset sample for the SVM learning algorithm.

Table 1 gives a brief summary of these datasets. For each one, the number of examples (Instances), the total number of features (Feats.), the total number of values (Total), and the number of classes (CL) are shown.

Table 1: SUMMARY DESCRIPTION FOR ECBDL14 DATASET

Dataset	Instances	Feats.	Total	CL
ECBDL14-10	6 500 391	631	4 101 746 721	2
ECBDL14-30	19 501 174	631	12 305 240 794	2
ECBDL14-50	32 501 957	631	20 508 734 867	2
ECBDL14-75	48 752 935	631	30 763 101 985	2
ECBDL14-100	65 003 913	631	41 017 469 103	2

We have established 100 iterations, a step size of 0.01 and a regularization parameter of 0.01 for the SVM. For the LR, 100 iterations and a step size of 0.00001 are used. Finally, for DITFS 10 features are selected using minimum Redundancy Maximum Relevance algorithm [20].

As an evaluation criteria, we have employed the overall learning runtime (in seconds) for SVM and Linear Regression, as well as the overall runtime for DITFS.

For all experiments we have used a cluster composed of 9 computing nodes and one master node. The computing nodes hold the following characteristics: 2 processors x Intel Xeon CPU E5-2630 v3, 8 cores per processor, 2.40 GHz, 20 MB cache, 2 x 2TB HDD, 128 GB RAM. Regarding the software, we have used the following configuration: Hadoop 2.6.0-cdh5.5.1 from Cloudera's open-source Apache Hadoop distribution, Apache Spark and MLlib 1.6.0, 279 cores (31 cores/node), 900 GB RAM (100 GB/node) and Apache Flink 1.0.3, 270 TaskManagers (30 TaskManagers/core), 100 GB RAM/node.

Table 2 shows the learning runtime values obtained by SVM with 100 iterations, using the reduced version of the datasets with 150 features. Currently SVM is not present in the Spark ML library, so we omit that experiment. As we can see, Spark scales much better than Flink. The time difference between Spark and Flink increases with the size of the dataset, being 2.5x slower at the beginning, and 4.5x with the complete dataset.

Table 2: SVM Learning Time in Seconds

Dataset	Spark MLlib	Flink	Difference
ECBDL14-10	42	111	69
ECBDL14-30	61	196	135
ECBDL14-50	103	302	199
ECBDL14-75	123	456	333
ECBDL14-100	174	783	609

Table 3 compares the learning runtime values obtained by LR with 100 iterations. The time difference between Spark MLlib and Spark ML can be explained by internally transforming the dataset from DataFrame to RDD in order to use the same implementation of the algorithm present in MLlib. Spark ML is around 8x times faster than Flink. Spark MLlib version have shown to perform specially better compared to Flink.

Table 3: LR Learning Time in Seconds

Dataset	Spark MLlib	Spark ML	Flink
ECBDL14-10	3	26	181
ECBDL14-30	5	63	815
ECBDL14-50	6	173	1314
ECBDL14-75	8	260	1878
ECBDL14-100	12	415	2566

Table 4 compares the runtime values obtained by DITFS algorithm selecting the top 10 features of the discretized datset. As stated previously, the differences between Spark MLlib and Spark ML can be explained with the internal transformation between DataFrame and RDD. We observe that Flink is around 10x times slower than Spark for 10%, 30% and 50% of the dataset, 8x times slower for 75%, and 4x times slower for the complete dataset.

In Figure 1 we can see the scalability of the three algorithms compared side to side.

Table 4: DITFS Runtime in Seconds

Dataset	Spark MLlib	Spark ML	Flink
ECBDL14-10	44	55	487
ECBDL14-30	111	143	1891
ECBDL14-50	317	441	3240
ECBDL14-75	590	783	4928
ECBDL14-100	1696	2159	6615

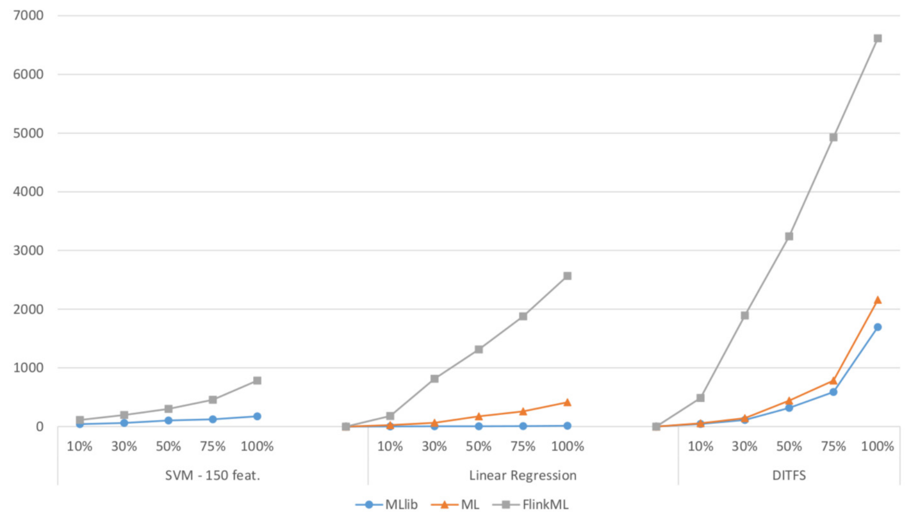


Figure 1: Scalability of SVM, LR and DITFS algorithm in seconds

Conclusions

In this paper, we have performed a comparative study for batch data processing of the scalability of two popular frameworks for processing and storing Big Data, Apache Spark and Apache Flink. We have tested these two frameworks using SVM and LR as learning algorithms, present in their respective ML libraries. We have also implemented and tested a feature selection algorithm in both platforms. Apache Spark have shown to be the framework with better scalability and overall faster runtimes. Although the differences between Spark's MLlib and Spark ML are minimal, MLlib performs slightly better than Spark ML. These differences can be explained with the internal transformations from DataFrame to RDD in order to use the same implementations of the algorithms present in MLlib.

Flink is a novel framework while Spark is becoming the reference tool in the Big Data environment. Spark has had several improvements in performance over the different releases, while Flink has just hit its first stable version. Although some of the Apache Spark improvements are already present by design in Apache Flink, Spark is much refined than Flink as we can see in the results.

Apache Flink has a great potential and a long way still to go. With the necessary improvements, it can become a reference tool for distributed data streaming analytics. It is pending a study on data streaming, the theoretical strengh of Apache Flink.

References

[1] IDC. The Digital Universe of Opportunities, 2014. `http://www.emc.com/infographics/digital-universe-2014.htm`, [Online; accessed 14-July-2016].

[2] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[3] Apache Hadoop Project. Apache Hadoop, 2016. `http://hadoop.apache.org`, [Online; accessed 14-July-2016].

[4] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[5] M. Hamstra, H. Karau, M. Zaharia, A. Konwinski, and P. Wendell. *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, 2015.

[6] Apache Spark. Apache Spark: Lightning-fast cluster computing, 2016. `http://spark.apache.org`, [Online; accessed 14-July-2016].

[7] Apache Flink. Apache Flink, 2016. `http://flink.apache.org`, [Online; accessed 14-July-2016].

[8] Apache Mahout Project. Apache Mahout, 2016. `http://mahout.apache.org`, [Online; accessed 14-July-2016].

[9] MLlib. Machine Learning Library (MLlib) for Spark, 2016. `http://spark.apache.org/docs/latest/mllib-guide.html`, [Online; accessed 14-July-2016].

[10] Jimmy 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):28–37, 2012.

[11] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 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, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[12] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. Mllib: Machine learning in apache spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016.

[13] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.

[14] Apache Spark Project. Project Tungsten (Apache Spark), 2016. `https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html`, [Online; accessed 14-July-2016].

[15] Apache Flink Project. Peeking into Apache Flink's Engine Room, 2016. `https://flink.apache.org/news/2015/03/13/peeking-into-Apache-Flinks-Engine-Room.html`, [Online; accessed 14-July-2016].

[16] Gavin Brown, Adam Pocock, Ming-Jie Zhao, and Mikel Luján. Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. *J. Mach. Learn. Res.*, 13:27–66, January 2012.

[17] Martin Jaggi, Virginia Smith, Martin Takác, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I. Jordan. Communication-efficient distributed dual coordinate ascent. *CoRR*, abs/1409.1458, 2014.

[18] Sara del Río, Victoria López, José Manuel Benítez, and Francisco Herrera. On the use of mapreduce for imbalanced big data using random forest. *Information Sciences*, 285:112 – 137, 2014.

[19] S. Ramírez-Gallego, S. García, H. Mouriño-Talín, and D. Martínez-Rego. Distributed entropy minimization discretizer for big data analysis under apache spark. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, volume 2, pages 33–40, Aug 2015.

[20] Chris Ding and Hanchuan Peng. Minimum redundancy feature selection from microarray gene expression data. *Journal of bioinformatics and computational biology*, 3(02):185–205, 2005.

2 Principal Components Analysis Random Discretization Ensemble for Big Data

- D. García-Gil, S. Ramírez-Gallego, S. García, F. Herrera. Knowledge-Based Systems 150, 166-174 (2018).
  - Status: **Published**.
  - Impact Factor (JCR 2018): **5.101**
  - Subject Category: **Computer Science, Artificial Intelligence**
  - Rank: **17/134**
  - Quartile: **Q1**

Principal Components Analysis Random Discretization Ensemble for Big Data

Diego García-Gil[*]
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
djgarcia@decsai.ugr.es

Sergio Ramírez-Gallego
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
sramirez@decsai.ugr.es

Salvador García
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
salvagl@decsai.ugr.es

Francisco Herrera
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
herrera@decsai.ugr.es

Abstract

Humongous amounts of data have created a lot of challenges in terms of data computation and analysis. Classic data mining techniques are not prepared for the new space and time requirements. Discretization and dimensionality reduction are two of the data reduction tasks in knowledge discovery. Random Projection Random Discretization is a novel and recently proposed ensemble method by Ahmad and Brown in 2014 that performs discretization and dimensionality reduction to create more informative data. Despite the good efficiency of random projections in dimensionality reduction, more robust methods like Principal Components Analysis (PCA) can improve the performance.

We propose a new ensemble method to overcome this drawback using the Apache Spark platform and PCA for dimension reduction, named Principal Components Analysis Random Discretization Ensemble. Experimental results on five large-scale datasets show that our solution outperforms both the original algorithm and Random Forest in terms of prediction performance. Results also show that high dimensionality data can affect the runtime of the algorithm.

Keywords Big Data · Discretization · Spark · Decision Tree · PCA · Data Reduction

1 Introduction

Nowadays everything is constantly creating and storing data. This massive data accumulation can be found in a broad spectrum of real-world areas [1]. In 2014 IDC predicted that by 2020, the digital universe will

[*]Corresponding author.

be 10 times as big as it was in 2013, totaling an astonishing 44 zettabytes [2]. Big Data is not only a huge amount of data, but a new paradigm and set of technologies that can store and process this data. Many of the classic knowledge extraction techniques are not capable of working with these amounts of data, mostly because they were not conceived to work in a Big Data environment.

This scenario becomes particularly important using data reduction techniques. Data reduction techniques are frequently applied to reduce the size of the original data and to clean some errors that it may contain [3] [4]. Two important data reduction techniques are discretization and dimensionality reduction. Discretization is the process of splitting a continuous variable into different categories depending on which interval it falls into [5]. Dimensionality reduction is the mapping of data to a lower dimensional space [6]. One of the most popular methods for dimensionality reduction is Principal Components Analysis (PCA) [7]. PCA is a linear transformation from a high dimensional data space to a principal component feature space. It has been widely used as a dimension reduction technique in many applications.

Ensembles are methods that combine a set of base classifiers to make predictions [8] [9]. These methods have been attracting increasing attention over the last few years due to their ability to correct errors across many diverse base classifiers. Classifier ensembles have shown to be very effective in a broad spectrum of real-world applications. These classifiers have been proven to be *accurate* and *diverse* [10] [11]. Ensembles of decision trees like Random Forest are well known for creating diverse decision trees [12]. This diversity is usually introduced via randomization. Through small changes in input data, diverse decision trees are created and better ensembles are obtained.

This principle is followed by Ahmad and Brown in [13]. Random Projection Random Discretization (RPRD) is an ensemble method that applies two data reduction techniques to the input data and joins the results to create a more informative dataset. First it performs their proposed Random Discretization (RD), which selects $s - 1$ random instances to create s categories and discretize the data using the selected values as limits. Their next step is to perform Random Projection (RP) [14] in the original data to select d features that are the linear combinations of the original m features ($d < m$). Finally the algorithm joins the results of RD and RP to create a new $m + d$ dataset and trains a decision tree with it.

The RPRD Ensemble algorithm has shown to be competitive and to outperform other popular ensemble methods. However, despite its good performance, it still has three main drawbacks: 1) As the projected dimension is decreased, as it drops below $\log k$, random projection suffers a gradual degradation in performance [15]. 2) RP is highly unstable - different random projections may lead to radically different results [16]. There are more informative dimensionality reduction methods like PCA. 3) RPRD is not prepared for working with Big Data. This therefore limits the potential use of the ensemble. For example, in cases with thousands of features or millions of instances, RPRD cannot be used.

In order to fill this gap and inspired by the RPRD ensemble algorithm, we propose a new ensemble method under Apache Spark using PCA, called Principal Components Analysis Random Discretization Ensemble (PCARDE) for Big Data. In our design we use PCA instead of RP for improving the dimensionality reduction step. The choice of PCA is motivated by the fact that it is not an iterative method and can be parallelized. The usage of Big Data frameworks like Apache Spark enables the use of PCA over datasets with thousands of features and millions of instances. It also allows to perform huge matrix multiplications for methods like RP. In Big Data problems, the computational cost difference between PCA and RP becomes unclear, since the multiplication of large matrices in RP is a computationally demanding operation.

We have also designed an algorithm named $\mathcal{X}^2$ Random Discretization Ensemble ($\mathcal{X}^2$ RD), which uses $\mathcal{X}^2$ for performing feature selection, in order to show the importance and impact of the addition of PCA to our proposed ensemble method. The choice of $\mathcal{X}^2$ comes motivated by the fact that $\mathcal{X}^2$ is a more informed method than RP and this can lead to a performance improvement.

Apache Spark is a fast and general engine designed for large-scale data processing based on in-memory computation [17], [18]. Apache Spark has its own Machine Learning library named MLlib [19]. Among our objectives is designing an ensemble algorithm for Big Data and to integrate the algorithm into the MLlib Library as a third-party package. Spark's implementation of the algorithm can be downloaded from the Spark's community repository[2].

To show the effectiveness of our approach, we have carried out an experimental evaluation with five large datasets, namely *poker, SUSY, HIGGS, epsilon and ECBDL14*. These datasets have very different properties and allow us to test all aspects of our implementation. Finally, we show a comparative study of the performance of PCARDE, $\mathcal{X}^2$ RD, RPRD and Random Forest [20] [21].

The remainder of this paper is organized as follows: Section II outlines the main concepts of the RPRD Ensemble. Section III explains the new ensemble design based on PCA. Section IV describes the experiments carried out to check the effectiveness of this proposal. Finally, Section V concludes the paper.

2 Background

In this section we first introduce the RPRD Ensemble algorithm used as reference in our ensemble interpretation and its two components, RD and RP. Then we introduce PCA and MapReduce Model.

2.1 Random Discretization

Discretization is the process of partitioning a set of continuous attributes into discrete attributes by associating categorical values to the intervals [22].

To create s categories we need s - 1 different intervals. There are different methods to create these intervals like entropy minimization [23], implemented in Apache Spark. The main problem with these methods is that they create the same discretized dataset after different executions. In an ensemble some randomization is necessary in order to introduce diversity to the decision trees.

In RD randomization is introduced to the discretization process. In algorithm 1 we describe the mechanism of RD from lines 5 to 12. First $s-1$ data points are randomly selected from the training data to create s categories. Then for each feature, every $s-1$ data points are sorted. Finally the dataset is discretized into s categories using these $s-1$ sorted data points. These thresholds are selected randomly each iteration of the ensemble. It is possible that in some features, all selected data points will have the same value. In this case, $s-1$ are selected randomly between the minimum and the maximum values of the feature.

2.2 Random Projection

The objective of dimensionality reduction techniques is to produce a compact low-dimensional encoding of a given high dimensional dataset. Random projection (RP) has emerged as a novel method for the dimensionality reduction problem.

In RP, the original m-dimensional data is projected to a d-dimensional ($d << m$) subspace through the origin, using a random $d \times m$ matrix $\boldsymbol{R}$ whose columns have unit lengths, and whose elements $r_{i,j}$ are often Gaussian distributed. Using matrix notation where $\boldsymbol{X}_{m\times N}$ is the original set of N m-dimensional observations,

$$\boldsymbol{X}^{RP}_{d\times N} = \boldsymbol{R}_{d\times m}\boldsymbol{X}_{m\times N}$$

[2]`http://spark-packages.org/package/djgg/PCARD`

Algorithm 1 RPRD Ensemble

```
1: **Input:** Dataset $T$ with $m$ continuous features and $k$ classes $c_1, c_2, ..., c_k$. $L$ the size of the ensemble.
2: **Learning Phase**
3: **for** $i = 1...L$ **do**
4:     **Random Discretization**
5:     For $s$ categories in each dimension, select $s-1$ data points randomly from the training data.
6:     **for** $j = 1...m$ **do**
7:         Get the $j^{th}$ feature values of $s-1$ points and sort them.
8:         If all points have the same value, select $s-1$ points randomly until there are at least two different
           values.
9:     **end for**
10:    **for** $i = 1...N$ **do**
11:        Discretize the $i^{th}$ data point using the values got in the previous step to create $m$ discretized features
           $S_i$.
12:    **end for**
13:    **Random Projection**
14:    Use Random Projection $RP_i$ to create $d$ features $R_i$.
15:    Combine $S_i$ and $R_i$ to create $m + d$ dimensional dataset $T_i$.
16:    **Learning Model**
17:    Treat dataset $T_i$ as continuous and learn $D_i$ decision tree on it.
18: **end for**
19: **Prediction Phase**
20: **for** $i = 1...L$ **do**
21:    Convert $x$ into $m + d$ dimensional data point $x_i$ using Random Discretization $(RD_i)$ and Random
       Projection $(RP_i)$.
22:    Let $p_{i,j}(x)$ be the probability for $x_i$ by the decision tree $D_i$ to the hypothesis that $x$ comes from
       class $c_j$. Calculate $p_{i,j}(x)$ for all classes $j = 1..k$.
23: **end for**
       Calculate the confidence $C(j)$ for each class $c_j (j = 1..k)$ by the average contribution method,
       $C(j) = \frac{1}{L} \sum_{i=1}^{L} p_{i,j}(x)$
       The class with the largest confidence will be the class of $x$.
```

is the projection of the data onto a lower *d*-dimensional subspace. The key idea of random mapping arises from the Johnson-Lindenstrauss lemma [14]: if points in a vector space are projected onto a randomly selected subspace of suitably high dimension, then the distances between the points are approximately preserved.

2.3 Random Projection Random Discretization Ensembles and classification

RD and RP perform data reduction, but have different mechanisms; RD performs random discretization (lines 5 to 12) in input space whereas RP creates new features (line 14) that are the linear combinations of the original features. The RPRD ensemble was based on the idea that both RP and RD can be combined to create a better ensemble method (RPRDE). In each iteration RD and RP are performed on the input data, and then the results are fused. Finally a decision tree is trained using this new data. This results in better trees compared to the original data as they have more features to select at each node.

In the prediction phase a data point is converted into a $m+d$ dimensional data point using the corresponding values of RD and RP for the iteration. Then the probabilities of each class are calculated by the decision tree. This process is repeated for each iteration. Finally the confidence value for each class is calculated. The class with the highest confidence value will be the class of the data point.

2.4 Principal Components Analysis

The use of PCA allows the number of variables in a multivariate dataset to be reduced, whilst retaining as much of the variation present in the dataset as possible. This reduction is achieved by taking *p* variables $T_1, T_2...T_p$ and finding the combinations of these to produce principal components (*PC*) $PC_1, PC_2...PC_p$, which are uncorrelated. These *PC* are also termed eigenvectors. The lack of correlation is a useful property as it means that the *PC* are measuring different ”dimensions” in the data. Nevertheless, *PC* are ordered so that PC_1 exhibits the greatest amount of the variation, PC_2 exhibits the second greatest amount of the variation, and so on. That is $var(PC_1) \geq var(PC_2) \geq var(PC_3) \geq ... \geq var(PC_p)$, where $var(PC_i)$ expresses the variance of PC_i in the dataset being considered. $var(PC_i)$ is also called the eigenvalue of PC_i.

PCA always produces the same results for a fixed number of principal components. We need some diversity in order to obtain diverse decision trees. We achieve diversity by introducing randomization in the number of principal components we select. For each decision tree, we select the number of principal components randomly in the interval [*1*, *m-1*] (*m* number of features).

2.5 MapReduce Model

MapReduce is a framework designed by Google in 2003 [24]. It has become one of the most relevant tools for processing and generating large datasets with parallel and distributed algorithms on a cluster. The MapReduce framework manages all data transfers and communications between the systems. It also provides redundancy, fault-tolerance and job scheduling. The end users only have to launch their tasks and the framework will do the rest [25].

MapReduce model is composed of a Map procedure that performs a transformation, and a Reduce method that performs a summary operation. The workflow of a MapReduce program is as follows: first the master node splits the dataset and distributes the results across the cluster. Then each node applies the Map function to the local data. After that process is finished the data is redistributed based on the key-value pairs generated in the Map phase. Once the data has been redistributed so that all pairs belonging to one key are in the same node, it is processed in parallel.

The Map and Reduce functions are defined to work with data structured in <key, value> pairs. Map function takes one pair of data as input and returns a list of intermediate pairs:

$$Map(< key1, value1 >) \to list(< key2, value2 >)$$

The Map function is then applied to each pair in the input data in parallel. The result is grouped by key and then distributed across the cluster. The Reduce phase is applied to the previous list of intermediate pairs to produce a single output value:

$$Reduce(< key2, list(value2) >) \to < key2, value3 >$$

Apache Hadoop [26] [27] is the most popular open-source framework for large-scale data storing and processing based on the MapReduce model. The framework is designed to handle hardware errors automatically. In spite of its popularity and performance, Hadoop presents some important limitations [28]:

- Poor performance on online and iterative computing.
- Low inter-communication capacity.
- Insufficiency for in-memory computation.

Apache Spark is an open-source framework built around speed, ease of use, and sophisticated analytics [17] [18] [29]. It has an advanced Directed Acyclic Graph (DAG) execution engine that supports cyclic data flow and has in-memory computing, the most important feature of Spark. This enables applications to run up to 100 times faster compared to Hadoop (in certain cases). It is designed to store and process as much in-memory data as possible. It allows users to persist the data both in-memory or to disk.

Spark's core concepts are Resilient Distributed Datasets (RDDs) [30]. RDDs are a distributed memory abstraction, they can be described as a collection of data partitioned across the clusters. RDDs are immutable, applying a transformation produces a new RDD while the original RDD remains the same. RDDs support two types of operations: transformations, which are not evaluated when defined and will produce a new RDD. And actions, which evaluate and return a new value. When an action is called on a RDD, all the previous transformations are applied in parallel to each partition of the RDD. RDDs are also fault-tolerant, if a partition is lost, it can be recalculated with the associated DAG. Spark provides a list of transformations and actions that allow more complex programs than those implemented in Hadoop to be created.

3 Principal Components Analysis Random Discretization Ensemble

In this section, we present the design of the ensemble by using a more powerful method like PCA, proving its performance over big real-world problems.

For the implementation of the algorithm, we have used some basic Spark primitives. Here, we outline those more relevant for the algorithm:

- map: Applies a transformation to each element of a RDD and returns a new RDD representing the results.
- zip: Joins one RDD with another one.
- $zipWithIndex$: Zips a RDD with its element indices.
- $lookup$: Returns the list of values in the RDD for a given key.

PCARDE has two phases, learning and prediction. In the learning phase we train a model from the input data and in the prediction phase, we apply this model to the test data in order to obtain a prediction. In the learning phase we discretize the training data using RD. As RDDs are unsorted by nature, to select a specific instance for the RD method it performs the $zipWithIndex$ operation to the RDD in order to add an index to each instance. With the added index we can get the values of the features using the $lookup$ operation. For iterating through every instance and to discretize them, Spark's map function is used. Once RD has been performed, PCA is also applied to the training data with a random value of principal components in the interval [*1*, *m-1*] (m number of features). Finally we join the results with the zip function and train a decision tree using this new dataset. We repeat this process L times, L the size of the ensemble.

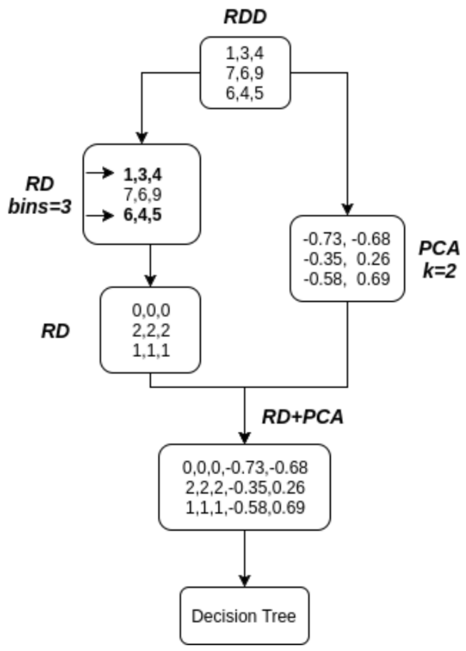


Figure 1: PCARDE learning phase flowchart

In Figure 1 we can see a flowchart of the PCARDE learning phase process.

In the prediction phase we discretize the data point with the cut points obtained in the learning phase and perform PCA with the model obtained previously. Then we predict the probability of the data point belonging to each class. We repeat this process L times. Finally we add all probabilities. The class with the largest probability will be the class of the data point.

Our algorithm is divided into two procedures explained in two sections as follows: Section 3.1 describes the learning phase. And Section 3.2 provides details of the prediction phase.

3.1 Learning models phase

Procedure 2 explains the learning phase in the ensemble. The algorithm discretizes using RD method and performs PCA, both with the training data, and then joins the result of both methods to create a new dataset. Then a decision tree is trained with this new data. It requires the following as input parameters: the dataset, the size of the ensemble and the number of categories for the discretization.

The first step is to perform RD. First we calculate the thresholds for the discretization. With these thresholds the data is discretized through a Map function. It iterates through every feature and assigns a discrete value depending on the feature's value and the thresholds selected. The second step is to perform PCA. First we select a random number d in the interval [*1*, *m-1*] (m number of features). Then we project the data to a lower dimensional space using PCA, keeping only the first d principal components. The final step is to fuse the results from RD and PCA, and to train a decision tree with it. We repeat this process L times, saving the cut points, the PCA models and the trees created at each iteration. Once all the trees have been trained, the model is created and returned.

Procedure 3 describes the process of selecting the thresholds for the discretization. First we select $s - 1$ random numbers in the interval [0, n] (n the number of instances in the data). Then we add an index to each instance in the dataset in order to get the values of the features we have selected in the previous step.

Algorithm 2 Main PCARDE algorithm

```
 1: Input: data an RDD of type LabeledPoint (features, label).
 2: Input: $L$ the size of the ensemble.
 3: Input: $s$ the number of categories for the discretization.
 4: Output: The model created, an object of class PCARDModel
 5: for $i = 0...L$ do
 6:     Random Discretization
 7:     $cutPoints(i) \leftarrow get\_cut\_points(data, s)$
 8:     $rdData \leftarrow$
 9:     map $l \in data$
10:         for $c = 0...size(l) - 1$ do
11:             $l \leftarrow discretize(l(c), cutPoints(i)(c))$
12:         end for
13:     end map
14:     PCA
15:     $d \leftarrow random(1, size(data) - 1)$
16:     $pcaModels(i) \leftarrow PCA(data, d)$
17:     $pcaData \leftarrow transform(data, pcaModels(i))$
18:     $pcardData \leftarrow zip(rdData, pcaData)$
19:     $trees(i) \leftarrow decisionTree(pcardData)$
20: end for
21: $return(PCARDModel(L,$
22:             $cutPoints, pcaModels, trees))$
```

The next step is to get those values, sort them and check if they are all equal. This process is described as follows: first we transpose the thresholds array in order to access to the cut points of each feature through a Map function, and add an index to each threshold. Then we iterate through the thresholds using a Map function. The cut points are sorted and checked if they are different. If they are all equal, we get all the different values for that feature, then take $s - 1$ points randomly and finally sort them. The result is a list of the thresholds for each feature, which is returned to the main algorithm as $cutPoints$.

3.2 Prediction phase

Procedure 4 explains the prediction phase in the ensemble. The algorithm discretizes using RD and performs PCA on the test data point using the cut points and the models for PCA provided the same way as was described in the main procedure. Then the results of the two methods are joined and then predicted with the corresponding tree. The tree gives the probabilities for each class. These probabilities are added to a list of predictions at each iteration. The class with the greatest probability is selected. It selects the index of the maximum probability for an instance as a decision.

Algorithm 3 Function to select the cut points for a given dataset (get_cut_points)

```
1: Input: data an RDD of type LabeledPoint (features, label).
2: Input: s the number of categories for the discretization.
3: Output: An array with thresholds for each feature
4: $instances \leftarrow get\_random\_array(s-1, size(data))$
5: $indexData \leftarrow zipWithIndex(data)$
6: for $i = 0...s-1$ do
7:     $values \leftarrow lookup(indexData, instances(i))$
8: end for
9: $thresholds \leftarrow zipWithIndex(transpose(values))$
10: $cutPoints \leftarrow$
11: map $(l, i) \in thresholds$
12:     $feature \leftarrow sorted(distinct(l))$
13:     if $size(feature) = 1$ then
14:         $col \leftarrow distinct(get\_feature\_values(data, i))$
15:         $sorted(take\_random(s-1, col))$
16:     else
17:         $feature$
18:     end if
19: end map
20: $return(cutPoints)$
```

Algorithm 4 PCARDModel algorithm

```
1: Input: L the size of the ensemble.
2: Input: cutPoints the thresholds for the discretization.
3: Input: pcaModels the models for performing PCA.
4: Input: trees the models of the trained trees.
5: Output: The class of the data point.
6: function TEST($test : LabeledPoint$)
7:     $rawPredictions \leftarrow 0$
8:     for $i = 0...L$ do
9:         $rd \leftarrow 0$
10:        for $c = 0...size(test) - 1$ do
11:            $rd(c) \leftarrow discretize(test(c), cutPoints(i)(c))$
12:        end for
13:        $pcaTest \leftarrow transform(test, pcaModels(i))$
14:        $pcardData \leftarrow zip(rd, pcaTest)$
15:        $rawPredictions \leftarrow rawPredictions + predict(trees(i), pcardData)$
16:    end for
17:    $label \leftarrow max\_index(rawPredictions)$
18:    $return(label)$
19: end function
```

4 Experimental Results

This section describes the experiments carried out to show the performance of PCARDE for Big Data algorithm in five huge problems. We carried out the comparative study of PCARDE method facing the original proposal, $\mathcal{X}^2$ RD, and MLlib's implementation of Random Forest to test the effectiveness of PCARDE approach. We also give a brief description of $\mathcal{X}^2$ RD ensemble. Additionally, we compare the results of PCARDE, $\mathcal{X}^2$ RD and RPRD using Naïve Bayes [31] as a classifier. Computing times are also analyzed, including learning and prediction runtimes. Finally, we have performed an analysis of the impact of PCA on our proposed ensemble method. RPRD and $\mathcal{X}^2$ RD ensemble algorithms have been also implemented in Apache Spark.

4.1 Experimental Framework

Five huge classification datasets are used in our experiments:

- Poker hand dataset, which has 1,025,000 instances with 11 attributes. In this dataset each record is an example of a hand consisting of five playing cards drawn from a standard deck of 52. Each card is described using two attributes (suit and rank), for a total of 10 predictive attributes. There is one Class attribute that describes the "Poker Hand".
- SUSY dataset, which consists of 5,000,000 instances and 18 attributes. This dataset contains simulated collisions events at the Large Hadron Collider. The task is to distinguish between a signal process which produces supersymmetric (SUSY) particles and a background process which does not [32].
- HIGGS dataset, which has 11,000,000 instances and 28 attributes. This dataset is a classification problem to distinguish between a signal process which produces Higgs bosons and a background process which does not.
- Epsilon dataset, which consists of 500,000 instances with 2,000 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 [33].
- ECBDL14 dataset. This dataset 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. It 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: 2% of positive instances. For this problem, the Random OverSampling (ROS) algorithm used in [34] was applied in order to replicate the minority class instances from the original dataset until the number of instances for both classes was equalized, summing a total of 65 million instances.

We carried out experiments on three different sizes of ensembles. The sizes were selected such that one may represent small ensembles, other medium ensembles, and the last one represents large ensembles. There is no clear definition of small, medium and large ensembles. Following [35], we established the size of small ensembles as 10 iterations and large ensembles as 100 iterations.

The experiments were conducted following 5 fold cross-validation.

Table 1 gives a brief summary of these datasets. For each one, the number of examples (Instances), the total number of attributes (Atts.), the total number of training data (Total), the number classes (CL), and the size in memory (expressed in GB) are shown.

Table 1: Summary Description for Classification Datasets

Dataset	Instances	Atts.	Total	CL	Size (GB)
poker	1,025,010	11	11,275,110	10	0.023
SUSY	5,000,000	18	90,000,000	2	2.23
HIGGS	11,000,000	28	308,000,000	2	7.39
epsilon	400,000	2,000	800,000,000	2	14.16
ECBDL14	65,003,913	631	39,847,398,669	2	123.76

We have established 5 intervals for RD, the same for RPRD and $\mathcal{X}^2$ RD methods. For RPRD method, recommended values are used (5 bins for RD, the number of new features created by using RP d as $2(\log_2 c)$ where c is the number of features, and the elements r_{ij} of the Random Matrix $\boldsymbol{R}$ are Gaussian distributed.)

For Random Forest, default values are used (featureSubsetStrategy = "auto", impurity = "gini", maxDepth = 5 and maxBins = 32).

Datasets have been normalized to [0,1] interval to avoid negative values.

As evaluation criteria, we use two well-known evaluation metrics. First of all, prediction accuracy is used to evaluate the accuracy produced by the predictors (number of examples correctly labeled as belonging to a given class divided by the total number of elements). ECBDL14 dataset is highly unbalanced, we have used the True Positive Rate (TPR) and True Negative Rate (TNR) TPR·TNR metric, as used in the competition [36].

For all experiments we have used a cluster composed of 14 computing nodes and one master node. The computing nodes hold the following characteristics: 2 processors x Intel Core i7-4930K, 6 cores per processor, 3.40 GHz, 12 MB cache, 4 TB HDD, 64 GB RAM. Regarding software, we have used the following configuration: Hadoop 2.6.0-cdh5.4.3 from Cloudera's opensource Apache Hadoop distribution, Apache Spark and MLlib 1.5.1, 252 cores (18 cores/node), 728 RAM GB (52 GB/node).

4.2 $\mathcal{X}^2$ Random Discretization Ensemble

Our first approach to improve the dimensionality reduction step has been to replace the RP method with a more informed one like $\mathcal{X}^2$ [37]. We use $\mathcal{X}^2$ for performing feature selection. It performs a $\mathcal{X}^2$ test of independence between each feature and the class label, then it selects the top features based on the dependency between the feature and the class label.

Like PCA, $\mathcal{X}^2$ also produces the same results for a fixed number of features. As stated previously the solution is to introduce randomization in the number of features we compute in order to obtain diverse decision trees. For each decision tree, we select the number of features randomly in the interval [*1*, *m-1*] (*m* number of features).

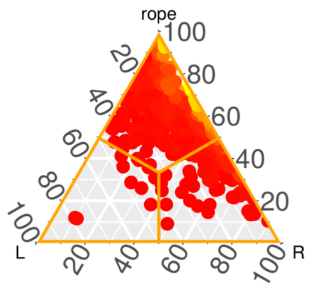
4.3 Analysis of Accuracy Performance

In this section, we present the analysis of the performance results obtained by PCARDE, facing RPRD and Random Forest. To prove that the combination of RD and PCA produces a better ensemble method, we also show the prediction accuracy for both RD and PCA independently.

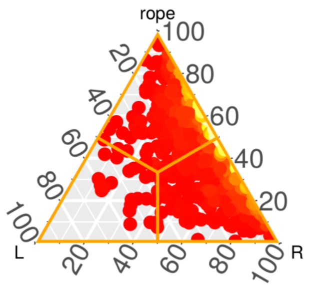
Table 2 shows the prediction accuracy values for the five datasets and the three sizes of ensemble using our new design and a Decision Tree as a classifier. According to these results, it is demonstrated that the

Table 2: RD vs PCA vs PCARDE Test Accuracy using a Decision Tree

Dataset	Trees	RD	PCA	PCARDE
Poker	10	54.73(±0.43)	54.68(±0.24)	55.07(±0.19)
	50	54.76(±0.49)	54.81(±0.13)	54.92(±0.20)
	100	54.73(±0.28)	54.76(±0.28)	54.97(±0.23)
SUSY	10	78.00(±0.09)	75.30(±0.20)	78.31(±0.07)
	50	78.26(±0.04)	74.97(±0.08)	78.47(±0.09)
	100	78.31(±0.07)	75.31(±0.34)	78.49(±0.03)
HIGGS	10	68.64(±0.25)	60.10(±2.01)	68.75(±0.56)
	50	68.98(±0.15)	60.44(±0.76)	69.28(±0.18)
	100	69.17(±0.12)	60.81(±0.25)	69.35(±0.10)
epsilon	10	68.78(±0.39)	78.14(±0.09)	78.57(±0.37)
	50	69.04(±0.19)	78.14(±0.09)	78.57(±0.25)
	100	69.22(±0.25)	78.14(±0.09)	78.58(±0.27)
ECBDL14 [3]	10	0.1884	0.2400	0.4742
	50	0.1885	0.2410	0.4717
	100	0.1880	0.2415	0.4742



(a) RD



(b) PCA

Figure 2: Bayesian Sign Test heatmap for RD and PCA against PCARDE with 10 trees

combination of RD and PCA produces a better ensemble method. We can assert that ensembles of 10 trees are the best choice as the improvement in prediction with 5 and 10 times more trees is minimal. The ensemble also proves to be very stable, as there is little or no improvement in bigger ensemble sizes.

For a deeper analysis of the results, we have performed a Bayesian Test in order to demonstrate the validity of PCARDE. The key idea with Bayesian tests is to obtain a distribution of the difference between two algorithms, and to make a decision when 95% of the distribution is in one of the three regions: left, rope (region of practical equivalence), and right [38]. Bayesian Sign Test is a Bayesian version of non-parametric sign test that uses the Dirichlet Process [39]. It is applied to the mean accuracy of each dataset. We obtain a sample of the distribution on the probabilities, which means that each point in the sample is a triplet with the probability of the difference between two algorithms belonging to the left, right or rope regions.

[3]For this dataset TPR·TNR metric is being used.

Figure 2 shows a heatmap of the Bayesian Sign Test for RD and PCA against PCARDE, using 10 trees. As we can observe, the probability of the difference being to the left is minimal since most of the points are on the right side of the triangle. This means that Bayesian Sign Test is assigning a probability of 0 to PCARDE performing worse than RD or PCA. For the Bayesian tests and graphics, we have used an R package that contains a set of non-parametric and Bayesian tests, named rNPBST [40].

Table 3 compares the accuracy values in prediction obtained by PCARDE, $\mathcal{X}^2$ RD and RPRD for all datasets using a Decision Tree as a classifier. We show that $\mathcal{X}^2$ RD is an improvement for some datasets over the original proposal, while our algorithm performs better than both of them. This improvement is especially important in the Epsilon dataset, where there is a difference of 10% more accuracy. PCA is selecting the most informative principal components, while RP is projecting informative features and less informative features randomly to a lower dimensional space.

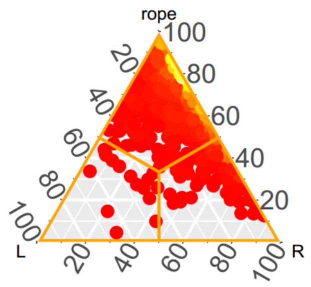
Table 3: PCARDE vs $\mathcal{X}^2$ RD vs RPRD Test Accuracy using a Decision Tree

Dataset	Trees	PCARDE	$\mathcal{X}^2$ RD	RPRD
poker	10	55.07(±0.19)	54.72(±0.13)	53.84(±0.26)
	50	54.92(±0.20)	54.64(±0.26)	53.82(±0.25)
	100	54.97(±0.23)	54.70(±0.21)	53.82(±0.07)
SUSY	10	78.31(±0.07)	77.43(±0.15)	78.19(±0.05)
	50	78.47(±0.09)	77.57(±0.18)	78.28(±0.09)
	100	78.49(±0.03)	77.65(±0.17)	78.35(±0.08)
HIGGS	10	68.75(±0.56)	68.48(±0.14)	68.36(±0.09)
	50	69.28(±0.18)	69.06(±0.06)	69.01(±0.13)
	100	69.35(±0.10)	69.18(±0.06)	69.22(±0.17)
epsilon	10	78.57(±0.37)	64.60(±1.33)	68.64(±0.33)
	50	78.57(±0.25)	66.35(±0.34)	69.10(±0.27)
	100	78.58(±0.27)	66.05(±0.80)	69.31(±0.29)
ECBDL14[3]	10	0.4742	0.4512	0.4735
	50	0.4714	0.4524	0.4775
	100	0.4742	0.4519	0.4757

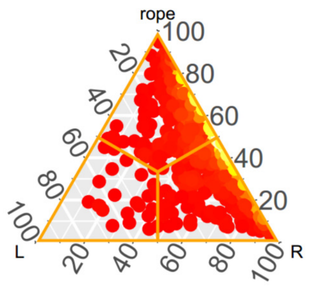
Figure 3 represents the Bayesian Sign Test for $\mathcal{X}^2$ RD and RPRD against PCARDE using a Decision Tree, all three with 10 trees. As we can see, PCARDE is performing much better than $\mathcal{X}^2$ RD and RPRD, since the probability of most of the points are on the right side of the triangle. This is especially accentuated in RPRD. This means that Bayesian Sign Test is choosing PCARDE as a better performing algorithm compared to $\mathcal{X}^2$ RD or RPRD.

In Table 4 we have performed the same experimentation of Table 3 but changing the classifier of the three ensemble methods for MLlib's implementation of Naïve Bayes. Results show that, in spite of the overall drop in test accuracy, PCARDE is still the best performing method. This proves that the combination of RD and PCA produces a better ensemble method regardless of the classifier used.

Figure 4 illustrates the Bayesian Sign Test for $\mathcal{X}^2$ RD and RPRD against PCARDE using Naïve Bayes, all three with 10 trees. As observed previously, PCARDE is still performing better than $\mathcal{X}^2$ RD and RPRD. The probability of most of the points are on the right side of the triangle, especially in $\mathcal{X}^2$ RD. As stated



(a) $\mathcal{X}^2$ RD

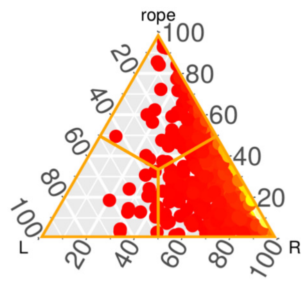


(b) RPRD

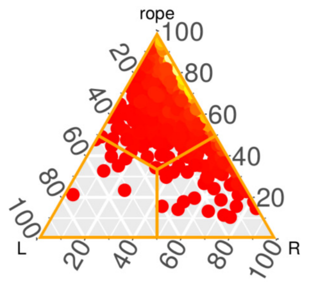
Figure 3: Bayesian Sign Test heatmap for $\mathcal{X}^2$ RD and RPRD against PCARDE with 10 trees

Table 4: PCARDE vs $\mathcal{X}^2$ RD vs RPRD Test Accuracy using Naïve Bayes

Dataset	Trees	PCARDE	$\mathcal{X}^2$ RD	RPRD
Poker	10	50.17($\pm$0.01)	50.12($\pm$0.01)	50.12($\pm$0.01)
	50	50.12($\pm$0.01)	50.12($\pm$0.01)	50.12($\pm$0.01)
	100	50.17($\pm$0.01)	50.12($\pm$0.01)	50.12($\pm$0.01)
SUSY	10	73.12($\pm$0.48)	67.44($\pm$0.56)	72.68($\pm$0.38)
	50	73.41($\pm$0.16)	67.30($\pm$0.53)	73.40($\pm$0.16)
	100	73.38($\pm$0.10)	66.94($\pm$2.06)	73.39($\pm$0.25)
HIGGS	10	58.17($\pm$0.92)	56.96($\pm$0.91)	57.07($\pm$1.16)
	50	58.74($\pm$0.31)	57.49($\pm$0.67)	58.71($\pm$0.47)
	100	58.82($\pm$0.39)	57.09($\pm$0.36)	58.68($\pm$0.35)
epsilon	10	69.77($\pm$1.21)	67.66($\pm$3.04)	69.15($\pm$1.37)
	50	69.81($\pm$0.74)	68.06($\pm$2.68)	69.22($\pm$1.08)
	100	69.84($\pm$0.51)	68.14($\pm$2.01)	69.26($\pm$0.89)
ECBDL14	10	0.2122	0.0076	0.2179
	50	0.2128	0.0079	0.2174
	100	0.2130	0.0077	0.2168



(a) $\mathcal{X}^2$ RD



(b) RPRD

Figure 4: Bayesian Sign Test heatmap for $\mathcal{X}^2$ RD and RPRD against PCARDE with 10 trees

previously, Bayesian Sign Test is choosing PCARDE as a better performing algorithm compared to $\mathcal{X}^2$ RD or RPRD.

Table 5 compares the accuracy values in prediction obtained by PCARDE with 10 trees and Random Forest with 200 and 500 trees. We show that our algorithm also outperforms Random Forest. Even with big ensembles with up to 500 trees, Random Forest can not match or outperform PCARDE with 10 trees.

Table 5: PCARDE vs Random Forest Test Accuracy

Dataset	PCARDE	RF 200	RF 500
poker	55.07($\pm$0.19)	51.56($\pm$0.98)	51.61 ($\pm$0.97)
SUSY	78.31($\pm$0.07)	77.73($\pm$0.04)	77.76($\pm$0.07)
HIGGS	68.75($\pm$0.56)	67.98($\pm$0.12)	67.94($\pm$0.13)
epsilon	78.57($\pm$0.37)	73.24($\pm$0.32)	73.41($\pm$0.22)
ECBDL14[3]	0.4742	0.4642	0.4634

4.4 Computing Times

In the previous section we have shown the suitability of PCARDE in terms of accuracy. In order to be adequate for Big Data environments, the proposed ensemble method has to be scalable as well. This section is devoted to present the computing times for PCARDE, $\mathcal{X}^2$ RD, RPRD and Random Forest.

Table 6 shows learning runtime values obtained by PCARDE, $\mathcal{X}^2$ RD, RPRD (all three with 10 trees) and Random Forest. As we can see, for datasets with a small number of features PCARDE performs as fast as the $\mathcal{X}^2$ RD algorithm. The Epsilon dataset represents a challenge for PCA, as it has a very large number of features to compute. RP performs a matrix multiplication whilst PCA has to compute the principal components of 2,000 features. However the performance improvement obtained by PCARDE over RPRD and Random Forest justifies this result. Something similar happens to ECBDL14 as there is a huge amount of instances to be computed by PCA.

Table 6: Learning Time Values in Seconds

Dataset	PCARDE 10	$\mathcal{X}^2$ RD 10	RPRD 10	RF 500
poker	159	175	169	294
SUSY	193	151	328	351
HIGGS	248	234	604	325
epsilon	2,048	441	338	124
ECBDL14	22,093	17,280	12,607	4,460

In classification problems, learning time is not as important as prediction time. As example, deep convolutional networks can take days, weeks or even months to learn, but they achieve very good performance in prediction [41] [42]. Prediction times are the most important measure in terms of runtimes for classification algorithms.

Table 7 shows prediction runtime values for one test example obtained by PCARDE, $\mathcal{X}^2$ RD, RPRD (all three with 10 trees) and Random Forest. As we can see, PCARDE is more competitive in prediction. $\mathcal{X}^2$ RD and RPRD only perform better than PCARDE in the Epsilon dataset.

4.5 Experimental Study to Measure the Impact of PCA

In this section we show two comparative studies of PCA and $\mathcal{X}^2$ in the Epsilon dataset in order to show the importance and impact of the addition of PCA to our proposed ensemble method. We have selected the Epsilon dataset because of its high dimensionality.

Table 7: Prediction Time Values in Microseconds

Dataset	PCARDE 10	$\mathcal{X}^2$ RD 10	RPRD 10	RF 500
poker	63.41	99.51	78.05	82.93
SUSY	34.00	60.00	41.00	44.00
HIGGS	16.36	30.45	23.18	14.55
epsilon	2,350	412.50	325.00	50.00
ECBDL14	148.97	245.37	214.14	13.10

In the first study we show the most important features selected by PCA and $\mathcal{X}^2$ in an ensemble with 100 trees, while in the second one, we compare the accuracy of a decision tree trained using PCA, $\mathcal{X}^2$ and RP with the same number of features.

For the first study, in order to select the most important features for PCA, we have used the correlation between the principal components and the original features [43]. This correlation is defined as follows:

$$\rho_{Y_i,X_k} = \frac{e_{ki}\sqrt{\lambda_i}}{\sqrt{\sigma_k}}$$

Being p the principal components Y_i, associated with the random vector X with the known co-variance matrix σ and being (λ_i, e_i) its eigenvalues-eigenvectors.

Once we have calculated the correlation between the principal components and the original features, we sort them by importance for each principal component. Then we select the top k most important features and add them to a frequency list, increasing the count if they were already present. We repeat this process for each iteration of the ensemble.

For $\mathcal{X}^2$ we add the selected features for each iteration of the ensemble to a frequency list and then we select the top k most frequent features.

In Figure 5 we can see that there are a few features that PCA selects many times, while the rest are less frequently selected.

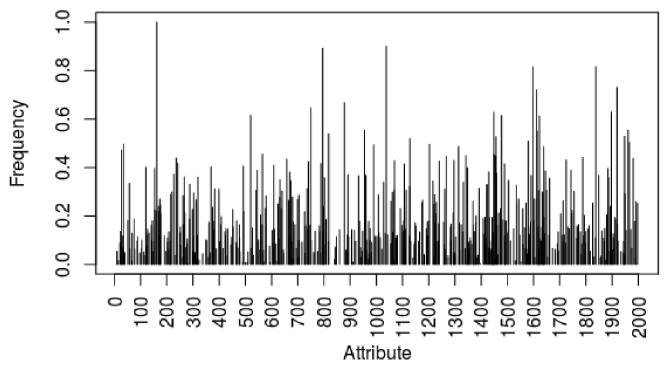


Figure 5: PCA most selected features

In Figure 6 we can see that $\mathcal{X}^2$ is selecting some features many times, while the rest of the features are much less frequently selected.

Both Figures 5 and 6 are a graphic representation of the most important features selected by PCA and $\mathcal{X}^2$. To make the graphics clearer, we have selected only the top 300 most selected features for both methods.

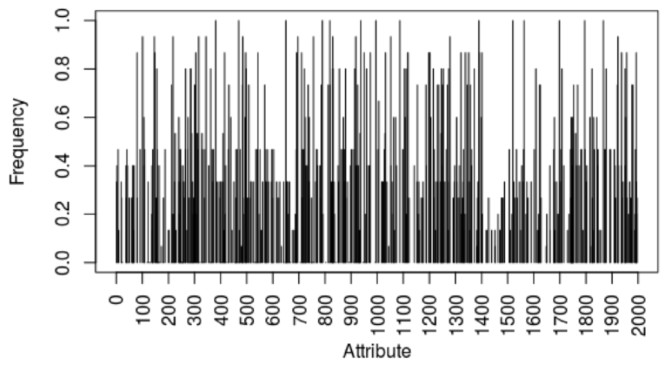


Figure 6: $\mathcal{X}^2$ most selected features

The X-Axis of the figures represents the 2,000 attributes of the Epsilon dataset. The Y-Axis represents the normalized frequency of each feature.

If we compare the two figures, we can see that the two methods are selecting very different features. For example, there is a remarkable difference between the two methods around the feature 1,400 and 1,500, while $\mathcal{X}^2$ almost does not select any feature, PCA is selecting many of them.

In the second study, we have performed PCA, $\mathcal{X}^2$ and RP to the Epsilon dataset with 50, 100 and 500 features. The resulting datasets are trained using Spark's Decision Tree with the default parameters (impurity = "gini", maxDepth = 5 and maxBins = 32).

Table 8 compares the accuracy values in prediction obtained by Spark's Decision Tree trained with 50, 100 and 500 features selected by PCA, $\mathcal{X}^2$ and RP.

Table 8: PCA vs $\mathcal{X}^2$ vs RP Test Accuracy Using a Decision Tree

Features	PCA	$\mathcal{X}^2$	RP
50	78.18	49.96	55.03
100	78.15	49.96	56.18
500	78.13	49.96	56.75

The results show that PCA with just 50 features outperforms $\mathcal{X}^2$ and RP. $\mathcal{X}^2$ performs poorly and does not improve as the number of features increases. RP performance increases with the number of features as expected, because it has more features to consider. PCA achieves an improvement of more than a 20% more accuracy than RP.

In view of the results we can conclude that:

- The performance of PCA against RP and $\mathcal{X}^2$ has proven to be better for every tested dataset, achieving with certain datasets up to 10% more accuracy.
- The PCARDE algorithm has shown to be able to work with huge datasets in a short amount of time.
- It is a very stable method for 10 trees. It shows little or no improvement with bigger ensemble sizes.
- It outperforms the original proposal as well as Random Forest for most of the tested datasets. This difference is more noticeable in the Epsilon dataset.

- The computational cost of PCA only becomes noticeable in datasets with a large number of features. For datasets with small number of features, PCA can perform faster than the other methods.

5 Conclusions

In this paper, a new ensemble method is proposed inspired by RPRD Ensemble. It replaces the inconsistency of Random Projections by using a more informative dimensionality reduction method such as PCA. We have proposed a new design for improving the performance and potential applications of this new algorithm.

Thereby we proposed the PCARDE algorithm, a new ensemble method based on PCA for the dimensionality reduction step and Random Discretization, capable of working with Big Data and integrated in Spark's MLlib Library as a third-party package.

The experimental results have demonstrated the stability and improvement in prediction accuracy when using our ensemble solution for the five datasets used. PCARDE learning times have shown to be faster than RPRD and Random Forest for datasets with a small number of features. Additionally, results suggest that PCARDE is very effective for ensembles with a small number of trees.

Acknowledgment

This work is supported by FEDER, the Spanish National Research Project TIN2014-57251-P and TIN2017-89517-P, and the Project BigDaP-TOOLS - Ayudas Fundación BBVA a Equipos de Investigación Científica 2016.

References

[1] Xindong Wu, Xingquan Zhu, Gong-Qing Wu, and Wei Ding. Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):97–107, Jan 2014.

[2] IDC. The Digital Universe of Opportunities. `http://www.emc.com/infographics/digital-universe-2014.htm`, 2014.

[3] Salvador García, Julin Luengo, and Francisco Herrera. *Data Preprocessing in Data Mining*. Springer, 2015.

[4] Salvador García, Julián Luengo, and Francisco Herrera. Tutorial on practical tips of the most influential data preprocessing algorithms in data mining. *Knowledge-Based Systems*, 98:1–29, 2016.

[5] Sergio Ramírez-Gallego, Salvador García, Héctor Mouriño-Talín, David Martínez-Rego, Verónica Bolón-Canedo, Amparo Alonso-Betanzos, José Manuel Benítez, and Francisco Herrera. Data discretization: taxonomy and big data challenge. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 6(1):5–21, 2016.

[6] Jialei Wang, Peilin Zhao, S.C.H. Hoi, and Rong Jin. Online feature selection and its applications. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):698–710, March 2014.

[7] I.T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.

[8] Thomas G. Dietterich. Ensemble Methods in Machine Learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, pages 1–15, London, UK, UK, 2000. Springer-Verlag.

[9] Xu-Ying Liu and Zhi-Hua Zhou. Ensemble methods for class imbalance learning. *Imbalanced Learning: Foundations, Algorithms, and Applications*, pages 61–82.

[10] Yubin Park and J. Ghosh. Ensembles of $(alpha)$-trees for imbalanced classification problems. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):131–143, Jan 2014.

[11] Michał Woźniak, Manuel Graña, and Emilio Corchado. A survey of multiple classifier systems as hybrid systems. *Information Fusion*, 16:3 – 17, 2014.

[12] Lior Rokach. Decision forest: Twenty years of research. *Information Fusion*, 27:111 – 125, 2016.

[13] A. Ahmad and G. Brown. Random Projection Random Discretization Ensembles - Ensembles of Linear Multivariate Decision Trees. *IEEE Transactions on Knowledge and Data Engineering*, 26(5):1225–1239, May 2014.

[14] William B Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[15] Sanjoy Dasgupta. Experiments with random projection. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI'00, pages 143–151, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[16] Dmitriy Fradkin and David Madigan. Experiments with random projections for machine learning. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 517–522, New York, NY, USA, 2003. ACM.

[17] M. Hamstra, H. Karau, M. Zaharia, A. Konwinski, and P. Wendell. *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, 2015.

[18] Apache Spark. Apache Spark: Lightning-fast cluster computing. `http://spark.apache.org/`, 2018.

[19] MLlib. Machine Learning Library (MLlib) for Spark. `http://spark.apache.org/docs/latest/mllib-guide.html`, 2018.

[20] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, October 2001.

[21] Lior Rokach and Oded Maimon. *Data mining with decision trees: theory and applications*. World scientific, 2014.

[22] S García, J Luengo, J Sáez, V López, and F Herrera. A survey of discretization techniques: taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):734–750, 2013.

[23] U. M. Fayyad and K. B. Irani. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In *13th International Joint Conference on Uncertainly in Artificial Intelligence(IJCAI93)*, pages 1022–1029, 1993.

[24] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[25] Alberto Fernández, Sara del Río, Victoria López, Abdullah Bawakid, María J. del Jesus, José M. Benítez, and Francisco Herrera. Big data with cloud computing: an insight on the computing environment, mapreduce, and programming frameworks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(5):380–409, 2014.

[26] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[27] Apache Hadoop Project. Apache Hadoop. `http://hadoop.apache.org/`, 2018.

[28] Jimmy Lin. Mapreduce is good enough? if all you have is a hammer, throw away everything that's not a nail! *Big Data*, 1(1):28–37, 2013.

[29] Diego García-Gil, Sergio Ramírez-Gallego, Salvador García, and Francisco Herrera. A comparison on scalability for batch big data processing on Apache Spark and Apache Flink. *Big Data Analytics*, 2(1):9, Mar 2017.

[30] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 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, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[31] P. Domingos and M. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29:103–137, 1997.

[32] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for Exotic Particles in High-Energy Physics with Deep Learning. *Nature Commun.*, 5:4308, 2014.

[33] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011.

[34] Sara del Río, Victoria López, José Manuel Benítez, and Francisco Herrera. On the use of mapreduce for imbalanced big data using random forest. *Information Sciences*, 285:112 – 137, 2014.

[35] J.J. Rodriguez, L.I. Kuncheva, and C.J. Alonso. Rotation Forest: A New Classifier Ensemble Method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1619–1630, Oct 2006.

[36] Isaac Triguero, Sara del Río, Victoria López, Jaume Bacardit, José M Benítez, and Francisco Herrera. ROSEFW-RF: the winner algorithm for the ECBDL'14 big data competition: an extremely imbalanced big data bioinformatics problem. *Knowledge-Based Systems*, 87:69–79, 2015.

[37] Huan Liu and Rudy Setiono. Chi2: Feature selection and discretization of numeric attributes. In *ICTAI*, pages 388–391, 1995.

[38] Alessio Benavoli, Giorgio Corani, Janez Demsar, and Marco Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *arXiv preprint arXiv:1606.04316*, 2016.

[39] Alessio Benavoli, Giorgio Corani, Francesca Mangili, Marco Zaffalon, and Fabrizio Ruggeri. A bayesian wilcoxon signed-rank test based on the dirichlet process. In *International Conference on Machine Learning*, pages 1026–1034, 2014.

[40] Jacinto Carrasco, Salvador García, María del Mar Rueda, and Francisco Herrera. rnpbst: An r package covering non-parametric and bayesian statistical tests. In Francisco Javier Martínez de Pisón, Rubén Urraca, Héctor Quintián, and Emilio Corchado, editors, *Hybrid Artificial Intelligent Systems*, pages 281–292, Cham, 2017. Springer International Publishing.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[43] Hervé Abdi and Lynne J. Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

3 Enabling Smart Data: Noise Filtering in Big Data Classification

- D. García-Gil, J. Luengo, S. García, F. Herrera. Information Sciences 479, 135-152 (2019).
  - Status: **Published**.
  - Impact Factor (JCR 2018): **5.524**
  - Subject Category: **Computer Science, Information Systems**
  - Rank: **9/155**
  - Quartile: **Q1**

ENABLING SMART DATA: NOISE FILTERING IN BIG DATA CLASSIFICATION

Diego García-Gil[*]
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
djgarcia@decsai.ugr.es

Julián Luengo
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
julianlm@decsai.ugr.es

Salvador García
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
salvagl@decsai.ugr.es

Francisco Herrera
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
herrera@decsai.ugr.es

ABSTRACT

In any knowledge discovery process the value of extracted knowledge is directly related to the quality of the data used. Big Data problems, generated by massive growth in the scale of data observed in recent years, also follow the same dictate. A common problem affecting data quality is the presence of noise, particularly in classification problems, where label noise refers to the incorrect labeling of training instances, and is known to be a very disruptive feature of data. However, in this Big Data era, the massive growth in the scale of the data poses a challenge to traditional proposals created to tackle noise, as they have difficulties coping with such a large amount of data. New algorithms need to be proposed to treat the noise in Big Data problems, providing high quality and clean data, also known as Smart Data. In this paper, two Big Data preprocessing approaches to remove noisy examples are proposed: an homogeneous ensemble and an heterogeneous ensemble filter, with special emphasis in their scalability and performance traits. The obtained results show that these proposals enable the practitioner to efficiently obtain a Smart Dataset from any Big Data classification problem.

***K*eywords** Big Data · Smart Data · Classification · Class Noise · Label Noise.

1 Introduction

Vast amounts of information surround us today. Technologies such as the Internet generate data at an exponential rate thanks to the affordability and great development of storage and network resources. It is

[*]Corresponding author.

predicted that by 2020, the digital universe will be 10 times as big as it was in 2013, totaling an astonishing 44 zettabytes. The current volume of data has exceeded the processing capabilities of classical data mining systems [1] and have created a need for new frameworks for storing and processing this data. It is widely accepted that we have entered the Big Data era. Big Data is the set of technologies that make processing such large amounts of data possible [2], while most of the classic knowledge extraction methods cannot work in a Big Data environment because they were not conceived for it.

Big Data as concept is defined around five aspects: data volume, data velocity, data variety, data veracity and data value. While the volume, variety and velocity aspects refer to the data generation process and how to capture and store the data, veracity and value aspects deal with the quality and the usefulness of the data. These two last aspects become crucial in any Big Data process, where the extraction of useful and valuable knowledge is strongly influenced by the quality of the used data.

In Big Data, the usage of traditional preprocessing techniques [3, 4, 5] to enhance the data is even more time consuming and resource demanding, being unfeasible in most cases. The lack of efficient and affordable preprocessing techniques implies that the problems in the data will affect the models extracted. Among all the problems that may appear in the data, the presence of *noise* in the dataset is one of the most frequent. Noise can be defined as the partial or complete alteration of the information gathered for a data item, caused by an exogenous factor not related to the distribution that generates the data. Learning from noisy data is an important topic in machine learning, data mining and pattern recognition, as real world data sets may suffer from imperfections in data acquisition, transmission, storage, integration and categorization. Noise will lead to excessively complex models with deteriorated performance [6], resulting in even larger computing times for less value.

The impact of noise in Big Data, among other pernicious traits, has not been disregarded. Recently, Smart Data (focusing on veracity and value) has been introduced, aiming to filter out the noise and to highlight the valuable data, which can be effectively used by companies and governments for planning, operation, monitoring, control, and intelligent decision making. Three key attributes are needed for data to be smart, it must be accurate, actionable and agile:

- Accurate: data must be what it says it is with enough precision to drive value. Data quality matters.
- Actionable: data must drive an immediate scalable action in a way that maximizes a business objective like media reach across platforms. Scalable action matters.
- Agile: data must be available in real-time and ready to adapt to the changing business environment. Flexibility matters.

Advanced Big Data modeling and analytics are indispensable for discovering the underlying structure from retrieved data in order to acquire Smart Data. In this paper we provide several preprocessing techniques for Big Data, transforming raw, corrupted datasets into Smart Data. We focus our interest on classification tasks, where two types of noise are distinguished: *class noise*, when it affects the class label of the instances, and *attribute noise*, when it affects the rest of attributes. The former is known to be the most disruptive [7, 8]. Consequently, many recent works, including this contribution, have been devoted to resolving this problem or at least to minimize its effects (see [9] for a comprehensive and updated survey).

While some architectural designs are already proposed in the literature[10], there is no particular algorithm which deals with noise in Big Data classification, nor a comparison of its effect on model generalization abilities or computing times.

Thereby we propose a framework for Big Data under Apache Spark for removing noisy examples composed of two algorithms based on ensembles of classifiers. The first one is an homogeneous ensemble, named

Homogeneous Ensembe for Big Data (HME-BD), which uses a single base classifier (Random Forest) over a partitioning of the training set. The second ensemble is an heterogeneous ensemble, namely Heterogeneous Ensembe for Big Data (HTE-BD), that uses different classifiers to identify noisy instances: Random Forest, Logistic Regression and K-Nearest Neighbors (KNN) as base classifiers. For the sake of a more complete comparison, we have also considered a simple filtering approach based on similarities between instances, named Edited Nearest Neighbor for Big Data (ENN-BD). ENN-BD examines the nearest neighbors of every example in the training set and eliminates those whose majority of neighbors belong to a different class. All these techniques have been implemented under the Apache Spark framework [11] and can be downloaded from the Spark's community repository [2].

To show the performance of the three proposed algorithms, we have carried out an experimental evaluation with four large datasets, namely *SUSY*, *HIGGS*, *Epsilon* and *ECBDL14*. We have induced several levels of class noise to evaluate the effects of applying such framework and the improvements obtained in terms of classification accuracy for two classifiers: a decision tree and the KNN technique. Decision trees with pruning are known to be tolerant to noise, while KNN is a noise sensitive algorithm when the number of selected neighbors is low. These differences allow us to better compare the effect of the framework in classifiers which behave differently towards noise. We also show that, for the Big Data problems considered, the classifiers also benefit from applying the noise treatment even when no additional noise is induced, since Big Data problems contain implicit noise due to incidental homogeneity, spurious correlations and the accumulation of noisy examples [12]. The results obtained indicate that the framework proposed can successfully deal with noise. In particular, the homogeneous ensemble is the first suitable technique for dealing with noise in Big Data problems, with low computing times and enabling the classifier to achieve better accuracy.

The remainder of this paper is organized as follows: Section 2 presents the concepts of noise, MapReduce and Smart Data. Section 3 explains the proposed framework. Section 4 describes the experiments carried out to check the performance of the framework. Finally, Section 5 concludes the paper.

2 Related work

In this section we first present the problem of noise in classification tasks in Section 2.1. Then we introduce the MapReduce framework commonly used in Big Data solutions in Section 2.2. Finally, we provide an insight into Smart Data in 2.3.

2.1 Class noise vs. attribute noise

In a classification problem, several effects of this noise can be observed by analyzing its spatial characteristics: noise may create small clusters of instances of a particular class in the instance space corresponding to another class, displace or remove instances located in key areas within a concrete class, or disrupt the boundaries of the classes resulting in an increased boundaries overlap. All these imperfections may harm data interpretation, the design, size, building time, interpretability and accuracy of models, as well as decision making [8].

As described by Wang et al. [13], from the large number of components that comprise a dataset, class labels and attribute values are two essential elements in classification datasets. Thus, two types of noise are commonly differentiated in the literature [8, 13]:

[2]`https://spark-packages.org/package/djgarcia/NoiseFramework`

- *Class noise*, also known as *label noise*, takes place when an example is wrongly labeled. Class noise includes contradictory examples [7] (examples with identical input attribute values having different class labels) and misclassifications [8] (examples which are incorrectly labeled).
- *Attribute noise* refers to corruptions in the values of the input attributes. It includes erroneous attribute values, missing values and incomplete attributes or "do not care" values. Missing values are usually considered independently in the literature, so *attribute noise* is mainly used for erroneous values [8].

Class noise is generally considered more harmful to the learning process, and methods for dealing with class noise are more frequent in the literature [8]. Class noise may have many reasons, such as errors or subjectivity in the data labeling process, as well as the use of inadequate information for labeling. Data labeling by domain experts is generally costly, and automatic taggers are used (e.g., sentiment analysis polarization [14]), increasing the probability of class noise.

Due to the increasing attention from researchers and practitioners, numerous techniques have been developed to tackle it [9, 8, 3]. These techniques include learning algorithms robust to noise as well as data preprocessing techniques that remove or "repair" noisy instances. In [9] the mechanisms that generate label noise are examined, relating them to the appropriate treatment procedures that can be safely applied:

- On the one hand, *algorithm level* approaches attempt to create robust classification algorithms that are little influenced by the presence of noise. This includes approaches where existing algorithms are modified to cope with label noise by either being modeled in the classifier construction [15], by applying pruning strategies to avoid overfiting or by diminishing the importance of noisy instances with respect to clean ones [16]. Recent proposals exist which that combine these two approaches, which model the noise and give less relevance to potentially noisy instances in the classifier building process [17].
- On the other hand, *data level* approaches (also called *filters*) try to develop strategies to cleanse the dataset as a previous step to the fit of the classifier, by either creating ensembles of classifiers [18], partitioning the data [19], iteratively filtering noisy instances [20], computing metrics on the data or even hybrid approaches that combine several of these strategies.

In the Big Data environment there is a special need for noise filter methods. It is well known that the high dimensionality and example size generate accumulated noise in Big Data problems [12]. Noise filters reduce the size of the datasets and improve the quality of the data by removing noisy instances, but most of the classic algorithms for noisy data, noise filters in particular, are not prepared for working with huge volumes of data as they have an iterative approach.

2.2 Big Data. MapReduce and Apache Spark

The globalization of the Big Data paradigm is generating a large response in terms of technologies that must deal with the rapidly growing rates of generated data [21]. Among all of them, MapReduce is the seminal framework designed by Google in 2003 [22, 23]. It follows a divide and conquer approach to process and generate large datasets with parallel and distributed algorithms on a cluster. The MapReduce model is composed of two phases: Map and Reduce. The Map phase performs a transformation of the data, and the Reduce phase performs a summary operation. Briefly explained, first the master node splits the input data and distributes it across the cluster. Then the Map transformation is applied to each key-value pair in the local data. Once that process is finished the data is redistributed based on the key-value pairs generated in the Map phase. Once all pairs belonging to one key are in the same node, it is processed in parallel. Apache Hadoop [24] is the most popular open-source framework based on the MapReduce model.

Apache Spark [11] is an open-source framework for Big Data processing built around speed, ease of use and sophisticated analytics. Its main feature is its ability to use in-memory primitives. Users can load their data into memory and iterate over it repeatedly, making it a suitable tool for ML algorithms. The motivation for developing Spark came from the limitations in the MapReduce/Hadoop model [23, 25, 26]:

- Intensive disk usage
- Insufficiency for in-memory computation
- Poor performance on online and iterative computing.
- Low inter-communication capacity.

Spark is built on top of a distributed data structure called Resilient Distributed Datasets (RDDs) [27]. Operations on RDDs are applied to each partition of the node local data. RDDs support two types of operations: transformations, which are not evaluated when defined and produce a new RDD, and actions, which evaluate all the previous transformations and return a new value. The RDD structure allows programmers to persist them into memory or disk for re-usability purposes. RDDs are immutable and fault-tolerant by nature. All operations are tracked using a ”lineage”, so that each partition can be recalculated in case of failure.

Although new promising frameworks for Big Data are emerging, like Apache Flink [28], Apache Spark is becoming the reference in performance [29, 23].

2.3 From Big Data to Smart Data

Big Data is an appealing discipline that presents an immense potential for global economic growth and promises to enhance competitiveness of high technological countries. Such as occurs in any knowledge extraction process, vast amounts of data are analyzed, processed, and interpreted in order to generate profits in terms of either economic or advantages for society. Once the Big Data has been analyzed, processed, interpreted and cleaned, it is possible to access it in a structured way. This transformation is the difference between “Big” and “Smart” Data [30].

The first step in this transformation is to perform an integration process, where the semantics and domains from several large sources are unified under a common structure. The usage of ontologies to support the integration is a recent approach [31], but graph databases are also an option where the data is stored in a relational form, as in healthcare domains [32]. Even when the integration phase ends, the data is still far from being “smart”: the accumulated noise in Big Data problems creates problems in classical Data Mining techniques, specially when the dimensionality is large [33]. Thus, in order to be “smart”, the data still needs to be cleaned even after its integration, and data preprocessing is the set of techniques utilized to encompass this task [3, 34].

Once the data is “smart”, it can hold the valuable data and allows interactions in “real time”, like transactional activities and other Business Intelligence applications. The goal is to evolve from a data-centered organization to a learning organization, where the focus is set on the knowledge extracted instead of struggling with the data management [35]. However, Big Data generates great challenges to achieve this since its high dimensionality and large example size imply noise accumulation, algorithmic instability and the massive sample pool is often aggregated from heterogeneous sources [12]. While feature selection, discretization or imbalanced algorithms to cope with the high dimensionality have drawn the attention of current Big Data frameworks (such as Spark's MLlib [36]) and researchers [37, 38, 39], algorithms to clean noise are still a challenge. In summary, challenges are still present to fully operate a transition between Big Data to Smart Data. In this paper we provide an automated preprocessing framework to deal with class noise, enabling the practitioner to reach Smart Data.

3 Towards Smart Data: Noise filtering for Big Data

In this section, we present the first suitable framework for Big Data under Apache Spark for removing noisy examples based on the MapReduce paradigm, proving its performance over real-world large problems. It is a MapReduce design where all the noise filter processes are performed in a distributed way.

In Section 3.1 we describe the Spark primitives used for the implementation of the framework. In Section 3.2 we explain in detail the classification algorithms used in the implementation of the framework. We have designed two algorithms based on ensembles. Both perform a k-fold on the training data, learn a model on the training partition and clean noisy instances in the test partition. The first one is an homogeneous ensemble using Random Forest as a classifier, named HME-BD (Section 3.3). The second one, named HTE-BD (Section 3.4) is a heterogeneous ensemble based on the use of three different classifiers: Random Forest, Logistic Regression and KNN. We have also implemented a simple filter based on the similarity between the instances, named ENN-BD (Section 3.5).

3.1 Spark Primitives

For the implementation of the framework, we have used some basic Spark primitives from Spark API. These primitives offer much complex operations by extending the MapReduce paradigm. Here, we outline those more relevant to the algorithms [3]:

- map: Applies a transformation to each element of a RDD. Once the operation has been performed to each element, the resulting RDD is returned.
- $zipWithIndex$: for each element of a RDD, a pair consisting in the element and its index is created, starting at 0. The resulting RDD is then returned.
- $join$: Return a RDD containing all pairs of elements with matching keys between two RDDs.
- $filter$: Return a new RDD containing only the elements that satisfy a predicate.
- $union$: Return a RDD of pairs as result of the union of two RDDs.
- $kFold$: Returns a list of k pairs of RDDs with the first element of each pair containing the $train$ data, a complement of the $test$ data, and the second element containing the $test$ data, being a unique 1/kth of the data. Where k is the number of folds.
- $randomForest$: Method to learn a Random Forest model for classification problems.
- $predict$: Returns a RDD containing the features and the predicted labels for a given dataset using the learned model.
- $learnClassifiers$: Although its not a pure Spark primitive, we use it to simplify the description of the algorithms. This primitive learns a Random Forest, Logistic Regression and 1NN models from the input data.

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

These Spark primitives from Spark API are used in the following sections where HME-BD, HTE-BD and ENN-BD algorithms are described.

3.2 Classification Algorithms

In this section we describe in detail the classification algorithms used in the implementation of the framework.

3.2.1 Decision Tree

Decision trees are one of the most popular methods in machine learning for both classification and regression tasks. They are easy to interpret, can handle categorical features and extend to the multiclass classification problem among other features.

A decision tree uses a tree-like graph for decision making. It starts with a single node which divides into possible outcomes. Each of those outcomes leads to additional nodes, which in turn are divided into other nodes. The end nodes are the decision of a certain branch of the tree.

Spark's implementation of the decision tree is optimized for scalability. The key optimizations are: level-wise training, for selecting the splits for all nodes at the same level of the tree, approximate quantiles, bin-wise computation, for saving computation on each iteration by precomputing the binned representations of each instance, and the avoiding of the map operation.

3.2.2 Random Forest

Ensembles are algorithms that combines a set of models build upon other machine learning algorithms. Random Forests are a combination of decision trees where each tree is trained independently using a random sample of the data.

Spark's Random Forest implementation builds upon the decision tree code, which distributes the learning of single trees. Many of the optimizations are based upon Google's PLANET project [40]. Random Forests are easily paralleled since each tree can be trained independently. Spark's Random Forest does exactly that, a variable number of sub-trees are trained in parallel.

3.2.3 KNN

KNN is a supervised learning method typically used for classification. It is based on a learning through close examples in the space of the elements.

Since Spark doesn't have a KNN implementation, we have used an exact implementation of KNN present in Spark's community repository kNN-IS[4] [41]. This implementation takes advantage of Spark's in-memory operations for improving the scalability of the KNN algorithm.

3.2.4 Logistic Regression

Logistic Regression is a linear method widely used to predict a binary response. The loss function is given by the logistic loss.

Spark's implementation of the logistic regression algorithm uses the limited-memory BFGS (L-BFGS) algorithm [42] for optimization of the memory used.

[4]`https://spark-packages.org/package/JMailloH/kNN_IS`

Algorithm 1 HME-BD Algorithm

```
1: **Input:** *data* a RDD of tuples (label, features)
2: **Input:** *P* the number of partitions
3: **Input:** *nTrees* the number of trees for Random Forest
4: **Output:** the filtered RDD without noise
5: $partitions \leftarrow kFold(data, P)$
6: $filteredData \leftarrow \emptyset$
7: for all $train, test$ in $partitions$ do
8:     $rfModel \leftarrow randomForest(train, nTrees)$
9:     $rfPred \leftarrow predict(rfModel, test)$
10:    $joinedData \leftarrow join(zipWithIndex(test), zipWithIndex(rfPred))$
11:    $markedData \leftarrow$
12:    map $original, prediction \in joinedData$
13:        if $label(original) = label(prediction)$ then
14:            $original$
15:        else
16:            $(label = \emptyset, features(original))$
17:        end if
18:    end map
19:    $filteredData \leftarrow union(filteredData, markedData)$
20: end for
21: $return(filter(filteredData, label \neq \emptyset))$
```

3.3 Homogeneous Ensemble: HME-BD

The homogeneous ensemble is inspired by Cross-Validated Committees Filter (CVCF) [19]. This filter removes noisy examples by partitioning the data in P subsets of equal size. Then, a decision tree, such as C4.5, is learned P times, each time leaving out one of the subsets of the training data. This results in P classifiers which are used to predict all the training data P times. Then, using a voting strategy, misclassified instances are removed.

HME-BD is also based on a partitioning scheme of the training data. There is an important difference with respect to CVCF: the use of Spark's implementation of Random Forest instead a of a decision tree as a classifier. CVCF creates an ensemble from partitioning of the training data. HME-BD also partitions the training data, but the use of Random Forest allows us to improve the voting step:

- CVCF predicts the whole dataset P times. We only predict the instances of the partition that Random Forest has not seen while learning the model. This step is repeated P times. With this change we not only improve the performance, but also the computing time of the algorithm since it only has to predict a small part of the training data each iteration.
- We don't need to implement a voting strategy, the decision of whether an instance is noisy is associated with the Random Forest prediction.

Algorithm 1 describes the noise filtering process in HME-BD:

- The algorithm filters the noise in a dataset by performing a $kFold$ on the training data. As stated previously, Spark's $kFold$ function returns a list of $(train, test)$ for a given P, where $test$ is a unique 1/kth of the data, and $train$ is a complement of the $test$ data.

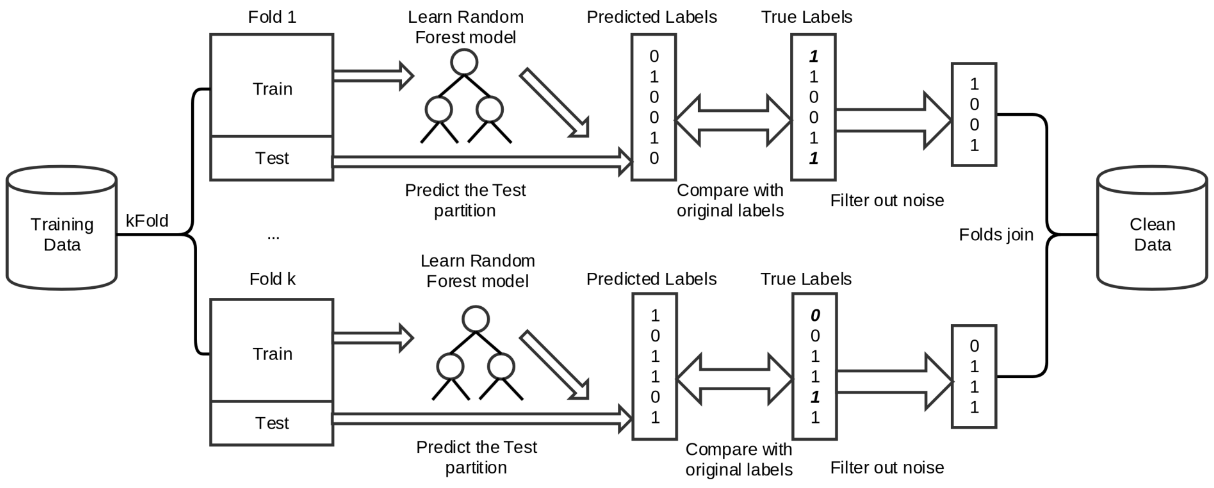


Figure 1: HME-BD noise filtering process flowchart

- We iterate through each partition, learning a Random Forest model using the $train$ as input data and predicting the $test$ using the learned model.
- In order to join the $test$ data and the predicted data for comparing the classes, we use the $zipWithIndex$ operation in both RDDs. With this operation, we add an index to each element of both RDDs. This index is used as key for the join operation.
- The next step is to apply a Map function to the previous RDD in order to check for each instance the original class and the predicted one. If the predicted class and the original are different, the instance is marked as noise.
- The result of the previous Map function is a RDD where noisy instances are marked. These instances are finally removed using a $filter$ function and the resulting dataset is returned.

The following are required as input parameters: the dataset ($data$), the number of partitions (P) and the number of trees for the Random Forest ($nTrees$).

In Figure 1 we can see a flowchart of the HME-BD noise filtering process.

The computational complexity of the algorithm is reduced to the Random Forest learning and prediction time complexity. As theoretically proved in [43] Random Forest's computational complexity is: $O(MK\tilde{N}\log\tilde{N})$, being M the number of randomized trees, K the number of variables randomly drawn at each node, N denotes the number of samples of the training partition, and $\tilde{N} = 0.632N$ due to the fact that bootstrap samples draw, on average, 63.2% of unique samples. The prediction of the Random Forest is $O(M\log L)$, being L the number of samples in the test partition. We repeat this process P times, so the final computational complexity is $O(P(MK\tilde{N}\log\tilde{N})) + O(P(M\log L))$.

3.4 Heterogeneous Ensemble: HTE-BD

Heterogeneous Ensemble is inspired by Ensemble Filter (EF) [18]. This noise filter uses a set of three learning algorithms for identifying mislabeled instances in a dataset: a univariate decision tree (C4.5), KNN and a linear machine. It performs a k-fold cross validation over the training data. For each one of the k parts, three algorithms are trained on the other $k-1$ parts. Each of the classifiers is used to tag each of the

Algorithm 2 HTE-BD Algorithm

```
1: Input: data a RDD of tuples (label, features)
2: Input: P the number of partitions
3: Input: nTrees the number of trees for Random Forest
4: Input: vote the voting strategy (majority or consensus)
5: Output: the filtered RDD without noise
6: $partitions \leftarrow kFold(data, P)$
7: $filteredData \leftarrow \emptyset$
8: for all $train, test$ in $partitions$ do
9:     $classifiersModel \leftarrow learnClassifiers(train, nTrees)$
10:    $predictions \leftarrow predict(classifiersModel, test)$
11:    $joinedData \leftarrow join(zipWithIndex(predictions), zipWithIndex(test))$
12:    $markedData \leftarrow$
13:    map $rf, lr, knn, orig \in joinedData$
14:        $count \leftarrow 0$
15:        if $rf \neq label(orig)$ then $count \leftarrow count + 1$ end if
16:        if $lr \neq label(orig)$ then $count \leftarrow count + 1$ end if
17:        if $knn \neq label(orig)$ then $count \leftarrow count + 1$ end if
18:        if $vote = majority$ then
19:            if $count \geq 2$ then $(label = \emptyset, features(orig))$ end if
20:            if $count < 2$ then $orig$ end if
21:        else
22:            if $count = 3$ then $(label = \emptyset, features(orig))$ end if
23:            if $count \neq 3$ then $orig$ end if
24:        end if
25:    end map
26:    $filteredData \leftarrow union(filteredData, markedData)$
27: end for
28: $return(filter(filteredData, label \neq \emptyset))$
```

$test$ examples as noisy or clean. At the end of the k-fold, each example of the input data has been tagged. Finally, using a voting strategy, a decision is made and noisy examples are removed.

HTE-BD follows the same working scheme as EF. The main difference is the choice of the three learning algorithms:

- Instead of a decision tree, we use Spark's implementation of Random Forest.
- We use an exact implementation of KNN with the euclidean distance present in Spark's community repository.
- The linear machine has been replaced by Spark's implementation of Logistic Regression, which is another linear classifier.

The noise filtering process in HTE-BD is shown in Algorithm 2:

- For each $train$ and $test$ partition of the k-fold performed to the input data, it learns three classification algorithms: Random Forest, Logistic Regression and 1NN using the $train$ as input data.

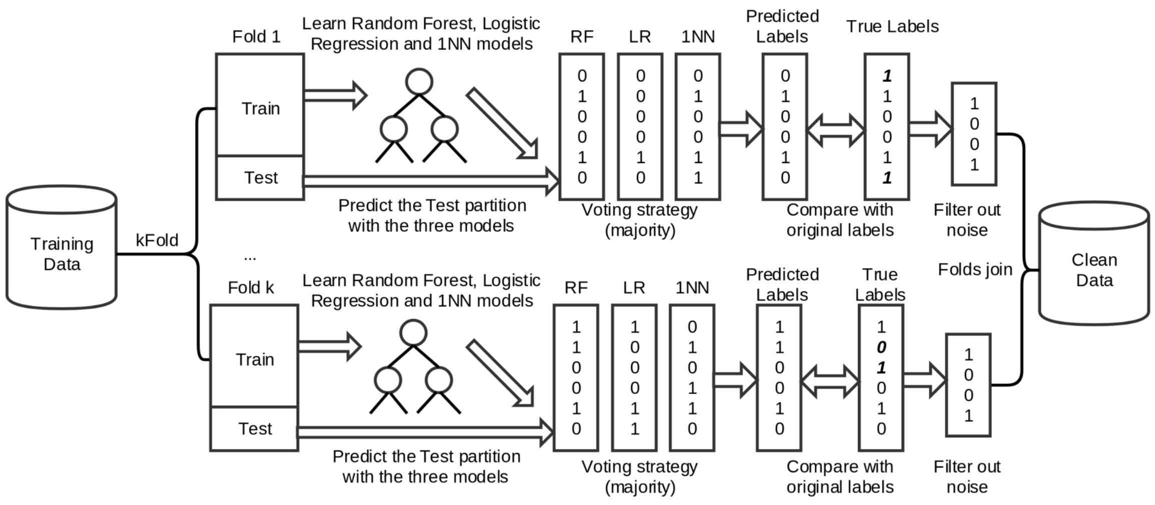

Training Data
kFold
Fold 1
Train
Test
Learn Random Forest, Logistic Regression and 1NN models
Predict the Test partition with the three models
RF
LR
1NN
Voting strategy (majority)
Predicted Labels
True Labels
Compare with original labels
Filter out noise
Folds join
Clean Data
...
Fold k

Figure 2: HTE-BD noise filtering process flowchart

- Then it predicts the $test$ data using the three learned models. This creates a RDD of triplets (rf, lr, knn) with the prediction of each algorithm for each instance.
- The predictions and the $test$ data are joined by index in order to compare the predictions and the original label.
- It compares the three predictions of each instance in the $test$ data with the original label using a Map function and, depending upon the voting strategy, the instance is marked as noise or clean.
- Once the Map function has been applied to each instance, noisy data is removed using a $filter$ function and the dataset is returned.

The following are required as input parameters: the dataset ($data$), the number of partitions (P), the number of trees for the Random Forest ($nTrees$) and the voting strategy ($vote$).

In Figure 2 we show a flowchart of the HTE-BD noise filtering process.

The computational complexity of HTE-BD is defined by the three classifiers used. As explained previously, time complexity of learning and predicting a Random Forest is $O(MK\tilde{N}\log\tilde{N}) + O(M\log L)$. kNN-IS is internally executed in two steps [41], first a mapPartitions phase is performed with a computational complexity of $O(MN)$ for finding the nearest neighbor training example of a single test instance. Finally, a reduce phase is performed whose complexity can be despised. Lastly, Logistic Regression time complexity depends upon the internal optimizer used, in our case L-BFGS. L-BFGS computational complexity is given by $O(NT)$, being T the number of iterations. In summary, HTE-BD computational complexity can be defined as $O(P(MK\tilde{N}\log\tilde{N})) + O(P(M\log L)) + O(P(MN)) + O(P(NT))$.

3.5 Similarity: ENN-BD

ENN-BD is a simple filtering algorithm that works as a baseline for comparison purposes. It has been designed based on the Edited Nearest Neighbor algorithm (ENN) [44] and follows a similarity between

Algorithm 3 ENN-BD Algorithm

```
 1: Input: data a RDD of tuples (label, features)
 2: Output: the filtered RDD without noise
 3: $knnModel \leftarrow KNN(1, "euclidean", data)$
 4: $knnPred \leftarrow zipWithIndex(predict(knnModel, data))$
 5: $joinedData \leftarrow join(zipWithIndex(data), knnPred)$
 6: $filteredData \leftarrow$
 7: map $original, prediction \in joinedData$
 8:     if $label(original) = label(prediction)$ then
 9:         $original$
10:     else
11:         $(noise, features(original))$
12:     end if
13: end map
14: $return(filter(filteredData, label \neq noise))$
```

instances approach. ENN removes noisy instances in a dataset by comparing the label of each example with its closest neighbor. If the labels are different, the instance is considered as noisy and removed.

ENN-BD performs a 1NN using Spark's community repository kNN-IS with the euclidean distance. It checks for each instance if its closest neighbor belongs to the same class. In case the classes are different, the instance is marked as noise. Finally, marked instances are removed from the training data. This process is described in Algorithm 3. The only input parameter required is the dataset ($data$).

Computational complexity of ENN-BD is reduced to the time complexity of KNN. As described in HTE-BD, computational complexity of kNN-IS is $O(MN)$.

4 Experimental Results

This section describes the experimental details and the analysis carried out to show the performance of the three noise filter methods over four huge problems. In Section 4.1, we present the details of the datasets and the parameters used in the methods. We analyze the accuracy improvements generated by the proposed framework and the study of instances removed in Section 4.2. Finally, Section 4.3 is devoted to the computing times of the proposals.

4.1 Experimental Framework

Four classification datasets are used in our experiments:

- SUSY dataset, which consists of 5,000,000 instances and 18 attributes [45]. The first eight features are kinematic properties measured by the particle detectors at the Large Hadron Collider. The last ten are functions of the first eight features. The task is to distinguish between a signal process which produces supersymmetric (SUSY) particles and a background process which does not [46].

- HIGGS dataset, which has 11,000,000 instances and 28 attributes [45]. This dataset is a classification problem to distinguish between a signal process which produces Higgs bosons and a background process which does not.

Table 1: Datasets used in the analysis

Dataset	Instances	Atts.	Total	CL
SUSY	5,000,000	18	90,000,000	2
HIGGS	11,000,000	28	308,000,000	2
Epsilon	500,000	2,000	1,000,000,000	2
ECBDL14	1,000,000	631	631,000,000	2

Table 2: Parameter setting for the noise filters

Algorithm	Parameters	Classifiers
HME-BD	P = 4, 5	Random Forest: featureSubsetStrategy = ”auto”, impurity = ”gini”, maxDepth = 10 and maxBins = 32
HTE-BD	P = 4, 5 Voting = majority, consensus	1NN, Random Forest: featureSubsetStrategy = ”auto”, impurity = ”gini”, maxDepth = 10 and maxBins = 32
ENN-BD	K = 1	distance = ”euclidean”

- Epsilon dataset, which consists of 500,000 instances with 2,000 numerical features. This dataset was artificially created for the Pascal Large Scale Learning Challenge in 2008. It was further pre-processed and included in the LibSVM dataset repository [47].
- ECBDL14 dataset, which has 32 million instances and 631 attributes (including both numerical and categorical) [39]. This dataset 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. It is a binary classification problem where the class distribution is highly imbalanced: 98% of negative instances. For this problem, we use a reduced version with 1,000,000 instances and 30% of positive instances.

Table 1 provides a brief summary of these datasets, showing the number of examples (Instances), the total number of attributes (Atts.), the total number of training data (Total), and the number classes (CL).

We carried out experiments on five levels of uniform class noise: for each level of noise, a percentage of the training instances are altered by replacing their actual label by another label from the available classes. The selected noise levels are 0%, 5%, 10%, 15% and 20%. In this case, a 0% noise level indicates that the dataset was unaltered. We have conducted a hold-out validation due to the time limitations of the KNN algorithm.

In Table 2 we can see the complete list of parameters used for the noise treatment algorithms. In order to evaluate the effect of the number of partitions on the behavior of the filters, we have selected 4 and 5 training partitions for HME-BD and HTE-BD. For the heterogeneous filter, HTE-BD, we also use two voting strategies: consensus (same result for all classifiers) and majority (same result for at least half the classifiers). For ENN-BD, with *k = 5* we have higher network overload, and due to the huge data redundancy in big datasets, it achieved similar performance in internal tests in comparison to *k = 1*. This is why we have chosen the most efficient option.

Two classifiers, one MLlib classifier, a decision tree, and one algorithm present in Spark's community repository, KNN, are used to evaluate the effectiveness of the filtering carried out by the two ensemble proposals and the similarity filter. The decision tree can adapt its depth to avoid overfitting to noisy instances, while KNN is known to be sensitive to noise when the number of selected neighbors is low. Prediction

Table 3: Parameter setting for the classifiers

Classifier	Parameters
KNN	K = 1, distance = ”euclidean”
Decision Tree	impurity = ”gini”, maxDepth = 20 and maxBins = 32

accuracy is used to evaluate the model’s performance produced by the classifiers (number of examples correctly labeled as belonging to a given class divided by the total number of elements). The parameters used for the classifiers can be seen in Table 3. Default parameters are used, except for the decision tree, in which we have tuned the depth of the tree for a better detection of noisy instances. KNN is used with *k = 1* as it is more sensitive to noise, as opposed to the decision tree.

The experiments have been carried out according to the following scheme:

- No noise filtering: for each level of noise (from 0% to 20 %) we learn a KNN and a Decision Tree using the training partition of the data and then predict the test partition.
- Noise filtering: for each level of noise (from 0% to 20 %) we filter the training partition of the data using the corresponding filter. Then we learn a KNN and a Decision Tree using the filtered dataset and predict the test partition.

For all experiments we have used a cluster composed of 20 computing nodes and one master node. The computing nodes hold the following characteristics: 2 processors x Intel(R) Xeon(R) CPU E5-2620, 6 cores per processor, 2.00 GHz, 2 TB HDD, 64 GB RAM. Regarding software, we have used the following configuration: Hadoop 2.6.0-cdh5.4.3 from Cloudera’s open source Apache Hadoop distribution, Apache Spark and MLlib 1.6.0, 460 cores (23 cores/node), 960 RAM GB (48 GB/node).

4.2 Analysis of accuracy performance and removed instances

In this section, we present the analysis on the performance results obtained by the selected classifiers after applying the proposed framework. We denote with *Original* the application of the classifier without using any noise treatment techniques, in order to evaluate the impact of the increasing noise level in the quality of the models extracted by the classification algorithms.

Table 4 shows the test accuracy values for the four datasets and the five levels of noise using the KNN algorithm for classification. From these results we can point out that:

- It is important to remark that the usage of any noise treatment technique always improves the *Original* accuracy value at the same noise level. Please note that the usage of the noise treatment technique allows KNN to obtain better performance at any noise level, even at the highest ones, than *Original* at 0% level for every dataset. Since Big Datasets tend to accumulate noise, the proposed noise framework is able to improve the behavior and performance of the KNN classifier in every case.
- If we attend the best noise treatment strategy for KNN, we must point out that the homogeneous filter, HME-BD, enables KNN to obtain the highest accuracy values.
- The different number of partitions used for HME-BD has little impact in the accuracy values, which, in this respect, makes it a robust method.

Table 4: KNN test accuracy. The highest accuracy value per dataset and noise level is stressed in bold

Dataset	Noise (%)	Original	HME-BD		HTE-BD				ENN-BD
P			4	5	4	4	5	5	
Vote					Majority	Consensus	Majority	Consensus	
SUSY	0	71.79	**78.73**	78.72	77.86	74.64	77.88	74.65	72.02
	5	69.62	78.68	**78.69**	77.68	73.38	77.68	73.39	69.84
	10	67.44	**78.63**	78.62	77.44	72.01	77.46	72.00	67.66
	15	65.27	**78.62**	78.61	77.19	70.52	77.20	70.53	65.28
	20	63.10	78.56	**78.58**	76.93	69.10	76.93	69.04	63.25
HIGGS	0	61.21	**64.26**	64.25	63.94	62.30	63.93	62.23	60.65
	5	60.10	64.06	**64.07**	63.63	61.45	63.62	61.44	59.60
	10	58.97	63.83	**63.84**	63.29	60.65	63.24	60.66	58.56
	15	57.84	**63.65**	63.64	62.86	59.81	62.89	59.81	57.52
	20	56.69	**63.53**	63.40	62.55	58.89	62.55	58.85	56.45
Epsilon	0	56.55	**58.11**	58.06	57.43	55.19	57.39	55.40	56.21
	5	55.71	**58.64**	58.60	57.47	55.47	57.39	55.41	55.43
	10	55.20	58.51	**58.61**	57.26	55.25	57.26	55.25	54.79
	15	54.54	58.39	**58.41**	57.00	55.00	57.02	55.03	54.30
	20	54.05	58.02	**58.09**	56.75	54.72	56.71	54.72	53.68
ECBDL14	0	74.83	**76.06**	76.03	75.12	73.54	75.14	73.46	73.94
	5	72.36	**75.60**	75.59	74.59	72.89	74.59	72.84	72.77
	10	69.86	75.31	**75.32**	74.19	72.50	74.19	72.47	71.40
	15	67.39	75.11	**75.12**	73.99	72.11	74.01	72.06	69.68
	20	64.90	74.82	**74.83**	73.70	71.89	73.70	71.90	67.64

- The heterogeneous ensemble filter, HTE-BD, is also robust to the number of partitions chosen, but its performance is lower than HME-BD. However, the voting scheme is crucial for HTE-BD, as the consensus strategy will result in worse accuracy for KNN, being close to 2% less accuracy for the consensus voting strategy.
- While the *Original* accuracy value drops around 2% for each 5% increment of noise, totaling about 10% less accuracy in 20% of noise level, HME-BD drops less than a 1% of accuracy in total.
- The baseline noise filtering method, ENN-BD, is the worst option as KNN obtains the lowest accuracy values among the three noise treatment strategies. For ENN-BD, the accuracy drops around 2% for each 5% increment in noise instances. However, as mentioned earlier, ENN-BD is still preferable to not dealing with the noise at all. This is due to the noise sensitive nature of KNN.

Table 5 gathers the test accuracy values for the three noise filter methods using a deep decision tree. From these results we can point out that:

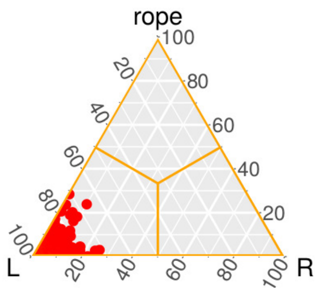
- Again, avoiding the treatment of noise is never the best option and using the appropriate noise filtering technique will provide a significant improvement in accuracy. However, since the decision tree is more robust against noise than KNN, not all the filters are better than avoiding filtering noise (*Original*). When the filters remove too many instances, both noisy and clean, the decision tree is more affected since it is able to withstand small amounts of noise while exploiting the clean instances. KNN was very affected by the noisy instances left, in a higher degree than the decision tree. Thus, a wrong filtering strategy will penalize the performance of the decision tree. We will elaborate more on this later.

Table 5: Decision tree test accuracy. The highest accuracy value per dataset and noise level is stressed in bold

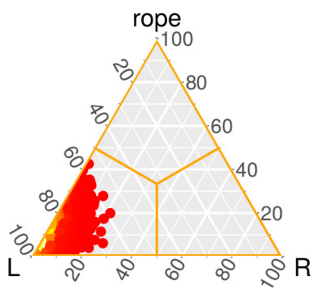
Dataset	Noise (%)	Original	HME-BD		HTE-BD				ENN-BD
P			4	5	4	4	5	5	
Vote					Majority	Consensus	Majority	Consensus	
SUSY	0	80.24	79.78	79.79	79.69	80.27	79.17	**80.29**	78.56
	5	79.94	79.99	79.97	80.07	**80.36**	80.10	80.34	77.49
	10	79.15	79.85	79.84	79.81	80.04	79.81	**80.22**	77.00
	15	78.21	**79.81**	79.80	79.32	79.47	79.61	79.48	75.81
	20	77.09	79.71	**79.73**	79.35	78.95	79.31	79.41	74.21
HIGGS	0	70.17	71.16	**71.17**	69.61	70.41	69.68	70.33	68.85
	5	69.61	**71.14**	71.11	69.34	69.98	69.36	69.92	68.29
	10	69.22	**71.06**	71.04	68.95	69.56	68.97	69.58	67.52
	15	68.65	**71.03**	70.99	68.52	69.04	68.65	69.06	66.93
	20	67.82	**71.05**	71.02	68.18	68.38	68.35	68.39	66.05
Epsilon	0	62.39	**66.86**	66.19	65.13	66.07	65.11	66.02	61.54
	5	61.10	66.64	**66.83**	65.32	66.09	65.33	66.09	60.41
	10	60.09	66.87	**67.00**	65.46	66.11	65.47	66.10	59.20
	15	59.02	66.62	**66.85**	65.33	65.99	65.29	66.00	58.09
	20	57.73	66.46	**66.79**	65.08	65.69	64.98	65.65	56.71
ECBDL14	0	73.98	74.59	74.38	74.21	74.51	74.35	**74.62**	73.66
	5	72.87	74.64	74.40	74.16	74.54	74.25	**74.75**	73.48
	10	71.67	74.59	74.25	73.84	74.51	73.94	**74.63**	72.75
	15	70.28	**74.61**	74.22	73.82	73.91	73.98	74.10	71.68
	20	68.66	**74.83**	74.18	73.78	73.82	73.85	73.86	70.16

- In terms of the best filtering technique for the decision tree, for low levels of noise, the heterogeneous ensemble HTE-BD can perform slightly better than the homogeneous HME-BD for some datasets. Nevertheless, from a 10% noise level onwards, HME-BD outperforms HTE-BD, making it a better approach to deal with noise for the decision tree.
- As observed previously, the *Original* accuracy drops for each increment of noise level. In this case, HME-BD is performing even better than observed with KNN since accuracy value is almost the same at 0% and 20% level of noise.
- Regarding the HTE-BD voting strategy, the consensus scheme achieves better results than the majority voting strategy. Please note that the opposite has been observed in KNN: since KNN is much more sensitive and demands cleaner class borders achieved with the majority voting, the decision tree benefits from a more accurate noise removal provided by the consensus voting.
- The baseline method, ENN-BD, is achieving around 1% less accuracy than the rest for low levels of noise, but this difference increases to 5% less accuracy in higher noise levels.

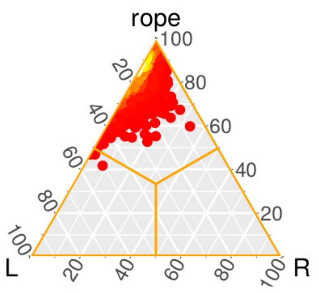
The results presented have shown the importance of applying a noise treatment strategy, no matter how much noise is present in the dataset. For a deeper analysis of the results, we have performed a Bayesian Test in order to analyze if one of the proposed algorithms is statistically better than the rest. Bayesian Tests obtain a distribution of the difference between two algorithms, and make a decision when 95% of the distribution is in one of the three regions: left, rope (region of practical equivalence), and right [48]. The Bayesian Sign Test is a Bayesian version of non-parametric sign test that uses the Dirichlet Process. This test is applied to the mean accuracy of each dataset. A sample of the distribution of the probabilities is obtained. Each point is a triplet with the probabilities of the difference between two algorithms belonging to the left, rope or right regions. For HME-BD and HTE-BD, the best performing configuration has been selected accordingly to the Friedman Test. HME-BD is used with 4 partitions while HTE-BD uses 5 partitions for both KNN and the decision tree. HTE-BD uses *majority* voting in KNN, and *consensus* voting with the decision tree.



HME-BD (L) vs *Original* (R)



HTE-BD (L) vs *Original* (R)

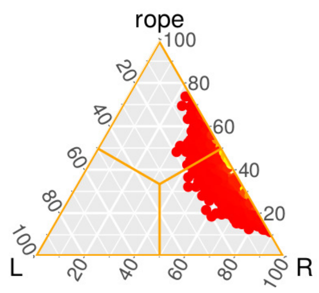


ENN-BD (L) vs *Original* (R)

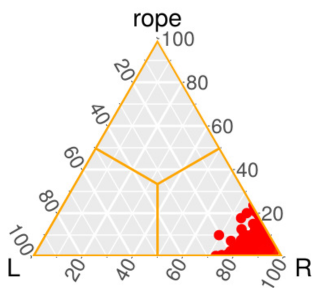
Figure 3: Bayesian Sign Test heatmap for HME-BD, HTE-BD and ENN-BD against the *Original* accuracy for KNN

In Figure 3 we compare HME-BD, HTE-BD and ENN-BD against the *Original* accuracy using KNN as classifier. As we can observe, the probability of the difference being to the right is minimal for HME-BD and HTE-BD. This means that the Bayesian Sign Test is assigning a probability of 0 to these methods performing worse than the *Original* accuracy. In ENN-BD we can see that the performance is very similar to the *Original* accuracy. For the Bayesian Tests and graphics we have employed an R package that contains a set of non-parametric and Bayesian Tests, namely rNPBST [49].

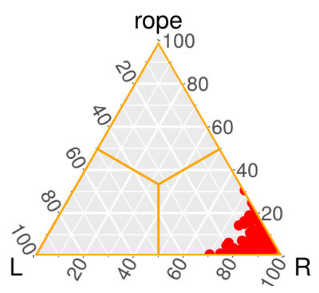
In Figure 4 we compare the proposed algorithms against each other using KNN as classifier. As we can see HME-BD is statistically better than HTE-BD according to the Bayesian Sign Test. Both HME-BD and HTE-BD performs much better than ENN-BD.



HTE-BD (L) vs HME-BD (R)



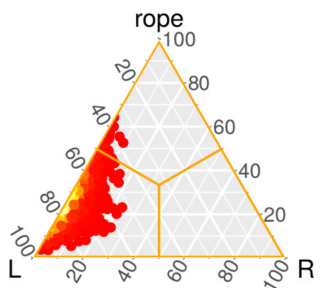
ENN-BD (L) vs HME-BD (R)



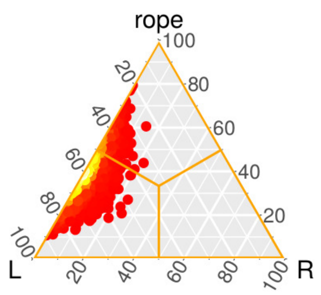
ENN-BD (L) vs HTE-BD (R)

Figure 4: Bayesian Sign Test heatmap for HME-BD, HTE-BD and ENN-BD for KNN

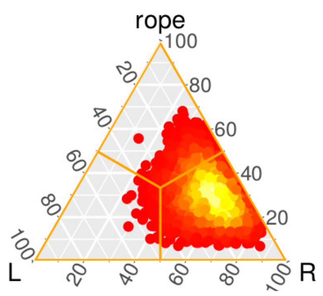
We have performed the same experimentation with the decision tree using the Bayesian Sign Test. These experiments can be seen in Figure 5 and Figure 6. These results show that HME-BD is still the best performing method according to the Bayesian Sign Test also for the decision tree, performing better than HTE-BD, ENN-BD and the *Original* accuracy. As expected, ENN-BD is not improving against the *Original* accuracy with a decision tree.



HME-BD (L) vs *Original* (R)

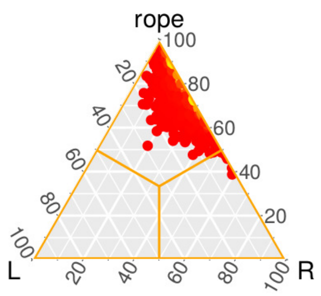


HTE-BD (L) vs *Original* (R)

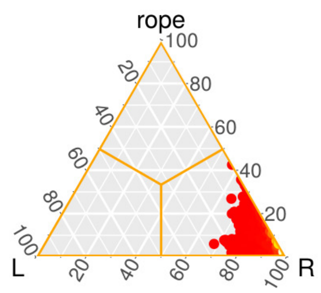


ENN-BD (L) vs *Original* (R)

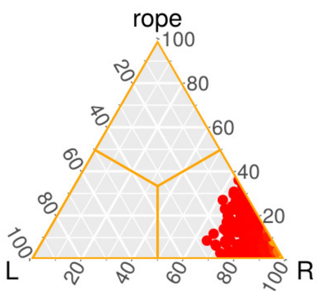
Figure 5: Bayesian Sign Test heatmap for HME-BD, HTE-BD and ENN-BD against the *Original* accuracy for a decision tree



HTE-BD (L) vs HME-BD (R)



ENN-BD (L) vs HME-BD (R)



ENN-BD (L) vs HTE-BD (R)

Figure 6: Bayesian Sign Test heatmap for HME-BD, HTE-BD and ENN-BD for a decision tree

To better explain why HME-BD is the best filtering strategy in the framework, we must study the amount of instances removed. In Table 6 we present the average number of instances left after the application of the three noise filtering methods for the four datasets. In Figure 7 we can see a graphic representation of the number of instances for the sake of a better depiction. As we can expect, the higher the percentage of noise, the lower the number of instances that remain in the dataset after applying the filtering technique. However, there are different patterns depending on the filtering technique used:

- For the homogeneous ensemble HME-BD, there is no effect in the number of partitions P chosen with respect to the amount of removed instances. On average, HME-BD removes around 20% of the instances at a 0% noise level. At each noise level increment an average of 3% of the instances are removed.
- For the Epsilon dataset, at 20% nosie, HME-BD does not remove as many instances as expected, but it is still the best option out of the two classifiers. A high instance redundancy in this dataset may cause homogeneous voting to not discard as many instances as the other filters.
- Like HME-BD, HTE-BD is not affected by the number of partitions, but the voting scheme does have a great impact on its behavior. While the majority voting strategy achieves almost the same number of removed instances as HME-BD, the consensus voting strategy is more conservative. Consensus voting removes 10% of the instances for 0% level of noise, and it is increasing a 3% on average as the level of noise increases, the same rate as HME-BD.

Table 6: Average number of instances for HME-BD, HTE-BD and ENN-BD

Dataset	Noise	Original	HME-BD		HTE-BD				ENN-BD
P			4	5	4	4	5	5	
Vote					Majority	Consensus	Majority	Consensus	
SUSY	0%	2,500,000	1,984,396	1,983,785	1,974,018	2,281,521	1,973,587	2,280,941	1,262,317
	5%	2,500,000	1,910,750	1,911,317	1,872,868	2,241,766	1,874,053	2,242,598	1,260,781
	10%	2,500,000	1,837,604	1,837,408	1,801,616	2,207,999	1,800,276	2,203,012	1,258,441
	15%	2,500,000	1,763,890	1,764,176	1,728,789	2,174,051	1,727,949	2,175,876	1,256,611
	20%	2,500,000	1,691,290	1,691,506	1,657,323	2,144,595	1,657,035	2,141,811	1,254,441
HIGGS	0%	5,500,000	3,900,547	3,900,035	3,567,784	5,048,874	3,564,879	5,051,498	2,765,831
	5%	5,500,000	3,787,000	3,786,366	3,484,271	5,014,344	3,484,274	5,013,132	2,763,942
	10%	5,500,000	3,672,429	3,672,553	3,404,181	4,972,401	3,401,624	4,973,794	2,760,547
	15%	5,500,000	3,554,120	3,557,252	3,324,547	4,930,575	3,323,465	4,932,060	2,754,636
	20%	5,500,000	3,446,352	3,443,459	3,242,174	4,888,991	3,240,623	4,886,961	2,756,382
Epsilon	0%	250,000	164,222	164,292	194,252	242,757	194,037	242,730	125,072
	5%	250,000	186,707	186,839	186,890	239,200	186,957	239,200	124,983
	10%	250,000	180,489	180,517	180,296	235,425	180,332	235,456	125,064
	15%	250,000	173,027	173,114	173,226	231,962	173,274	231,997	124,980
	20%	250,000	166,191	166,247	166,394	228,153	166,285	228,394	124,583
ECBDL14	0%	500,000	387,815	387,873	393,242	470,731	393,273	470,924	367,101
	5%	500,000	370,991	371,094	377,451	458,758	377,239	459,212	344,717
	10%	500,000	357,565	357,270	361,587	448,460	361,614	448,550	324,674
	15%	500,000	344,363	344,427	346,454	439,633	346,633	439,028	306,832
	20%	500,000	330,694	330,761	331,552	430,444	331,511	430,357	292,000

- ENN-BD is the filter that removes more instances. On average it removes half the instances of the datasets for 0% level of noise and then increases around 1% at each increment of noise level. This aggresive filtering hinders the performance of noise tolerant classifiers, such as the decision tree.

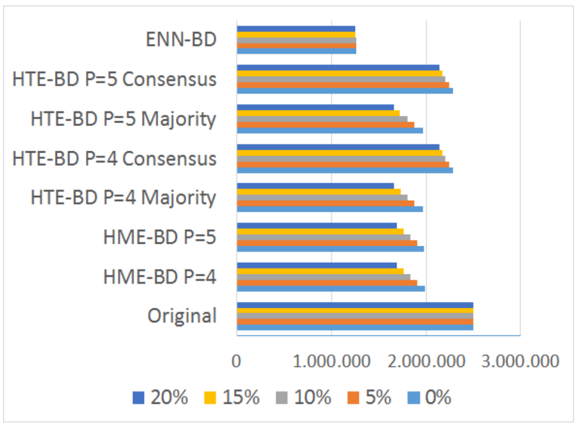
- In general, HME-BD is the most balanced technique in terms of instances removed and kept. Although the amount of instances removed by HTE-BD with majority voting is very similar to HME-BD, the instances selected to be eliminated are different, severely affecting the classifier used afterwards.

We have performed a deeper analysis of the removed instances, analyzing the amount of correctly removed instances for each method in the framework.

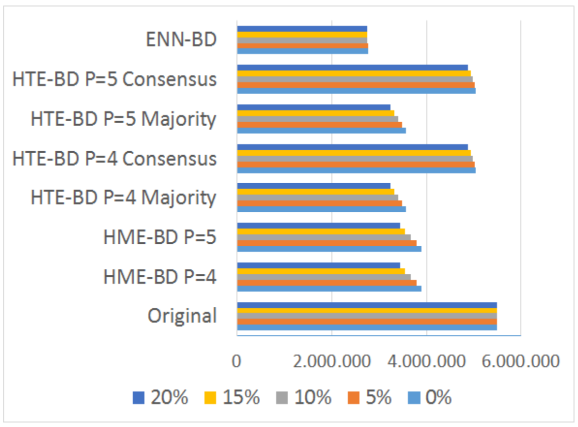
In Table 7 we present the average percentage of correctly removed instances after the application of the three noise filtering methods for the four datasets. In Figure 8 we can see a graphic representation of these percentages of correctly removed instances. As we can see, the consensus voting strategy is much more conservative removing noisy instances than the rest of the methods. We can also outline some patterns depending on the filtering method used:

- While ENN-BD is the filter that more instances removes, it is also the one that less noise removes from the datasets, averaging a 50% of noisy instances removed.

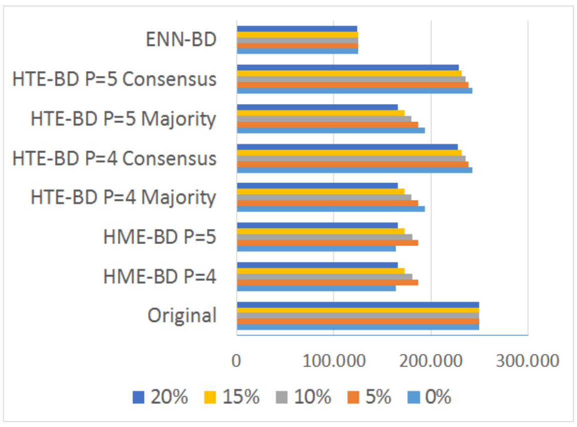
- Similarly to the number of instances removed, HME-BD and HTE-BD are not affected by the number of partitions, while the voting strategy does influence the percentage of correctly removed instances. As we could expect, the consensus voting strategy is the one that less noisy instances clean.



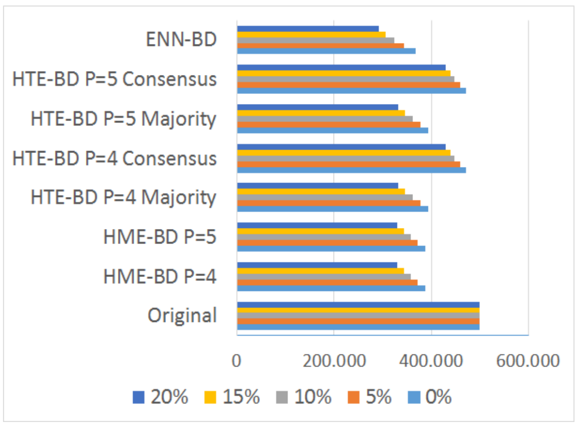
(a) SUSY



(b) HIGGS



(c) Epsilon



(d) ECBDL14

Figure 7: Number of instances after the filtering process

Consensus voting removes only 25% of noisy instances in HIGGS dataset, and only increases to 45% in ECBDL14 dataset.

- HME-BD and HTE-BD with majority voting, are removing aroung 65% and 80% of noisy instances. Both methods outperform the other in two out of four datasets.
- In Epsilon dataset, HTE-BD is cleaning 10% more noisy instances than HME-BD, but HME-BD performs better in test accuracy. This can be explained by the accumulated noise of this particular dataset.
- As we can expect, the higher the accuracy of the classifier used by the filters, the better the detection of noisy instances. That explains the different behaviors of the three noise filters, and why ENN-BD is not the method that most noisy instances removes.

In view of the results, we can conclude that HME-BD is the most suitable ensemble option in the proposed framework to deal with noise in Big Data problems. Even when we did not introduce any additional noise, the usage of noise treatment methods has proven to be very beneficial. As previously mentioned, Big Data problems tend to accumulate noise and the proposed noise framework is a suitable tool to clean and proceed from Big to Smart Datasets.

Table 7: Average percentage of correctly removed instances for HME-BD, HTE-BD and ENN-BD

Dataset	Noise	HME-BD		HTE-BD				ENN-BD
P		4	5	4	4	5	5	
Vote				Majority	Consensus	Majority	Consensus	
SUSY	5%	79.45	79.44	78.98	38.16	79.02	38.18	50.36
	10%	76.79	76.74	77.50	38.03	77.52	39.24	50.32
	15%	76.77	76.77	76.48	37.71	76.49	37.71	50.38
	20%	79.34	79.37	78.83	37.26	78.83	37.25	50.29
HIGGS	5%	69.19	69.04	66.04	26.18	66.03	26.17	49.12
	10%	69.26	69.24	66.10	25.82	66.09	25.82	49.56
	15%	69.39	69.37	66.14	25.81	66.14	25.80	49.62
	20%	69.45	69.49	66.23	25.78	66.23	25.78	49.55
Epsilon	5%	65.18	65.05	77.18	30.67	77.08	30.64	50.13
	10%	66.02	65.98	77.65	30.31	77.60	30.27	49.80
	15%	67.19	67.15	77.60	30.74	77.57	30.70	49.98
	20%	66.74	66.51	77.71	30.89	77.68	30.86	49.77
ECBDL14	5%	74.45	74.35	78.30	44.79	78.28	45.15	70.83
	10%	74.36	74.32	77.26	46.83	77.22	46.84	68.55
	15%	74.41	74.35	77.07	44.79	77.04	44.84	66.45
	20%	74.38	74.36	77.29	43.79	77.26	43.80	64.25

Table 8: Average run times for HME-BD, HTE-BD and ENN-BD in seconds

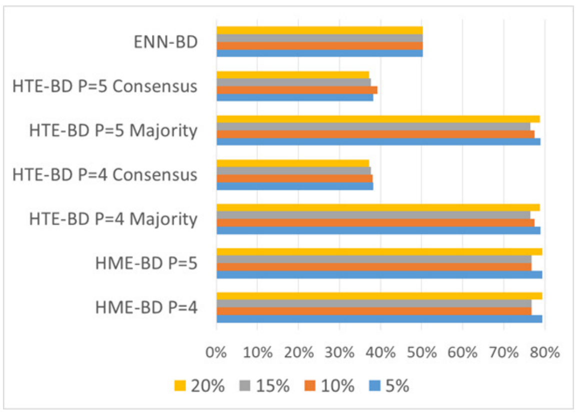
Dataset	HME-BD		HTE-BD				ENN-BD
P	4	5	4	4	5	5	
Vote			Majority	Consensus	Majority	Consensus	
SUSY	513.46	632.54	5,511.15	5,855.66	6,701.62	6,399.32	8,956.71
HIGGS	587.72	675.07	15,300.62	15,232.99	16,417.26	17,067.97	25,441.09
Epsilon	1,868.75	2,021.14	4,120.79	7,201.05	5,179.09	5,664.06	2,718.97
ECBDL14	1,228.24	1,348.10	9,710.70	11,217.02	10,798.18	11,366.01	14,080.03

4.3 Computing times

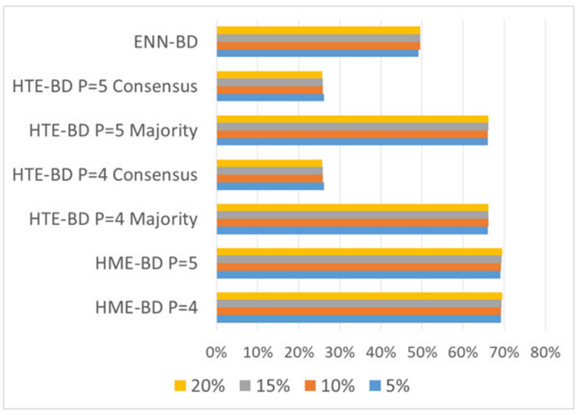
In the previous section we have shown the suitability of the proposed framework in terms of accuracy. In order to constitute a valid proposal in Big Data, this framework has to be scalable as well. This section is devoted to present the computing times for the two prospposed ensemble techniques, HME-BD and HTE-BD, and the simple similarity method, ENN-BD, used as a baseline.

In Table 8 we can see the average run times of the three methods for the four datasets in seconds. As the level of noise is not a factor that affects the run time, we show the average of the five executions performed for each dataset. In Figure 9 we can see a graphic representation of these times.

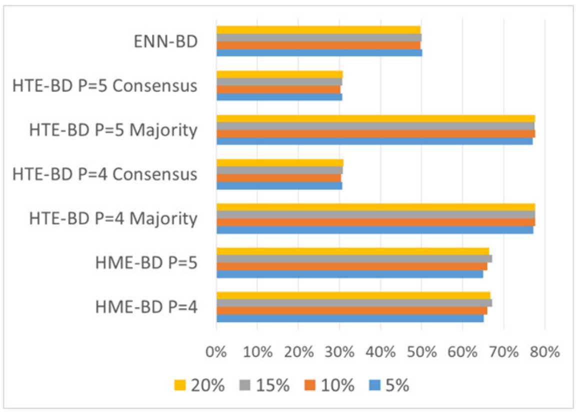
The measured times show that the homogeneous ensemble, HME-BD, is not only the best performing option in terms of accuracy, but also the most efficient one in terms of computing time. Although the accuracy value is not affected by the number of partitions, the run times of the algorithms suffer an increase in time with the number of partitions. HME-BD is about ten times faster than the heterogeneous filter HTE-BD and the



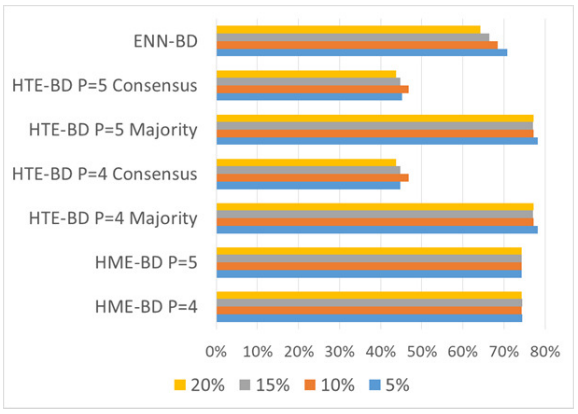
(a) SUSY



(b) HIGGS



(c) Epsilon



(d) ECBDL14

Figure 8: Percentage of correctly removed noisy instances after the filtering process

similarity filter ENN-BD. This is caused by the usage of the KNN classifier by HTE-BD and ENN-BD, which is very demanding in computing terms. As a result, HME-BD does not need to compute any distance measures, saving computing time and being the most recommended option to deal with noise in Big Data problems.

In view of the results we can conclude that:

- The usage of any of the noise treatment techniques in the framework always improves the *Original* accuracy value at the same noise level.
- HME-BD has shown to be the best performing method overall for both classifiers, KNN and the decision tree. It is also the most efficient method in terms of computing time.
- The number of partitions has little impact in the accuracy and almost no impact in the number of removed instances.
- The voting strategy has a huge impact in the number of removed instances.
- As we could expect, KNN is a very demanding method in computing terms. This is reflected in the longer computing time of HTE-BD and ENN-BD.

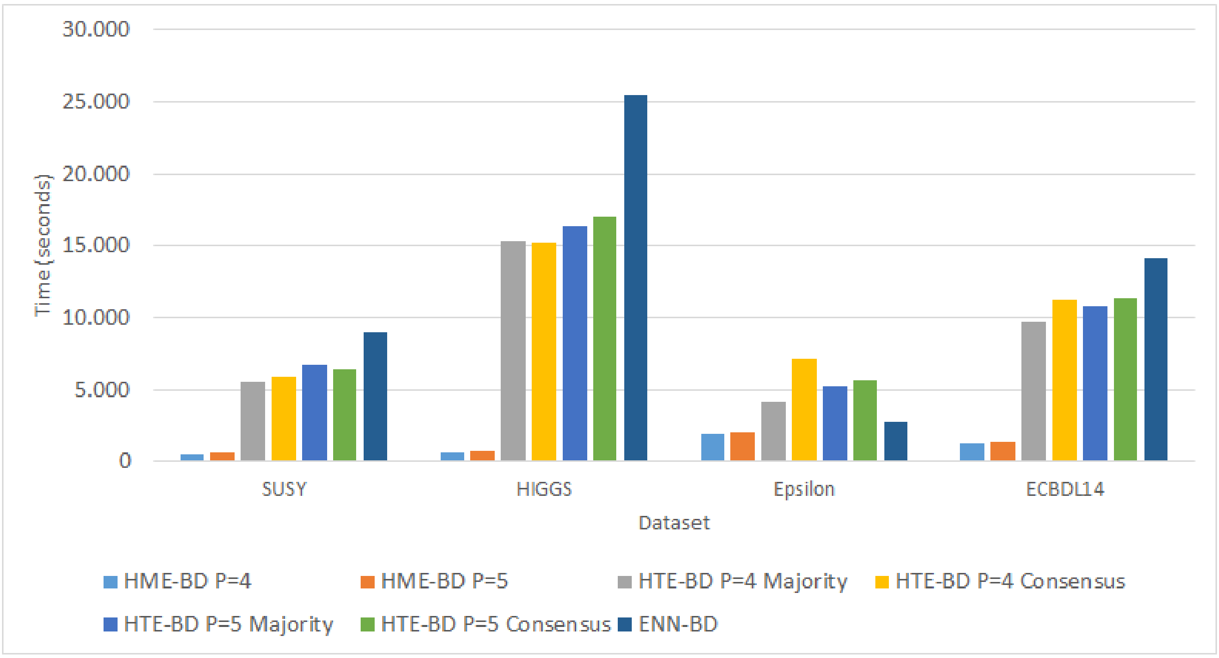


Figure 9: Run times chart

5 Conclusions

This work presents the first suitable noise filter in Big Data domains, where the high redundancy of the instances and high dimensional problems pose new challenges to classic noise preprocessing algorithms. We have proposed several noise filtering algorithms, implemented in a Big Data framework: Spark. These filtering techniques are based on the creation of ensembles of classifiers that are executed in the different maps, enabling the practitioner to deal with huge datasets. Different strategies of data partitioning and ensemble classifier combination have led to three different approaches: an homogeneous ensemble, an heterogeneous ensemble and a simple filtering approach based on similarities between instances.

The suitability of these proposed techniques has been analyzed using several data sets, in order to study the accuracy improvement, running times and data reduction rates. The homogeneous ensemble has shown to be the most suitable approach in most cases, both in accuracy improvement and better running times. It also shows the best balance between removing and keeping sufficient instances, being among the most balanced filter in terms of preprocessed training sets.

The problem of noise in Big Data classification is a crucial step in transforming such raw data into Smart Data [30]. We have tackled this problem and enabled the practitioner to reach Smart Data. Our proposal can deal with problems with millions of instances and thousands of features in a short time, obtaining clean datasets of noise.

This proposal opens promising research lines in this topic, where the presence of iterative algorithms and the usage of noise measures are also known as viable alternatives for dealing with noise. Another research line is the study of whether removing or relabelling of noisy instances may be a better strategy. Finally, for multiclass or imbalanced problems, cost sensitive filters which prioritize the removal of instances from the majority classes can be an interesting topic [50].

Acknowledgment

This work is supported by the Spanish National Research Project TIN2017-89517-P, and the Project BigDaP-TOOLS - Ayudas Fundación BBVA a Equipos de Investigación Científica 2016.

References

[1] Xindong Wu, Xingquan Zhu, Gong-Qing Wu, and Wei Ding. Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):97–107, 2014.

[2] C.L. Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*, 275:314 – 347, 2014.

[3] Salvador García, Julián Luengo, and Francisco Herrera. *Data Preprocessing in Data Mining*. Springer, 2015.

[4] Salvador García, Sergio Ramírez-Gallego, Julián Luengo, José Manuel Benítez, and Francisco Herrera. Big data preprocessing: methods and prospects. *Big Data Analytics*, 1(9):(2016).

[5] Diego García-Gil, Sergio Ramírez-Gallego, Salvador García, and Francisco Herrera. Principal Components Analysis Random Discretization Ensemble for Big Data. *Knowledge-Based Systems*, 150:166 – 174, 2018.

[6] Xindong Wu and Xingquan Zhu. Mining with noise knowledge: Error-aware data mining. *IEEE Transactions on Systems, Man, and Cybernetics*, 38:917–932, 2008.

[7] José A. Sáez, Mikel Galar, Julián Luengo, and Francisco Herrera. INFFC: An iterative class noise filter based on the fusion of classifiers with noise sensitivity control. *Information Fusion*, 27:19 – 32, 2016.

[8] Xingquan Zhu and Xindong Wu. Class Noise vs. Attribute Noise: A Quantitative Study. *Artificial Intelligence Review*, 22:177–210, 2004.

[9] Benoît Frénay and Michel Verleysen. Classification in the presence of label noise: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869, 2014.

[10] Btissam Zerhari. Class noise elimination approach for large datasets based on a combination of classifiers. In *Cloud Computing Technologies and Applications (CloudTech), 2016 2nd International Conference on*, pages 125–130. IEEE, 2016.

[11] M. Hamstra, H. Karau, M. Zaharia, A. Konwinski, and P. Wendell. *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, 2015.

[12] J. Fan, F. Han, and H. Liu. Challenges of big data analysis. *National Science Review*, 1(2):293–314, 2014.

[13] Xindong Wu. *Knowledge acquisition from databases*. Ablex Publishing Corp., Norwood, NJ, USA, 1996.

[14] Bing Liu. *Sentiment analysis: Mining opinions, sentiments, and emotions*. Cambridge University Press, 2015.

[15] Yunlei Li, Lodewyk F.A. Wessels, Dick de Ridder, and Marcel J.T. Reinders. Classification in the presence of class noise using a probabilistic Kernel Fisher method. *Pattern Recognition*, 40(12):3349–3357, 2007.

[16] Q. Miao, Y. Cao, G. Xia, M. Gong, J. Liu, and J. Song. Rboost: Label noise-robust boosting algorithm based on a nonconvex loss function and the numerically stable base learners. *IEEE Transactions on Neural Networks and Learning Systems*, 27(11):2216–2228, 2016.

[17] Charles Bouveyron and Stéphane Girard. Robust supervised classification with mixture models: Learning from data with uncertain labels. *Pattern Recognition*, 42(11):2649–2658, 2009.

[18] Carla E. Brodley and Mark A. Friedl. Identifying Mislabeled Training Data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

[19] S. Verbaeten and A.V. Assche. Ensemble methods for noise elimination in classification problems. In *4th International Workshop on Multiple Classifier Systems*, volume 2709 of *Lecture Notes on Computer Science*, pages 317–325. Springer, 2003.

[20] Taghi M. Khoshgoftaar and Pierre Rebours. Improving software quality prediction by noise filtering techniques. *Journal of Computer Science and Technology*, 22:387–396, 2007.

[21] Hai Wang, Zeshui Xu, Hamido Fujita, and Shousheng Liu. Towards felicitous decision making: An overview on challenges and trends of Big Data. *Information Sciences*, 367:747 – 765, 2016.

[22] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, January 2008.

[23] Sergio Ramírez-Gallego, Alberto Fernández, Salvador García, Min Chen, and Francisco Herrera. Big Data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce. *Information Fusion*, 42:51–61, 2018.

[24] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[25] Jimmy Lin. Mapreduce is good enough? if all you have is a hammer, throw away everything that's not a nail! *Big Data*, 1(1):28–37, 2013.

[26] Alberto Fernández, Sara del Río, Victoria López, Abdullah Bawakid, María J. del Jesús, José M. Benítez, and Francisco Herrera. Big data with cloud computing: an insight on the computing environment, mapreduce, and programming frameworks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(5):380–409, 2014.

[27] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedins of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, San Jose, CA, 2012. USENIX.

[28] Apache Flink Project. Apache Flink. `http://flink.apache.org/`, 2017.

[29] Diego García-Gil, Sergio Ramírez-Gallego, Salvador García, and Francisco Herrera. A comparison on scalability for batch big data processing on Apache Spark and Apache Flink. *Big Data Analytics*, 2(1):(2017).

[30] A. Lenk, L. Bonorden, A. Hellmanns, N. Roedder, and S. Jaehnichen. Towards a taxonomy of standards in smart data. In *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, pages 1749–1754, 2015.

[31] J. Chen, D. Dosyn, V. Lytvyn, and A. Sachenko. Smart data integration by goal driven ontology learning. In *Advances in Intelligent Systems and Computing*, volume 529, pages 283–292, 2017.

[32] P.V. Raja, E. Sivasankar, and R. Pitchiah. Framework for smart health: Toward connected data from big data. *Advances in Intelligent Systems and Computing*, 343:423–433, 2015.

[33] Jianqing Fan and Yingying Fan. High dimensional classification using features annealed independence rules. *Annals of statistics*, 36(6):2605–2637, 2008.

[34] Salvador García, Julián Luengo, and Francisco Herrera. Tutorial on practical tips of the most influential data preprocessing algorithms in data mining. *Knowledge-Based Systems*, 98:1–29, 2016.

[35] F. Iafrate. A journey from big data to smart data. *Advances in Intelligent Systems and Computing*, 261:25–33, 2014.

[36] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. Mllib: Machine learning in apache spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016.

[37] Mingkui Tan, Ivor W. Tsang, and Li Wang. Towards ultrahigh dimensional feature selection for big data. *Journal of Machine Learning Research*, 15:1371–1429, 2014.

[38] Sergio Ramírez-Gallego, Salvador García, Héctor Mouriño-Talín, David Martínez-Rego, Verónica Bolón-Canedo, Amparo Alonso-Betanzos, José Manuel Benítez, and Francisco Herrera. Data discretization: taxonomy and big data challenge. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 6(1):5–21, 2016.

[39] Isaac Triguero, Sara del Río, Victoria López, Jaume Bacardit, José M Benítez, and Francisco Herrera. Rosefw-rf: the winner algorithm for the ecbdl'14 big data competition: an extremely imbalanced big data bioinformatics problem. *Knowledge-Based Systems*, 87:69–79, 2015.

[40] Biswanath Panda, Joshua S Herbach, Sugato Basu, and Roberto J Bayardo. Planet: massively parallel learning of tree ensembles with mapreduce. *Proceedings of the VLDB Endowment*, 2(2):1426–1437, 2009.

[41] Jesus Maíllo, Sergio Ramírez, Isaac Triguero, and Francisco Herrera. kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data. *Knowledge-Based Systems*, 117:3 – 15, 2017.

[42] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, Aug 1989.

[43] Gilles Louppe. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*, 2014.

[44] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, 2(3):408–421, 1972.

[45] M. Lichman. UCI machine learning repository, 2013.

[46] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.

[47] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27:1–27:27, May 2011.

[48] Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *The Journal of Machine Learning Research*, 18(1):2653–2688, 2017.

[49] Jacinto Carrasco, Salvador García, María del Mar Rueda, and Francisco Herrera. rNPBST: An R Package Covering Non-parametric and Bayesian Statistical Tests. In Francisco Javier Martínez de Pisón, Rubén Urraca, Héctor Quintián, and Emilio Corchado, editors, *Hybrid Artificial Intelligent Systems*, pages 281–292, Cham, 2017. Springer International Publishing.

[50] Swagatam Das, Shounak Datta, and Bidyut B. Chaudhuri. Handling data irregularities in classification: Foundations, trends, and future challenges. *Pattern Recognition*, 81:674 – 693, 2018.

4 From Big to Smart Data: Iterative Ensemble Filter for Noise Filtering in Big Data classification

- D. García-Gil, F. Luque-Sánchez, J. Luengo, S. García, F. Herrera. International Journal of Intelligent Systems 34 (12), 3260-3274 (2019).
  - Status: **Published**.
  - Impact Factor (JCR 2018): **7.229**
  - Subject Category: **Computer Science, Artificial Intelligence**
  - Rank: **8/134**
  - Quartile: **Q1**

From Big to Smart Data: Iterative Ensemble Filter for Noise Filtering in Big Data classification

Diego García-Gil[*]
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
djgarcia@decsai.ugr.es

Francisco Luque-Sánchez
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
fluque1995@correo.ugr.es

Julián Luengo
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
julianlm@decsai.ugr.es

Salvador García
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
salvagl@decsai.ugr.es

Francisco Herrera
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
herrera@decsai.ugr.es

Abstract

The quality of the data is directly related to the quality of the models drawn from that data. For that reason, many research is devoted to improve the quality of the data and to amend errors that it may contain. One of the most common problems is the presence of noise in classification tasks, where noise refers to the incorrect labeling of training instances. This problem is very disruptive, as it changes the decision boundaries of the problem. Big Data problems pose a new challenge in terms of quality data due to the massive and unsupervised accumulation of data. This Big Data scenario also brings new problems to classic data pre-processing algorithms, as they are not prepared for working with such amounts of data, and these algorithms are key to move from Big to Smart Data. In this paper, an iterative ensemble filter for removing noisy instances in Big Data scenarios is proposed. Experiments carried out in six Big Data datasets have shown that our noise filter outperform the current state-of-the-art noise filter in Big Data domains. It has also proved to be an effective solution for transforming raw Big Data into Smart Data.

[*]Corresponding author.

***K*eywords** Big Data · Smart Data · Classification · Class Noise · Ensemble.

1 Introduction

Big Data collection has a simple purpose, to get to know people and our environment in the best possible way. We are in a moment in time where everything in the world has grown to certain limits, where companies and researchers have to go a step further and find new sources of information, new sources of knowledge which will enable us to develop and adapt further [1]. Nowadays, companies and researchers look for the information they have not been interested in so far, but as of today, this information is the only way to take this next step. All this information comes from a myriad of sources, both structured and unstructured [2]. Nevertheless, both companies and researchers are facing new challenges coping with the Volume, Velocity, Veracity and Value (among many other V's) that characterize this new paradigm [3].

The main concept of Big Data is that these huge amounts of information will enable machine learning algorithms to achieve better and more accurate models than ever before. Classical methods cannot handle this humongous new data space requirements. Recently, some software solutions have appeared for dealing with this Big Data scenario, like Apache Spark [4], based on the MapReduce paradigm [5]. Thanks to frameworks like Apache Spark, some classical data mining and machine learning methods are being adapted to deal with these amounts of data. However, due to the automation in data generation and acquisition, Big Data mining techniques must not only deal with the scalability problem (volume/velocity), but also must handle inaccurate data (noise or incomplete).

Recently, a new term related to the Big Data environment have emerged. Smart Data refers to the challenge of acquiring knowledge from raw data [6, 7]. In other words, Smart Data refers to the challenge of transforming information into knowledge. Smart Data aims to separate the raw (or Big) part of the data (volume/velocity), from the Smart part of it (veracity/value) [8]. Therefore, Smart Data is focused on extracting valuable knowledge from data, in the form of a subset, that contains enough quality for a successful data mining process [9].

Data preprocessing is strongly linked to the concept of Smart Data [10], as it is one of the most important phases of every data mining process. Raw data is likely to contain imperfections, redundancies or inconsistencies, making it unsuitable for a data mining process [11]. The goal of data preprocessing is to clean and amend errors in the data, in order to achieve better performance in the later machine learning process. However, data preprocessing methods are also affected by the increasing size and complexity of the data, making them unable to obtain a preprocessed/smart dataset in a reasonable time limit, and therefore, they need to be redesigned with Big Data technologies.

Among all imperfections that data can contain, noise is one of the most disruptive ones [12]. Noise can be defined as an external factor that affects the data by corrupting it. Noise disrupts the models obtained and decrements the performance of machine learning algorithms. Alleviating the effects of noise is a challenging task that require the correct identification of the corrupted examples in the data. The negative effects of noise will increase with the size of the data [13]. Although a Big Data scenario is very likely to contain huge amounts of noise, little research has been devoted to tackling noise in Big Data. That's why there is a special need for noise filters in Big Data. Although we can find many proposals for dealing with noise for normal-sized data in the literature, in Big Data scenarios we can find only a handful of proposals devoted to this problem [14, 8, 9].

In this paper, we propose a novel iterative ensemble noise filter for Big Data classification problems under Apache Spark, namely Iterative Ensemble Filter for Big Data (IEF-BD). This filter performs a k-fold to the input data and learns a Random Forest on each partition. Then it predicts the whole dataset and, according to a vote strategy, removes instances considered as noise. This process is repeated in a iterative fashion a number of times. IEF-BD has been implemented in Apache Spark [15], and it is available publicly as a Spark package in Spark's third party repository Spark Packages[2].

In order to show the performance of the proposed ensemble filter, we have carried out an in depth experimental evaluation on six Big Data datasets. These datasets have very different properties among them, ranging from binary to multiclass problems. They also vary from datasets with 250,000 instances to datasets with more than 11,000,000 instances. With these datasets, we can assess the performance of the proposed noise filter in very different scenarios. We have included five levels of random class noise to evaluate the effect of increasing levels of noise in the datasets. As classifier, we use Spark's MLlib implementation of a decision tree [16]. We compare our method with the most recent and best performing proposal in the literature for noise cleaning in Big Data, HME-BD [8]. We show that, for some problems, the classifier benefits from the noise filtering even when no noise is added. This shows that Big Data problems can contain implicit noise [13]. Results obtained indicate that the proposed ensemble filter is able to successfully remove noisy instances from a dataset. IEF-BD outperforms HME-BD in terms of accuracy for all tested datasets, achieving good performance even when HME-BD is not able to improve the base accuracy without a noise filtering strategy.

The remainder of this paper is organized as follows: Section 2 presents the concepts of noise and MapReduce. Section 3 explains the proposed noise ensemble filter. Section 4 describes the experiments carried out to check the performance of the noise filter. Finally, Section 5 concludes the paper.

2 Background

In this section, we introduce the problem of noise in classification tasks in Section 2.1. We conclude with a description of the MapReduce framework, the most popular solution for Big Data environments, in Section 2.2.

2.1 Noisy Data

As we have mentioned before, the presence of noise in the data involves a very negative impact in the models learned. This negative effect is aggravated if the learner is noise sensitive. Data size is a factor that increases the amount of noise in data, since noise accumulates when the number of dimensions and instances increases [13].

We can find two types of noise in the literature [17, 18], depending on which element of the dataset is affecting:

- *Class Noise*: also known as *label noise*, refers to wrongly labeled examples. Class noise is conformed by contradictory examples [19] (instances with the same input attributes, but with different classes), and misclassifications [17] (wrongly labeled examples).

- *Attribute Noise*: is referred to corruptions in the values of the features. Although it is composed of several data aberrations (missing values or "do not care" values among others), it is mainly focused on erroneous values [17].

Class noise is known to be more disruptive with the learning process. Thus, methods for dealing with this type of noise are more common in the literature [17]. Class noise can have many natures, such as errors in the data acquisition process, or subjectivity in the data labeling phase.

[2] `https://spark-packages.org/package/djgarcia/IEF_BD`

We can find different approaches to the noise problem in the literature [20, 17, 10]. These different techniques range from creating noise robust learners and algorithms, to data preprocessing methods that detect and either remove or relabel the noisy instances. Noise filtering has also been employed for automatic cluster discovering [21]. In [20], two main different approaches for the noise handling problem are distinguished:

- *Algorithm Level*: these methods aim to create robust learning algorithms that are little or no influenced by the presence of noise in the data. They include algorithms that model noise in the learning process [22], pruning strategies to avoid overfitting to noisy instances, or decreasing the importance of noisy instances with respect to clean ones [23]. Strategies that combine both approaches have also been proposed, which model the noise and diminish the importance of noise in the learning process [24].
- *Data Level*: also called *filters*, aim to remove noisy instances from the data, as a previous step to the learning process. We can find ensemble strategies [25], data partitioning approaches [26] or iterative algorithms [27].

As described previously, data size is a factor that affects greatly to the amount of noise in data. This, along with the many natures of class noise (errors in the data, subjectivity or lack of supervision), are the reasons why the presence of noise in Big Data environments is much greater than in normal-sized problems. Thus, models learned from this data will be weaker and more complex. Noise filters not only improve the quality of the data, but also obtain a reduced version of it, improving the speed and efficiency of the later data mining and machine learning processes. The solution goes by transforming from Big to Smart Data before the application of the learner.

2.2 Big Data: MapReduce and Apache Spark

MapReduce is the most popular and widely used paradigm for Big Data processing nowadays. It was originally proposed by Google in 2003 [5]. This paradigm is born from the necessity of processing and/or generating large amounts of data in a distributed and efficient fashion, while minimizing disk access and network use.

It has two main phases: the *map* phase, and the *reduce* phase. Before starting the *map* phase, the master node partitions the data and distributes it across the cluster. Then, the *map* function applies a transformation operation to the local key-value pairs of each computing node. Once the *map* phase is finished, all pairs belonging the the same key are redistributed. When all pairs belonging to the same key are in the same computing node, the *reduce* phase starts. The *reduce* phase is a summary operation that generates the final values.

Apache Hadoop is the most popular open-source implementation of MapReduce [28]. Although Hadoop is a very extended tool for Big Data processing, it as some important limitations [29]:

- It is not suitable for iterative algorithms.
- Intensive disk usage.
- Lack of in-memory computation.

Apache Spark is an open-source framework for Big Data, focused on speed, ease of use and sophisticated analytics [15]. Spark solves the Hadoop problems by in-memory computing, allowing to persist the data in memory for consecutive or iterative processing. It is built on top of a novel distributed data structure, named Resilient Distributed Datasets (RDDs) [4]. RDDs are unsorted and immutable by nature. They allow programmers to persist them in memory, and they are tracked using a lineage, so that each partition can be recomputed in case of failure. RDDs support two types of operations: transformations and actions. Transformations are applied to each partition of an RDD and produce a new RDD. They are also not evaluated until an action is computed. Actions triggers all previous transformations of an RDD, and return a value.

3 Iterative Ensemble Filter: a Big Data approach

In this section, we describe in depth our proposal for noise filtering in Big Data. IEF-BD is an iterative ensemble noise filtering algorithm based on partitions of the input data. It is based on the MapReduce paradigm, where all operations are performed in a distributed fashion. Section 3.1 describes the Spark primitives used for the implementation of IEF-BD. In Section 3.2, we explain in detail our proposed iterative noise filter, IEF-BD.

3.1 Spark Primitives

For the implementation of the algorithm we have employed some of the Spark primitives available in the Spark API. These primitives extend the MapReduce paradigm, allowing much more complex operations over the data. Here, we outline those more relevant to our proposed noise filter [3]:

- map: applies an user-defined operation to each element of an RDD. A new RDD is created after this transformation, and the resulting RDD is returned.
- $zipWithIndex$: appends an index to each element of an RDD. This index is zero-based and creates a new RDD of (value, index) tuples.
- $filter$: returns a new RDD with those instances that satisfy a predicate.
- $kFold$: returns a list of k pairs of RDDs with the first element of each pair containing the $train$ data, a complement of the $test$ data, and the second element containing the $test$ data, being a unique 1/kth of the data. Where k is the number of folds.
- $randomForest$: learns a Random Forest model on the input data. The learned model is returned for prediction.
- $predict$: returns an RDD with the predicted classes for each instance, using a previously learned model.
- $count$: returns the number of elements of an RDD.

These Spark primitives from Spark API are used in the following section, where IEF-BD algorithm is described.

3.2 Iterative Ensemble Filter: IEF-BD

The iterative ensemble filter is inspired by an ensemble-based noise filter called Iterative-Partitioning Filter (IPF) [30]. This filter removes the noisy instances in a dataset by partitioning the data into k equal parts. Then, a C4.5 decision tree is learned on each of the partitions. This k models are used to predict the k

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

Algorithm 1 IEF-BD Algorithm

```
 1: Input: data a RDD of tuples (label, features)
 2: Input: k the number of partitions
 3: Input: threshold the threshold for the vote strategy
 4: Input: maxIter the maximum number of iterations
 5: Input: nTrees the number of trees for Random Forest
 6: Output: the filtered RDD without noise
 7: $iter \leftarrow 0$
 8: $finalData \leftarrow \emptyset$
 9: do
10:     $predictions \leftarrow$
11:     map $instance \in data$
12:         $(instance, Array[k])$
13:     end map
14:     $partitions \leftarrow kFold(predictions, k)$
15:     for all $index, train$ in $partitions$ do
16:         $rfModel \leftarrow randomForest(train, nTrees)$
17:         $rfPred \leftarrow predict(rfModel, data)$
18:         $predictions \leftarrow add(index, rfPred)$
19:     end for
20:     $noisyData \leftarrow$
21:     map $instance, arrayPreds \in predictions$
22:         $errors \leftarrow count(filter(arrayPreds, predLabel \neq label(instance)))$
23:         if $errors \geq k * threshold$ then
24:             $(label = \emptyset, features(instance))$
25:         else
26:             $instance$
27:         end if
28:     end map
29:     $finalData \leftarrow filter(noisyData, label \neq \emptyset)$
30:     $iter \leftarrow iter + 1$
31: while $iter < maxIter$
32: $return(finalData)$
```

partitions. Finally, according to a voting strategy, missclassified instances are removed. The iterative nature of IPF offers a great advantage over other noise filters. By removing the noise progressively, the following iterations are more precise.

IEF-BD is also based on a partitioning scheme of the data. But there are some important differences. Each iteration, IEF-BD partitions the data performing a k-fold to the input data. Then it learns Spark's implementation of Random Forest on the $train$ partition and predicts the full dataset. This learning process is repeated k times. When this process is finished, each instance has k predictions. Using a voting strategy, the instance is either kept or removed. The filter is applied until a number of iterations are performed.

Algorithm 1 describes the filtering process of IEF-BD:

- First, the noise filter appends an array of size k to each instance using a map function. This array will contain the predicted classes for each of the k learners.

- The input data is partitioned using Spark's $kFold$ function. As stated previously, this function partitions the data into k partitions, returning a list of $(train, test)$ splits. The $train$ is formed by the union of $k-1$ partitions, and the $test$ contains the partition left.
- The next step is to learn the k Random Forest models using the $train$ as input data, with the specified number of trees ($nTrees$). Then, the whole dataset is predicted using the previously learned model. Finally, using the add function, we append to the corresponding array index, the predictions for each instance. Although add is not a pure Spark primitive, we use it to simplify the description of the algorithm.
- Once we have learned the $train$ partitions and predicted the whole dataset k times, we apply the voting strategy to decide whether to keep or remove every instance. This process is applied using a map function. For each instance, we start by keeping only the predictions that does not match the real label using the $filter$ function. Then we count the number of wrong labels using the $count$ method. If this number is greater or equal than a condition ($k * threshold$), the instance is marked as noise. Otherwise, the instance is kept. At the end of this step, all instances are marked either as noisy or correct.
- Then, instances marked as noisy are removed using the $filter$ function, and the number of iterations is updated.
- Finally, the algorithm checks if the number of iterations has not reached its maximum specified value. If the condition is satisfied, the algorithm finishes and returns the most recent cleaned dataset. In other case, the algorithm starts again, using the previously cleaned data as input data.

The following are required as input parameters: the dataset (*data*), the number of partitions (k), the threshold for the voting strategy ($threshold$), the maximum number of iterations ($maxIter$) and the number of trees for the Random Forest ($nTrees$).

In Figure 1 we show a flowchart of the IEF-BD noise filtering process.

4 Experimental Results

In this section, we show all the details related to the experimental framework, starting with the datasets selected for the experimentation, the performance metrics, and the parameters of the methods used. Additionally, we detail the hardware and software resources used to carry out the experiments.

We have selected six Big Data classification problems. These datasets are extracted from the UCI Machine Learning Repository [31], and are publicly available. In Table 1 we show the datasets selected, the number of examples (Instances), the total number of attributes (Atts.), the total number of training data (Total), and the number classes (CL).

We have conducted the experiments with five levels of uniform class noise. For each of the five noise levels, a percentage of the training instances has been altered by replacing the real class, with another randomly selected from the available classes. Selected noise levels have been: 0%, 5%, 10%, 15% and 20%. In this case, a 0% noise levels refers to the original dataset without added noise. We have conducted a 5 fold cross-validation scheme. This means that every dataset has been partitioned in five folds, with 80% of instances for training, and 20% for test.

The parameters used for both IEF-BD and HME-BD are shown in Table 2. The same seed and number of partitions is used for both IEF-BD and HME-BD. For HME-BD values recommended by the authors are

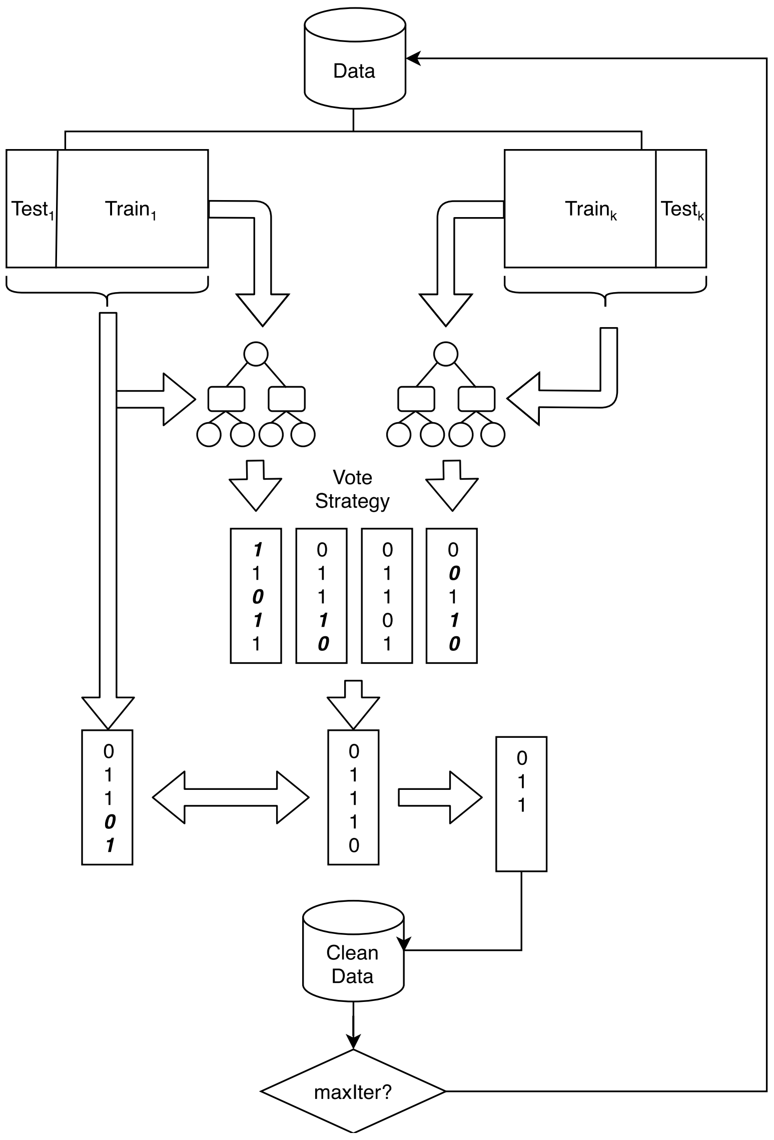


Figure 1: IEF-BD noise filtering process flowchart (instance labels are represented by 0's and 1's). It starts with a k-fold of the data. Then, each $train$ is used for learning a Random Forest, and then predicts the whole dataset. Finally, using a voting strategy, noisy instances are removed. This process is repeated a number of iterations.

used. For IEF-BD, we have increased the depth of the Random Forest, as well as the number of trees. This will lead to a more aggressive detection of noise, but combined with the voting strategy, we can keep the clean instances, while removing the noisy ones. We have selected one iteration for IEF-BD, as internal testing showed that one iteration was enough for removing the noisy instances. Several iterations removed both noisy and clean instances.

After the noise filtering process, we need to apply a classifier in order to assess the performance of the algorithms. We have selected Spark's MLlib distributed decision tree as a classifier. This algorithms is capable to learn a decision tree globally, using all data at once on the learning phase. Additionally, its depth can be adjusted for a better detection of noisy instances. Table 3 shows the parameters for the decision

Table 1: Datasets used in the analysis

Dataset	Instances	Atts.	Total	CL
skin_noskin	245,057	3	735,171	2
ht_sensor	928,991	11	10,218,901	3
watch_acc	3,540,962	20	70,819,240	7
watch_gyr	3,205,431	20	64,108,620	7
Susy	5,000,000	18	90,000,000	2
Higgs	11,000,000	28	308,000,000	2

Table 2: Parameter settings for the noise filters

Algorithm	Parameters	Classifier
IEF-BD	k = 4, maxIter = 1, nTrees = 200	Random Forest: featureSubsetStrategy = "auto", impurity = "gini", maxDepth = 12 and maxBins = 32
HME-BD	k = 4, nTrees = 100	Random Forest: featureSubsetStrategy = "auto", impurity = "gini", maxDepth = 10 and maxBins = 32

tree. As an evaluation metric for the noise filters, we have used the prediction accuracy produced by the classifier (number of examples correctly labeled as belonging to a given class, divided by the total number of examples).

Table 3: Parameter setting for the classifier

Classifier	Parameters
Decision Tree	impurity = "gini", maxDepth = 20 and maxBins = 32

The cluster used for all the experiments is composed of 14 nodes managed by a master node. All nodes have the same hardware and software configuration. Regarding the hardware, each node has 2 x Intel Xeon CPU E5-645 processors, 6 cores (12 threads) per processor, 2.40 GHz and 64 GB of RAM. The network used is Infiniband 40Gb/s. The operating system is CentOS 6.9, with Apache Spark 2.2.0.

In Table 4 we show the test accuracy for the two noise filter methods. The *Original* accuracy represents the dataset without any noise filtering applied. In view of the results, we can point out that:

- Avoiding noise treatment is not a good option. When a noise filtering algorithm is applied, we can see an important increase in accuracy. However not all noise filters are achieving the same results. This shows the importance of the right choice of the noise filtering technique.
- As it is expected, the *Original* accuracy drops with each level of noise. We can see up to a 2% drop in accuracy from the lower to the higher levels of noise.
- Regarding IEF-BD and HME-BD, both obtain very similar results in *skin_noskin* and *Susy* datasets. Although both results are similar, IEF-BD achieves slightly better results. In Higgs dataset, the difference increases to a 0.6% of accuracy in favor of IEF-BD.
- Although a filtering strategy is always recommended, not every noise filter is going to perform good for all datasets. HME-BD is not improving the accuracy in *ht_sensor* and both *watch_acc* and

Table 4: Decision tree test accuracy. The highest accuracy value per dataset and noise level is stressed in bold

Dataset	Noise (%)	Original	IEF-BD				HME-BD
Vote			0.25	0.50	0.75	1.00	
skin_noskin	0	**99.88**	99.77	99.83	99.84	99.84	99.78
	5	99.71	99.80	99.82	99.78	**99.84**	99.80
	10	99.51	99.79	**99.80**	99.78	**99.80**	99.75
	15	99.35	**99.78**	**99.78**	**99.78**	99.74	99.75
	20	99.02	**99.77**	99.72	99.74	99.64	99.72
ht_sensor	0	**99.99**	99.82	99.91	99.93	99.95	99.52
	5	99.84	99.82	99.83	99.85	**99.86**	98.84
	10	99.74	99.84	99.78	99.82	**99.90**	98.81
	15	99.57	99.81	99.82	99.86	**99.90**	99.08
	20	99.35	99.74	99.78	99.80	**99.85**	98.92
watch_acc	0	**90.96**	87.17	88.07	88.84	90.03	86.12
	5	**90.56**	86.73	88.05	88.69	89.62	85.77
	10	**90.13**	86.96	87.89	88.40	89.63	85.36
	15	89.51	86.87	87.65	88.50	**89.63**	85.39
	20	89.35	87.35	87.62	88.56	**89.63**	85.04
watch_gyr	0	**89.80**	84.67	85.89	87.33	88.83	85.51
	5	**89.30**	84.67	86.50	87.02	88.48	86.12
	10	88.79	84.63	85.63	87.23	**88.94**	85.27
	15	88.41	85.04	85.98	86.64	**88.56**	85.68
	20	87.83	84.66	85.51	86.38	**88.11**	86.21
Susy	0	77.64	**79.55**	**79.55**	79.54	79.52	79.33
	5	77.19	**79.53**	**79.53**	79.50	79.50	79.32
	10	76.71	**79.55**	79.52	79.50	79.48	79.29
	15	76.20	**79.52**	79.50	79.50	79.45	79.30
	20	75.60	**79.51**	79.50	79.50	79.43	79.32
Higgs	0	70.90	71.58	71.68	71.74	**71.81**	71.21
	5	70.44	71.57	71.61	71.68	**71.74**	71.14
	10	69.90	71.50	71.57	71.62	**71.65**	71.05
	15	69.35	71.47	71.51	71.59	**71.60**	71.09
	20	68.76	71.48	71.50	71.48	**71.52**	71.05

watch_gyr datasets. On the other hand, IEF-BD, thanks to the voting strategy, is able to improve the *Original* accuracy in the presence of noise for these datasets.

- Regarding the voting strategy in IEF-BD, we can see that it affects greatly to the results achieved. For multiclass problems, the voting strategy in IEF-BD enables it to be more conservative, and to be able to improve the *Original* accuracy. There are two datasets which benefit from the more aggressive voting (*skin_noskin* and *Susy*), and four that improve with the most conservative strategy.

Table 5: Heatmap with the average number of instances left for IEF-BD and HME-BD. Higher reduction rates are represented in blue.

Dataset	Noise (%)	Original	IEF-BD				HME-BD
Vote			0.25	0.50	0.75	1.00	
skin_noskin	0	196,044	195,397	195,667	195,778	195,842	195,217
	5	196,044	185,625	185,872	186,021	186,142	185,547
	10	196,044	175,825	176,160	176,295	176,452	175,759
	15	196,044	166,050	166,398	166,555	166,780	166,083
	20	196,044	156,206	156,648	156,857	157,177	156,368
ht_sensor	0	743,080	740,620	741,606	741,928	742,355	691,183
	5	743,080	703,072	704,396	704,696	705,130	663,353
	10	743,080	666,718	667,363	667,804	668,306	628,374
	15	743,080	629,649	630,389	630,791	631,265	592,795
	20	743,080	592,365	593,165	593,660	594,003	568,899
watch_acc	0	2,833,506	2,346,432	2,462,645	2,549,083	2,647,584	2,245,381
	5	2,833,506	2,209,657	2,337,209	2,424,937	2,515,130	2,135,871
	10	2,833,506	2,095,469	2,223,382	2,307,140	2,394,380	2,020,678
	15	2,833,506	1,975,212	2,096,939	2,186,882	2,282,026	1,928,061
	20	2,833,506	1,871,135	1,969,115	2,064,194	2,138,561	1,812,435
watch_gyr	0	2,564,234	2,043,638	2,168,677	2,260,954	2,359,461	2,059,045
	5	2,564,234	1,922,235	2,072,485	2,165,877	2,258,690	1,962,049
	10	2,564,234	1,844,276	1,966,590	2,057,306	2,138,387	1,877,071
	15	2,564,234	1,744,699	1,863,388	1,932,234	2,020,472	1,760,277
	20	2,564,234	1,646,141	1,747,086	1,816,750	1,899,865	1,660,317
Susy	0	3,999,282	3,149,713	3,184,854	3,214,815	3,247,301	3,173,871
	5	3,999,282	3,032,874	3,067,724	3,095,382	3,127,733	3,055,178
	10	3,999,282	2,907,968	2,947,894	2,979,413	3,016,833	2,937,199
	15	3,999,282	2,788,096	2,828,909	2,861,443	2,900,244	2,823,115
	20	3,999,282	2,674,538	2,713,045	2,742,538	2,778,414	2,706,100
Higgs	0	8,799,713	6,163,632	6,310,333	6,423,391	6,563,742	6,232,771
	5	8,799,713	5,973,177	6,114,038	6,234,391	6,368,410	6,053,240
	10	8,799,713	5,763,332	5,919,740	6,044,891	6,193,495	5,865,209
	15	8,799,713	5,577,546	5,731,428	5,850,257	5,996,852	5,690,875
	20	8,799,713	5,321,615	5,494,102	5,692,952	5,854,863	5,509,134

In Table 5 we gather the total number of instances left after the noise filtering process for the five levels of noise. In Figure 2 we can see a graphic representation of the reduction rate after the noise filtering process for the sake of a better depiction. In view of these results, we can draw the following conclusions:

- As it is expected, the number of instances removed increases with the increasing level of noise. This shows that the noise filters are detecting the noisy instances properly.
- The voting strategy in IEF-BD has a great impact in the number of instances removed. Thanks to this mechanism, IEF-BD is able to maintain the clean instances and to remove the noise effectively.

For datasets with good base accuracy, IEF-BD is keeping almost all instances in the data for low levels of noise. Also the voting strategy has a crucial role in multiclass datasets, maintaining the necessary number of instances for each level of noise.

- HME-BD is more aggressive removing instances than IEF-BD. Altough both remove a similar number of instances if we attend to the most aggressive voting strategy in IEF-BD.
- In *Susy* and *Higgs* datasets, both filters are removing an important number of instances at 0% of noise. This means that these datasets have intrinsic noise.
- As we can see in Figure 2, for datasets with good base accuracy (*skin_noskin* and *ht_sensor*), IEF-BD is removing almost the exact percentage of added noise.
- If we compare the number of instances removed, with the accuracy achieved by IEF-BD, we can conclude that there is not a better voting strategy in IEF-BD overal. There are datasets that benefit from the most conservative strategies, while others achieve better results with the higher percentage of instances removed. This shows the importance of the right choose of the voting strategy.

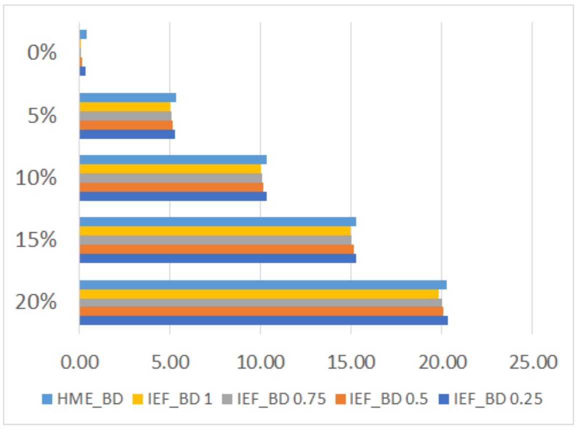
In order to constitute a proper Big Data proposal, we shall analyze the computing times for both IEF-BD and HME-BD. In Table 6 we show the runtimes for the noise filtering methods. As the level of noise is not a factor that affects the runtimes, we show the average runtimes for the five folds and the five levels of noise. As it is expected, IEF-BD is more computationally expensive than HME-BD. IEF-BD learns deeper Random Forests and with two times the number of trees compared to HME-BD. It also predicts the whole dataset, instead of just the $test$ partition. Even so, IEF-BD achieves very competitive computing times, being able to effectively remove the noise in Big Data datasets in a reasonable amount of time.

Table 6: Average runtimes for IEF-BD and HME-BD in seconds

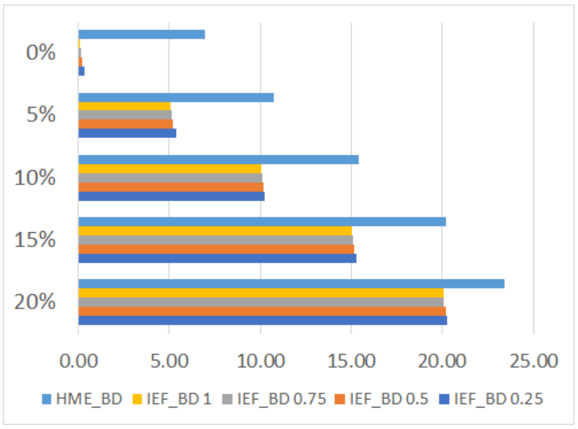
Dataset	IEF-BD	HME-BD
skin_noskin	562	175
ht_sensor	1,787	433
watch_acc	2,315	782
watch_gyr	2,083	625
Susy	4,157	2,333
Higgs	8,962	4,004

In view of these results, we can conclude that:

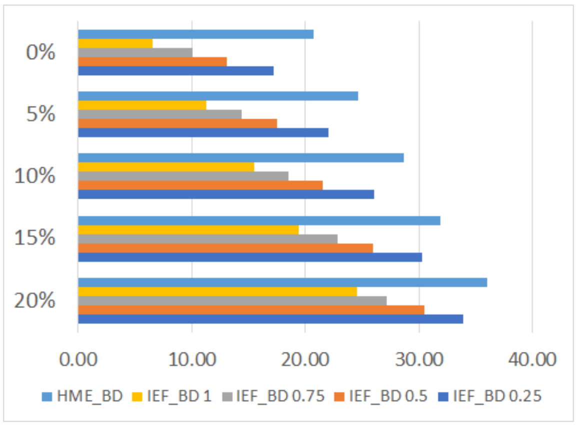
- The usage of a noise filtering technique can improve the *Original* accuracy even when no noise is added.
- IEF-BD has shown to perform better than the current state-of-the-art noise filter for Big Data HME-BD.
- The vote strategy in IEF-BD has a crucial role in the noise filtering performance.
- The computing times of IEF-BD are higher than HME-BD as expected, but they have shown to be reasonable for the size of the datasets employed.



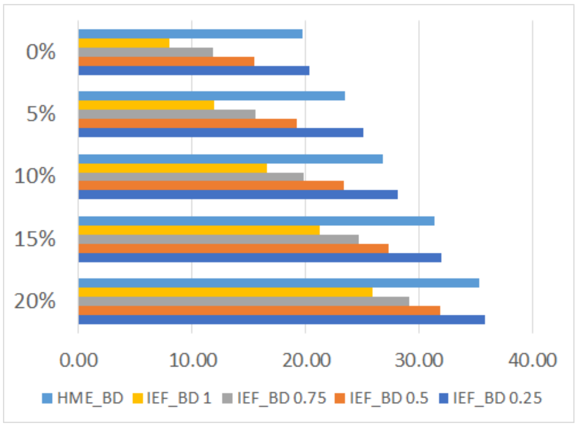
(a) skin_noskin



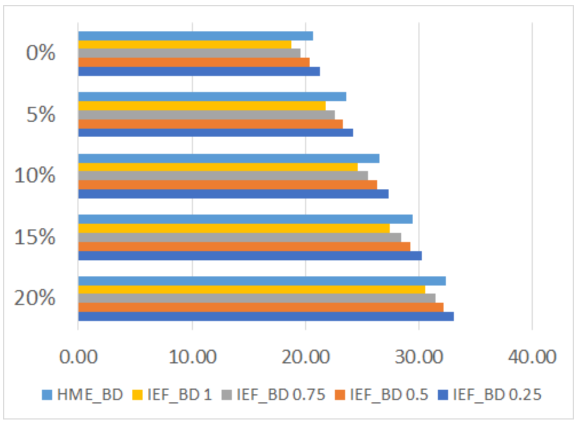
(b) ht_sensor



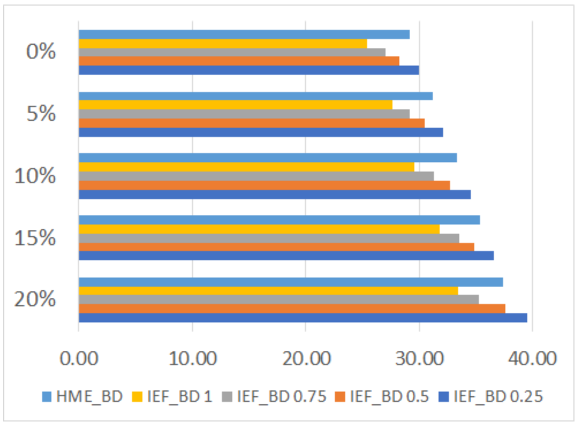
(c) watch_acc



(d) watch_gyr



(e) Susy



(f) Higgs

Figure 2: Reduction Rate (%) after the noise filtering process

5 Conclusions

In this work, we have proposed an iterative ensemble noise filter for noise cleaning in Big Data classification, IEF-BD. Big Data scenarios have a high density of data points and pose a challenge to classic noise preprocessing algorithms. Our proposed noise filter cleans the data in an iterative fashion, using a data partitioning strategy. It is implemented in a distributed manner for the Big Data framework Apache Spark.

Experimental results have shown that the proposed technique is able to detect and remove the noisy instances from the six tested Big Data datasets. Our method is able to perform better than the current state-of-the-art noise filter for Big Data for every tested dataset, HME-BD. It has also shown to be effective even when HME-BD is not able to improve the original accuracy without any noise filtering strategy applied. Different vote strategies have led to very different results, showing that the right choice of the vote strategy is crucial. Regarding computing times, IEF-BD has handled Big Data datasets in a reasonable amount of time for the size of the tested datasets.

The problem of noise is a crucial step in transforming Big Data into Smart Data, especially in Big Data scenarios. With this proposal, we have enabled the practitioner to reach Smart Data from raw and low quality Big Data [32]. Our noise filter is able to deal with Big Data problems in a short time, achieving a noise clean version of the dataset.

Acknowledgment

This work is supported by the Spanish National Research Project TIN2017-89517-P.

References

[1] Hsinchun Chen, Roger H. L. Chiang, and Veda C. Storey. Business intelligence and analytics: From big data to big impact. *MIS Q.*, 36(4):1165–1188, December 2012.

[2] Sergio Ramírez-Gallego, Alberto Fernández, Salvador García, Min Chen, and Francisco Herrera. Big Data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce. *Information Fusion*, 42:51–61, 2018.

[3] Alberto Fernández, Sara del Río, Victoria López, Abdullah Bawakid, María J del Jesus, José M Benítez, and Francisco Herrera. Big data with cloud computing: an insight on the computing environment, mapreduce, and programming frameworks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(5):380–409, 2014.

[4] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedins of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, San Jose, CA, 2012. USENIX.

[5] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, January 2008.

[6] F. Iafrate. A journey from big data to smart data. *Advances in Intelligent Systems and Computing*, 261:25–33, 2014.

[7] J. Chen, D. Dosyn, V. Lytvyn, and A. Sachenko. Smart data integration by goal driven ontology learning. In *Advances in Intelligent Systems and Computing*, volume 529, pages 283–292, 2017.

[8] Diego García-Gil, Julián Luengo, Salvador García, and Francisco Herrera. Enabling Smart Data: Noise filtering in Big Data classification. *Information Sciences*, 479:135–152, 2019.

[9] Isaac Triguero, Diego García-Gil, Jesús Maillo, Julián Luengo, Salvador García, and Francisco Herrera. Transforming big data into smart data: An insight on the use of the k-nearest neighbors algorithm to obtain quality data. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(2):e1289, 2019.

[10] Salvador García, Julián Luengo, and Francisco Herrera. *Data Preprocessing in Data Mining*. Springer, 2015.

[11] Salvador García, Sergio Ramírez-Gallego, Julián Luengo, José Manuel Benítez, and Francisco Herrera. Big data preprocessing: methods and prospects. *Big Data Analytics*, 1(1):9, 2016.

[12] Luís Paulo F. Garcia, André C. P. L. F. de Carvalho, and Ana Carolina Lorena. Effect of label noise in the complexity of classification problems. *Neurocomputing*, 160:108–119, 2015.

[13] J. Fan, F. Han, and H. Liu. Challenges of big data analysis. *National Science Review*, 1(2):293–314, 2014.

[14] Btissam Zerhari. Class noise elimination approach for large datasets based on a combination of classifiers. In *Cloud Computing Technologies and Applications (CloudTech), 2016 2nd International Conference on*, pages 125–130. IEEE, 2016.

[15] M. Hamstra, H. Karau, M. Zaharia, A. Konwinski, and P. Wendell. *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, 2015.

[16] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. Mllib: Machine learning in apache spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016.

[17] Xingquan Zhu and Xindong Wu. Class Noise vs. Attribute Noise: A Quantitative Study. *Artificial Intelligence Review*, 22:177–210, 2004.

[18] Xindong Wu. *Knowledge acquisition from databases*. Ablex Publishing Corp., Norwood, NJ, USA, 1996.

[19] José A. Sáez, Mikel Galar, Julián Luengo, and Francisco Herrera. INFFC: An iterative class noise filter based on the fusion of classifiers with noise sensitivity control. *Information Fusion*, 27:19 – 32, 2016.

[20] Benoît Frénay and Michel Verleysen. Classification in the presence of label noise: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869, 2014.

[21] Shounak Roychowdhury and Witold Pedrycz. Automatic discovery of clusters by removing noisy data. *International Journal of Intelligent Systems*, 33(9):1777–1797, 2018.

[22] Yunlei Li, Lodewyk F.A. Wessels, Dick de Ridder, and Marcel J.T. Reinders. Classification in the presence of class noise using a probabilistic Kernel Fisher method. *Pattern Recognition*, 40(12):3349–3357, 2007.

[23] Q. Miao, Y. Cao, G. Xia, M. Gong, J. Liu, and J. Song. Rboost: Label noise-robust boosting algorithm based on a nonconvex loss function and the numerically stable base learners. *IEEE Transactions on Neural Networks and Learning Systems*, 27(11):2216–2228, 2016.

[24] Charles Bouveyron and Stéphane Girard. Robust supervised classification with mixture models: Learning from data with uncertain labels. *Pattern Recognition*, 42(11):2649–2658, 2009.

[25] Carla E. Brodley and Mark A. Friedl. Identifying Mislabeled Training Data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

[26] S. Verbaeten and A.V. Assche. Ensemble methods for noise elimination in classification problems. In *4th International Workshop on Multiple Classifier Systems*, volume 2709 of *Lecture Notes on Computer Science*, pages 317–325. Springer, 2003.

[27] Taghi M. Khoshgoftaar and Pierre Rebours. Improving software quality prediction by noise filtering techniques. *Journal of Computer Science and Technology*, 22:387–396, 2007.

[28] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[29] Jimmy Lin. Mapreduce is good enough? if all you have is a hammer, throw away everything that's not a nail! *Big Data*, 1(1):28–37, 2013.

[30] T.M. Khoshgoftaar and P. Rebours. Improving software quality prediction by noise filtering techniques. *Journal of Computer Science and Technology*, 22:387–396, 2007.

[31] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[32] A. Lenk, L. Bonorden, A. Hellmanns, N. Roedder, and S. Jaehnichen. Towards a taxonomy of standards in smart data. In *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, pages 1749–1754, 2015.

5 Smart Data based Ensemble for Imbalanced Big Data Classification

- D. García-Gil, J. Holmberg, S. García, N. Xiong, F. Herrera.
  - Status: **Submitted**.

Smart Data based Ensemble for Imbalanced Big Data Classification

Diego García-Gil[*]
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
djgarcia@decsai.ugr.es

Johan Holmberg
Intelligent Future Technologies
School of Innovation Design & Engineering
Mälardalens högskola
Västerås, Sweden, 722 20
johan.holmberg@mdh.se

Salvador García
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
salvagl@decsai.ugr.es

Ning Xiong
Intelligent Future Technologies
School of Innovation Design & Engineering
Mälardalens högskola
Västerås, Sweden, 722 20
ning.xiong@mdh.se

Francisco Herrera
Department of Computer Science
and Artificial Intelligence
University of Granada
Granada, Spain, 18071
herrera@decsai.ugr.es

Abstract

Big Data scenarios pose a new challenge to traditional data mining algorithms, since they are not prepared to work with such amount of data. Smart Data refers to data of enough quality to improve the outcome from a data mining algorithm. Existing data mining algorithms unability to handle Big Datasets prevents the transition from Big to Smart Data. Automation in data acquisition that characterizes Big Data also brings some problems, such as differences in data size per class. This will lead classifiers to lean towards the most represented classes. This problem is known as imbalanced data distribution, where one class is underrepresented in the dataset. Ensembles of classifiers are machine learning methods that improve the performance of a single base classifier by the combination of several of them. Ensembles are not exempt from the imbalanced classification problem. To deal with this issue, the ensemble method have to be designed specifically. In this paper, a data preprocessing ensemble for imbalanced Big Data classification is presented, with focus on two-class problems. Experiments carried out in 21 Big Datasets have proved that our en-

[*]Corresponding author.

semble classifier outperforms classic machine learning models with an added data balancing method, such as Random Forests.

***K*eywords** Big Data · Smart Data · Classification · Ensemble · Imbalanced Data.

1 Introduction

We are experiencing a constant revolution in terms of data generation and transmission speeds. Technologies such as LTE/4G networks have been surpassed by faster standards like the upcoming 5G network [1]. Higher bandwidth and bigger storage is available every few years. The amount of connected devices in the Internet of Things is increasing rapidly [2]. All this only means one thing: more and more data is being generated, transmitted, consumed and stored every year [3].

This increasing amount of data contains very valuable insights for businesses. However, traditional data mining and machine learning techniques are not able to handle these new requirements in terms of size. But even if those methods could handle that data, the time requirements would be astronomical. This is the era of Big Data. Big Data has exceeded the processing capacity of traditional systems, and continues to do so every second. New paradigms, frameworks and computing systems are needed for analyzing and extracting all valuable insights from this data [4]. Big Data can be defined as a high *volume* of data, generated at a high *velocity*, with a potential high *value*, and high *veracity*. This conforms what is known as the four Big Data V's (among many other) [5].

Big Data can be seen as a huge collection of potentially useful data. Recently, the term Smart Data has emerged in the Big Data ecosystem. Smart Data refers to the challenge of extracting quality data from raw Big Data [6]. This new concept aims to achieve quality data (either big or not) with *value* and *veracity* properties [7]. Therefore, the objective of Smart Data based technologies, is to obtain a subset of the data that contains enough quality for the later data mining process to be successful [4].

Data mining can be defined as the set of techniques devoted to construct knowledge patterns through the analysis of structured data [8]. Attending to the type of the pattern targeted, data mining techniques can be classified into two different categories: descriptive methods and predictive techniques. The former aims to discover relationships in the data, whereas the second is focused on discovering how models will behave towards future inputs. Depending on whether the target variable is specified or not, we can distinguish between supervised and unsupervised learning [9]. Supervised learning defines the relation between input and target variables, and predicts its values for new incomes. Classification and regression are two sub-families of supervised learning, depending on the type of the target variable (discrete or continuous) On the other hand, in unsupervised problems the target variable is undefined. Similarly to supervised learning, unsupervised problems can be separated in two different sub-families: the grouping of instances by similarity (clustering), and the identification of associations between the variables (association rules).

Real-world applications are not equal in terms of data size per class. For observing and recording one small or temporary event, many idle observations may be needed. This leads to a class imbalance situation, known as imbalanced class distribution [10, 11]. Imbalanced classification occurs when one class (usually the one that contains the concept of interest) is underrepresented in the dataset [12]. This class usually receives the name of minority or negative class. Some real-world domains are known to suffer from this problem: finances [13], card fraud [14], cancer diagnosis [15], or anomaly detection [16], among others. Data imbalance is one of the biggest challenges in data mining [12].

Among the different approaches proposed to tackle the imbalanced classification problem, we can highlight data sampling as the most popular and widely used technique. This process is typically carried out using data preprocessing methods [17]. Data sampling solutions alter the original dataset by either increasing the number of minority class instances until a certain balance is reached, like Random OverSampling (ROS) [18], or decreasing the number of majority class instances, such as Random UnderSampling (RUS) [18]. We can also find distance based approaches for data balancing, like the "Synthetic Minority Oversampling TEchnique" (SMOTE) [19].

Many other data preprocessing proposals can be found in the literature, such as the majority weighted minority oversampling technique (MWMOTE) or an extension of SMOTE to multiclass problems (MLSMOTE) [20]. Clustering has also been employed effectively for the data imbalanced problem as a way to increase the density of points belonging to certain neighborhoods [21]. These methods balance the data by localizing groups of instances belonging to different neighborhoods, and then applying a data sampling technique, improving the later learning process [22, 23].

Ensembles of classifiers are methods designed to increase the global accuracy of a single classifier by learning different base classifiers and combining all the decisions to return a single label [24]. They correct errors in classification through learning classifiers that have some differences among them [25, 26]. Classification ensembles like XGBoost [27], LightGBM [28] or CatBoost [29] have become some of the best performing methods in machine learning nowadays. Because of their accuracy orientation, ensembles cannot be directly applied to imbalanced datasets, since the base classifiers will ignore the minority class. Their combination with other techniques that tackle the class imbalance problem can improve ensemble performance in these scenarios. These hybrid approaches involve the addition of a data sampling step that allows the classifier to better detect the different classes [30].

Diversity is key when working with ensembles. Diversity can be introduced through small changes in input data, or small changes in the parameters of the classifier. With diverse classifiers, ensembles will be more robust to noise and outliers, and will achieve better performance [31]. Diversity based on changes in input data can be introduced through data preprocessing methods which have a random component. This random component allows ensembles to learn with slightly different data, improving the global performance.

The advent of Big Data have brought new problems in terms of data size and time constraints to classic data preprocessing and data mining algorithms. Despite the extensive list of data preprocessing methods proposed in the literature, only classic algorithms have been adapted to Big Data scenarios [4]. If we attend to imbalanced Big Data classification, only a few classic sampling methods have been proposed to tackle imbalanced Big Data problems [32, 33]. The same applies to ensemble algorithms for classification. The adaptation of novel ensembles to Big Data scenarios is still an ongoing process. Only a handful of ensembles for classification can be found in Big Data environments, such as Random Forest [34, 35].

In this paper, we propose a novel ensemble method for imbalanced Big Data classification, namely Imbalanced Classification Ensemble for Big Data (ICE_BD), focused on binary classification problems. ICE_BD is aimed towards the creation of *smart* and diverse datasets through the use of different data preprocessing methods. This data preprocessing improves the quality of the data, and balances it for the posterior learning process. In particular, ICE_BD proposes the following:

1. ICE_BD performs several data preprocessing methods with a random component to the input data in order to achieve a Smart Data version of the dataset with the desired level of diversity. This produces a diverse and balanced Smart Dataset, that will produce better base classifiers.

2. We take advantage of different data preprocessing methods specifically designed for Big Data problems. For introducing diversity in a dataset, the combination of Random Discretization (RD) and randomized Principal Component Analysis (PCA) proposed in PCARDE [31] is used. For the data balancing step, a clustering-based ROS is proposed.

3. A novel combination of clustering and ROS is presented. ICE_BD performs clustering to the expanded data resulting from the combination of RD and PCA datasets. Then, it balances the clusters found using ROS technique.

4. ICE_BD has been implemented for the Big Data framework Apache Spark [36], and it is available publicly as a Spark package in Spark's third party repository Spark Packages[2].

To assess the performance of our proposal, we have conducted an extensive experimentation. We have tested ICE_BD using 21 Big Data imbalanced datasets from the UCI Machine Learning Repository [37]. These datasets have very different properties among them that will allow us to check the performance and balancing capabilities of our proposal. We have compared ICE_BD against Spark's MLlib implementation of a decision tree, Random Forest [38], and PCARDE, a data preprocessing ensemble present in Spark's community repository Spark Packages [31]. These three classifiers have been tested in four different variants: without any data balancing technique applied, using RUS, ROS and SMOTE. Results obtained have been validated by different Bayesian Sign Tests, in order to assess if ICE_BD achieves statistically better performance than the rest of the methods tested [39].

The rest of this paper is organized as follows: Section 2 gives a description of the imbalanced data classification and Big Data problem. Section 3 describes the proposal in detail. Section 4 shows all the experiments carried out to prove the performance of ICE_BD against several Big Data problems. Finally, Section 5 concludes the paper.

2 Related work

In this section, we provide an introduction to the class imbalance problem in classification, among with the different proposals to tackle it (Section 2.1). Then, the state of Big Data and MapReduce framework are analyzed in Section 2.2.

2.1 Imbalanced Data Classification

In a binary classification problem, a dataset is said to be imbalanced when there is a notable difference in the number of instances belonging to different classes [12]. The class with the greater number of instances is known as the majority class. Similarly, the class with the lower number of instances is known as the minority class, and usually contains the concept of interest.

This problem poses a major challenge to standard classifier learning algorithms, since they will bias towards the class with the greater representation, as their internal search process is guided by a global search measure weighted in favor of accuracy [4]. The imbalanced ratio (IR) measures the difference between the majority and minority classes (an IR of 100 means there is one instance of the minority class per 100 instances of the majority class). In datasets with high IR, classifiers that maximize the accuracy will treat the minority class as noise and ignore it, achieving a high accuracy by just classifying the majority class, because more general rules that models it will be preferred.

Many techniques have been proposed to tackle imbalanced data classification. These techniques can be divided into three groups: data level, algorithm level, and cost-sensitive methods [12]. The former modifies the data to obtain an equally distributed dataset using imbalanced data preprocessing techniques. Algorithm level techniques modifies existing classifiers to improve the detection of the minority class. Cost-sensitive learning solutions combine both data level and algorithm level approaches. They incorporate data modifications by adding cost to instances, and algorithm level adaptations.

[2]https://spark-packages.org/package/djgarcia/Imbalanced-Classification-Ensemble

Aside from these categories, ensemble methods can be classified into their own category [24]. Ensembles combine different mechanisms to produce better results, usually data level approaches [31]. Data level approaches can be easily incorporated into ensemble learning algorithms through the use of data preprocessing methods.

In the literature, data preprocessing methods for imbalanced data classification can be divided into different categories: oversampling methods, undersampling methods, and hybrid approaches[4, 12]. The former replicates the minority class instances until a certain balance is reached. On the other hand, undersampling techniques remove examples from the majority class until the proportion of classes is adjusted. Hybrid approaches combine the previous two techniques, usually starting with an oversampling of the data, followed by an undersampling step that removes samples from both classes, in order to remove noisy instances and improve the classifier performance.

The classic oversampling method, ROS [18], replicates instances from the minority class randomly, until the number of examples of the minority and majority classes is the same. On the other hand, for undersampling, RUS [18] removes instances randomly from the majority class, until both classes have the same amount of instances. The SMOTE algorithm [19] is an improved oversampling method. It adds synthetic instances from the minority class until the class distribution is balanced. Those new instances are created by the interpolation of several minority class instances that belong to the same neighborhood. SMOTE calculates the *k* nearest neighbors of each minority class example. Then, in the segment that connects every instance with its *k* closest neighbors, a synthetic instance is randomly created [20].

Performance evaluation is a key factor for assessing the classification performance. In binary classification problems, the confusion matrix (shown in Table 1) collects correctly and incorrectly classified examples from both classes.

Table 1: Confusion Matrix for Binary Classification Problems

	Positive Prediction	Negative Prediction
Positive class	True Positive (TP)	False Negative (FN)
Negative class	False Positive (FP)	True Negative (TN)

Traditionally, accuracy (Equation 1) has been the most extended and widely used metric for assessing classification performance. However, accuracy is not a valid metric when dealing with imbalanced datasets, since it will not show the classification of both classes, only the majority class, and it will led to wrong conclusions.

$$Acc = \frac{TP + TN}{TP + FN + FP + TN} \tag{1}$$

The Geometric Mean (GM), described in Equation 2, attempts to maximize the accuracy of both minority and majority classes at the same time [40]. The accuracy of both minority and majority classes is represented by the True Positive Rate (TPR) $= \frac{TP}{TP+FN}$ and True Negative Rate (TNR) $= \frac{TN}{TN+FP}$.

$$GM = \sqrt{TPR * TNR} \tag{2}$$

Another popular evaluation metric for imbalanced data is the Are Under the Curve (AUC) [41]. AUC combines the classification performance of both classes, showing the trade-off between the TPR and False Positive Rate. This metric provides a single measure of a classifier performance, compared against a random classifier.

2.2 Big Data and MapReduce

Big Data paradigm has brought new requirements in terms of hardware and software to process this amount of data. Regarding hardware, massive distributed clusters are used everyday for processing this Big Data. However, in the software section lies the biggest challenge. In order to tackle Big Data problems, not only new algorithms are needed, but also new frameworks that operate in distributed clusters are required. Google introduced MapReduce paradigm in 2004 [42]. This paradigm is nowadays the most popular and widely used paradigm for Big Data processing. It was born for allowing users to generate and/or process Big Data problems, while minimizing disk and network use.

MapReduce follows the simple but powerful divide and conquer approach. It can be divided in two phases, the *map* and *reduce* phase. Before entering the *map* stage, all data is partitioned and distributed across the cluster by the master node. The *map* function applies a transformation to each key-value pair located in each computing node. This way, all data is processed independently in a distributed fashion. When the *map* phase is finished, all pairs of data belonging to the same key are redistributed across the cluster. Once all pairs belonging to the same key are located in the same computing node, the *reduce* stage begins. The *reduce* phase can be seen as a summary operation that generates the final values.

MapReduce is a programming paradigm for dealing with Big Data. Apache Hadoop is the most popular open-source implementation of the MapReduce paradigm [43]. Despite its popularity and performance, Hadoop present some important limitations [44]:

- Not suitable for iterative algorithms.
- Very intensive disk usage. All *map* and *reduce* processes are read/write from/to disk.
- No in-memory computation.

Apache Spark can be seen as the natural evolution of Hadoop. It is an open-source framework, focused on speed, easy of use, and advanced analytics [36]. Spark is the solution of Hadoop problems, it has in-memory computation, and allows in-memory data persistence for iterative processes. Spark is built on top of a novel distributed data structure, namely Resilient Distributed Datasets (RDDs) [45]. These data structures are immutable and unsorted by nature. They can be persisted in memory for repetitive uses, and tracked using a lineage, so that each partition can be computed again in case of data lost. RDDs support two types of operations: *transformations* and *actions*. The former transforms the dataset by applying a function to each partition, and produces a new RDD. They are lazy operations, meaning that they are not computed until needed. On the other hand, *actions* triggers all previous *transformations* of an RDD, and return a value.

In 2012, a distributed machine learning library was created as an extra component of Apache Spark, named MLlib [34]. It was released and open-sourced to the community in 2013. The number of contributions have been growing steadily since its conception, making it the most popular machine learning library for Big Data processing nowadays. MLlib includes several algorithms for alike tasks, such as: classification, clustering, regression, or data preprocessing.

3 Imbalanced Classification Ensemble: a Big Data approach

In this section, we describe in detail the proposed method for imbalanced Big Data classification based on data preprocessing, ICE_BD. It is a distributed and parallel ensemble focused on imbalanced Big Data problems, implemented in Apache Spark. In Section 3.1, we explain in detail our proposed iterative imbalanced classification ensemble, ICE_BD. Section 3.2 describes the Spark primitives used for the implementation of the proposal. Finally, Section 3.3 depicts the implementation details of the proposal.

3.1 Imbalanced Classification Ensemble: ICE_BD

This ensemble classifier for imbalanced Big Data is based on the creation of *smart* and diverse datasets for improving the quality of the base classifiers. As stated in the introduction section, diversity is key for ensemble methods. ICE_BD achieves the required diversity by the use of several randomized methods, such as RD and PCA. RD method discretizes the data in $cuts$ intervals by randomly selecting $cuts - 1$ instances. Then, those selected values are sorted and used as thresholds for the discretization of each feature. On the other hand, PCA selects a number of variables in a dataset, whilst retaining as much of the variation present in the dataset as possible. This selection is achieved by finding the combinations of the original features to produce principal components, which are uncorrelated. PCA always produces the same result for a fixed number of principal components. In order to achieve the required diversity, a random number of selected components is used. The number of components must be in the interval $[1, T - 1]$, T being the total number of features of the input data.

Both RD and PCA are applied to the input data. Then, the resulting datasets of RD and PCA are joined together feature-wise. This data is a diverse and more informative version of the dataset, as demonstrated in [31]. Such diverse dataset needs to be balanced in order to correctly classify the minority and majority classes.

A novel combination of hierarchical clustering and oversampling is proposed. Bisecting k-Means is a hierarchical clustering method that uses a divisive (or "top-down") approach [46]. The algorithm starts from a single cluster that contains all points. Iteratively it finds divisible clusters on the bottom level and bisects each of them into two clusters using k-Means, until there are k leaf clusters in total or no leaf clusters are divisible. It has been chosen taking into account that it can often be much faster than regular k-Means. Bisecting k-Means has a linear time complexity. In case of a large number of clusters, Bisecting k-Means is even more efficient than k-Means since there is no need to compare every point to each clusters centroid. It just needs to consider the points in the cluster and their distances to two centroids.

Bisecting k-Means is applied to the resulting data from the join of RD and PCA for finding a random number of neighborhoods with a specified maximum of desired clusters. Found clusters are individually balanced using ROS technique until an IR of 1 is reached. The result of this process is a diverse, balanced and *smart* dataset, which will improve the later learning process.

Finally, using the previously balanced dataset, a decision tree is learned. This decision tree performs a recursive binary partitioning of the input features space. The tree predicts the same label for each leaf partition. These partitions are chosen in a greedy manner, selecting the best split from the set of possible splits, maximizing the information gain at the tree node [47].

ICE_BD preprocessing and learning process is repeated $iter$ times. In Figure 1 we can see a graphic representation of the learning workflow of ICE_BD algorithm.

All previous steps constitute the learning phase of the ensemble. This phase is composed of $iter$ sub-models, each of them containing the thresholds for RD and the weight matrices for PCA. For the prediction phase of the ensemble, for each data point, the same data preprocessing must be applied. First, data is discretized using the same cut points from RD calculated previously. Then, for selecting the same components as the learning phase, the same weight matrix obtained earlier for PCA at a given iteration is applied to the data. Next, the score of each class is predicted according to the decision tree. This score is calculated by the division of the instances at a leaf node, by the total number of instances. This process is repeated $iter$ times, adding those scores for each instance and iteration. Once this process is finished, for each instance, the class with the largest score is selected as the decision of the ensemble.

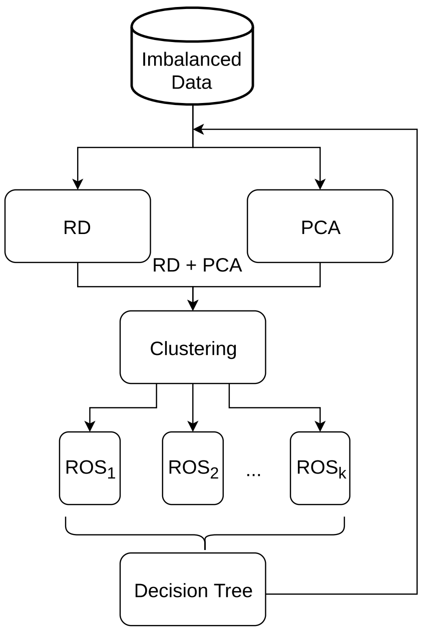


Figure 1: ICE_BD learning flowchart

3.2 Spark Primitives

For the implementation of the ensemble, some basic Spark primitives have been used. Here we outline those more relevant for the ensemble [3]:

- *map*: applies a transformation to each element of an RDD. Once that transformation has been applied, it returns a new RDD.
- *union*: merges two RDDs instance-wise and returns a new RDD.
- *zip*: zips two RDDs together.
- *filter*: selects all the instances in an RDD that satisfy a condition as a new RDD.

These Spark primitives from Spark API are used in the following section, where the implementation of ICE_BD algorithm is described.

3.3 ICE_BD Implementation Details

This section describes all the implementation details of ICE_BD. Both learning and prediction phases are implemented under Apache Spark, following the MapReduce paradigm.

Ensemble Learning Phase

Algorithm 1 explains the ensemble learning phase of ICE_BD. This process is divided into five steps: RD and PCA calculation in order to obtain a diverse dataset, cluster search for the discovery of neighborhoods, cluster balancing, and classifier learning. As stated earlier, ICE_BD starts by discretizing the training data

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

using RD method (lines 8-14). This is performed through the random selection of $cuts - 1$ instances (line 8). Those thresholds are used to discretize the training data using a map function (lines 10-14). For every instance, we assign the corresponding discretized value to each instance's attribute (lines 11-13).

Once RD has been applied to the training data, PCA is performed to select randomly the best principal components (lines 16-19). First, a random number of components is selected in the interval $[1, T - 1]$ (T being the total number of features of the training data) (line 16). Then, PCA is calculated on the training data, and the best $components$ are selected (lines 17-18). Finally, the resulting data from RD and PCA are joined together feature-wise using a distributed zip function (line 19).

The next step is the hierarchical clustering search (lines 21-23). We have used Spark's MLlib distributed implementation of Bisecting k-Means. First, we select a random number of clusters, with a maximum of $maxClust$ (line 21). Then, clusters are calculated using the previously RD and PCA zipped data (line 22). Once that process is finished, the same zipped data is predicted in order to assign a cluster to each data point (line 23). The prediction is done level-by-level from the root node to a leaf node, and at each node among its children the closest to the input point is selected.

Data balancing is applied to each individual cluster found. We apply ROS technique to the minority class of each cluster until both minority and majority classes are equal (lines 25-29). First, and empty set is created for the allocation of the future new dataset (line 25). For each cluster, ROS is applied with an IR of 1 (line 27). That balanced data is added to the empty set (line 28).

Finally, a decision tree is learned using this *smart*, diverse and balanced dataset (line 31). This data preprocessing and learning process is repeated $iter$ times, keeping each iteration, the computed thresholds for RD, the PCA weight matrices, and the learned tree model. Once all trees have been learned, the model is created and returned.

The following input parameters are required: the dataset (*data*), the number of iterations of the ensemble (*iter*), the number of intervals for the discretization (*cuts*), and the maximum number of clusters (*maxClust*).

Ensemble Prediction Phase

The ensemble prediction phase is depicted in Algorithm 2. This process is faster than learning, since clustering and data balancing are not required for prediction. Only the application of RD and PCA is required, both using the same models obtained in the ensemble learning phase. First, the data point is discretized using the same cut points from the learning phase (lines 9-12). Next, the principal components are calculated using the learning phase weight matrix for that iteration (line 13). The next step is to join both RD and PCA results using a zip function (line 14). Finally, the data point is predicted using the decision tree learned in that particular iteration of the ensemble (line 15). The scores of each of the $iter$ predictors are added. Once the instance have all $iter$ scores, the class with the largest weight is selected as the decision of the ensemble and returned (lines 17-18).

4 Experimental Results

In this section, we describe the experimental study carried out to compare the performance of different approaches to deal with imbalanced Big Data against our ensemble proposal. We begin with a description of all datasets employed in the comparison, followed by the performance metrics and parameters of the algorithms used. Finally, we detail all hardware and software resources used to carry out the experimental study.

We have selected a wide spectrum of datasets for assessing the performance of ICE_BD. These datasets have been extracted from the UCI Machine Learning Repository [37]. Specifically, we have selected the

Algorithm 1 ICE_BD learning algorithm

```
1:  Input: data an RDD of type LabeledPoint (features, label).
2:  Input: iter the number of iterations of the ensemble.
3:  Input: cuts the number of intervals for the discretization.
4:  Input: maxClust the maximum number of clusters.
5:  Output: The model created, an object of class ICEModel.
6:  for $i = 0...iter$ do
7:      Random Discretization
8:      $thresholds(i) \leftarrow compute\_RD\_thresholds(data, cuts)$
9:      $rdData \leftarrow$
10:     map $inst \in data$
11:         for $j = 0...length(inst) - 1$ do
12:             $inst \leftarrow discretize(inst(j), thresholds(i)(j))$
13:         end for
14:     end map
15:     PCA
16:     $components \leftarrow random(1, length(data) - 1)$
17:     $pcaModels(i) \leftarrow PCA(data, components)$
18:     $pcaData \leftarrow transform(data, pcaModels(i))$
19:     $joinedData \leftarrow zip(rdData, pcaData)$
20:     Clustering
21:     $k \leftarrow random(1, maxClust)$
22:     $clustModel \leftarrow hierarchicalClustering(joinedData, k)$
23:     $clustData \leftarrow predict(joinedData, clustModel)$
24:     Data Balancing
25:     $balancedData = \emptyset$
26:     for $l = 0...k$ do
27:         $rosData \leftarrow ROS(filter(clustData, "cluster" = l), 1.0)$
28:         $balancedData = union(rosData, balancedData)$
29:     end for
30:     Classifier Learning
31:     $trees(i) \leftarrow decisionTree(balancedData)$
32: end for
33: $return(ICEModel(iter, thresholds, pcaModels, trees))$
```

Poker Hand dataset, the Record Linkage Comparison Patterns (RLCP), SUSY and HIGGS datasets [48], the KDD Cup 1999 dataset, and ECBDL14 dataset [49]. ECBDL14 dataset was used as a reference at the ML competition of the Evolutionary Computation for Big Data and Big Learning, under the international conference GECCO-2014. It is a highly imbalanced binary classification dataset, composed of 98% of negative instances. For this problem, we have used two subsets with the same IR and the best 90 features found in the competition [49].

Since some of the selected datasets have more than two classes, we have sampled new binary datasets from them to address each case separately. In particular, we have selected new datasets using the majority classes against the minority classes. Table 2 shows all the details of the datasets, including the number of instances (#Inst.), number of attributes (#Atts.), class distribution and IR.

Algorithm 2 ICEModel prediction algorithm

```
1: Input: iter the number of iterations of the ensemble.
2: Input: cuts the cut points for the discretization.
3: Input: pcaModels the models for performing PCA.
4: Input: trees the models of the learned trees.
5: Output: The label of the test data point.
6: function PREDICT($test : LabeledPoint$)
7:     $scorePredictions \leftarrow \emptyset$
8:     for $i = 0...iter$ do
9:         $rdData \leftarrow \emptyset$
10:        for $j = 0...length(test) - 1$ do
11:            $rdData(c) \leftarrow discretize(test(j), cuts(i)(j))$
12:        end for
13:        $pcaData \leftarrow transform(test, pcaModels(i))$
14:        $joinedData \leftarrow zip(rdData, pcaData)$
15:        $scorePredictions \leftarrow scorePredictions + predict(joinedData, trees(i))$
16:    end for
17:    $label \leftarrow indexOfMax(scorePredictions)$
18:    $return(label)$
19: end function
```

Table 2: Datasets used in the analysis

Dataset	#Inst.	#Atts.	%Class(maj; min)	IR
poker0_vs_2	450,022	10	(91.32; 8.68)	10.52
poker0_vs_3	428,464	10	(95.99; 4.01)	23.94
poker0_vs_4	414,032	10	(99.23; 0.77)	128.06
poker0_vs_5	412,600	10	(99.60; 0.40)	250.59
poker0_vs_6	411,990	10	(99.70; 0.30)	337.81
poker1_vs_2	385,842	10	(89.89; 10.11)	8.89
poker1_vs_3	363,932	10	(95.24; 4.76)	20.03
poker1_vs_4	349,891	10	(99.11; 0.89)	110.82
poker1_vs_5	347,695	10	(99.55; 0.45)	221.17
poker1_vs_6	347,867	10	(99.68; 0.32)	308.77
rlcp	4,599,153	2	(99.63; 0.37)	271.12
susy_ir4	2,712,173	18	(80.00; 20.00)	4.00
susy_ir8	2,440,956	18	(88.89; 11.11)	7.99
susy_ir16	2,305,347	18	(94.12; 5.88)	15.99
higgs_ir4	5,829,123	28	(80.00; 20.00)	3.99
higgs_ir8	5,246,211	28	(88.89; 11.11)	8.00
higgs_ir16	4,954,752	28	(94.12; 5.88)	15.99
ecbdl14-1.2mill-90	960,000	90	(98.01; 1.99)	49.29
ecbdl14-10mill-90	9,600,000	90	(98.00; 2.00)	48.94
kddcup_normal_vs_DOS	1,942,816	41	(79.96; 20.04)	3.99
kddcup_DOS_vs_R2L	3,107,709	41	(99.97; 0.03)	3,475.18

All the datasets have been partitioned using a 5 fold cross-validation scheme. This means that all datasets have been partitioned in 5 folds, with 80% (four folds) of instances devoted to training, and the rest 20% for testing. The results provided are the average of running the algorithms with the five folds per dataset.

We have carried out a comparison of ICE_BD against three classification methods: Spark's MLlib distributed implementation of decision trees, Random Forest, and PCARDE, a data preprocessing ensemble present in Spark's community repository Spark Packages [31]. For balancing the data when those classifiers are used, we have employed the most widely used data balancing methods: RUS, ROS and SMOTE. For SMOTE algorithm, an implementation available in the Spark Packages repository has been used: SMOTE_BD [33]. The parameters used for the data preprocessing algorithms and the different classifiers are described in Table 3. Since ensembles correct errors across many base classifiers, we have chosen to increase the depth of the decision tree in ICE_BD for a better discrimination between both minority and majority classes. ROS and SMOTE_BD have been configured to balance the dataset to an $IR = 1$.

Table 3: Parameter settings for the data preprocessing and classification algorithms

Algorithm	Parameters
ROS_BD	ir = 1
SMOTE_BD	k = 5, distance = "euclidean", ir = 1
Decision Tree	impurity = "gini", maxDepth = 5, maxBins = 32
Random Forest	nTrees = 200, impurity = "gini", maxDepth = 4 maxBins = 32
PCARDE	nTrees = 10, bins = 5
ICE_BD	bins = 5, trees = 10, maxClust = 10, treeDepth = 10

As stated earlier, when dealing with imbalanced data it is important to choose the right performance metric. Accuracy is not useful in highly imbalanced datasets, because we can achieve great accuracy by just classifying correctly the majority class, while the minority class is ignored. For this reason, we have selected the two most widely used metrics for imbalanced classification: GM and AUC.

All the experimentation have been carried out in a cluster composed of 11 computing nodes and one master node. The computing nodes have the following hardware characteristics: 2 x Intel Core i7-4930K, 6 cores per processor, 3.40 GHz, 12 MB cache, 4 TB HDD, 64 GB RAM. Regarding software, we have used the following configuration: Apache Hadoop 2.9.1, Apache Spark 2.2.0, 198 cores (18 cores/node), 638 GB RAM (58 GB/node).

Table 4: Average results for the imbalanced Big Data cases of study using the GM measure. The highest GM value per dataset is stressed in bold.

Dataset	Baseline			RUS			ROS			SMOTE_BD			ICE_BD
	DT	RF	PCARDE	DT	RF	PCARDE	DT	RF	PCARDE	DT	RF	PCARDE	
poker0_vs_2	0.1986	0.0000	0.0000	0.5847	0.7086	0.5859	0.5272	0.5455	0.5813	0.5249	0.4846	0.6604	**0.8274**
poker0_vs_3	0.1261	0.0000	0.0000	0.5248	0.6954	0.6574	0.6890	0.7003	0.6423	0.5728	0.5728	0.6983	**0.8324**
poker0_vs_4	0.2383	0.0000	0.0000	0.8427	0.8407	0.8438	0.8468	0.8481	0.9029	0.7773	0.7757	0.9102	**0.9880**
poker_0_vs_5	0.0000	0.0000	0.7002	0.8745	0.8530	0.9735	0.8745	0.8743	0.9555	0.4840	0.4582	**1.0000**	0.9974
poker0_vs_6	0.0000	0.0000	0.0000	0.6197	0.6615	0.7250	0.5935	0.7005	0.5860	0.6209	0.5729	0.7415	**0.7998**
poker1_vs_2	0.0367	0.0000	0.0000	0.5993	0.5437	0.4893	0.5600	0.5539	0.5328	0.4136	0.3452	0.5380	**0.6635**
poker1_vs_3	0.0776	0.0000	0.0402	0.5948	0.5981	0.5193	0.6129	0.6204	0.5347	0.5073	0.4543	0.5720	**0.6396**
poker1_vs_4	0.0000	0.0000	0.0000	0.7675	0.7506	0.8620	0.7678	0.7424	0.8789	0.6571	0.7050	0.8375	**0.9361**
poker1_vs_5	0.0000	0.0000	0.7002	0.5423	0.7845	0.9522	0.5833	0.6073	0.9999	0.4649	0.4574	0.9964	**1.0000**
poker1_vs_6	0.0000	0.0000	0.0000	0.6190	0.5673	0.6105	0.6359	0.6269	0.5129	0.6327	0.5611	0.6060	**0.6576**
rlcp	0.0874	0.0927	0.0927	0.9310	0.9302	0.9301	0.9299	0.9305	0.9310	0.9306	0.9297	0.9302	**0.9313**
susy_ir4	0.6870	0.6187	0.6615	0.7679	0.7651	0.7737	0.7679	0.7647	0.7748	0.7622	0.7654	0.7757	**0.7824**
susy_ir8	0.5713	0.5482	0.5690	0.7671	0.7660	0.7737	0.7678	0.7655	0.7738	0.7623	0.7661	0.7746	**0.7802**
susy_ir16	0.5162	0.5205	0.4531	0.7667	0.7651	0.7725	0.7661	0.7647	0.7728	0.7627	0.7654	0.7746	**0.7815**
higgs_ir4	0.3498	0.0541	0.2712	0.6584	0.6695	0.6927	0.6613	0.6702	0.6891	0.6446	0.6622	0.6857	**0.7141**
higgs_ir8	0.2398	0.0000	0.1774	0.6612	0.6698	0.6893	0.6630	0.6688	0.6841	0.6479	0.6511	0.6827	**0.7174**
higgs_ir16	0.1368	0.0000	0.0000	0.6575	0.6679	0.6874	0.6600	0.6691	0.6886	0.6512	0.6506	0.6812	**0.7121**
ecbdl14-1.2mill-90	0.0143	0.0000	0.0000	0.7006	0.7056	0.7067	0.7001	0.7032	0.7141	0.6662	0.6920	0.6920	**0.7225**
ecbdl14-10mill-90	0.0000	0.0000	0.0000	0.6979	0.7047	0.7073	0.6976	0.7039	0.7082	0.6736	0.6850	0.6885	**0.7272**
kddcup_normal_vs_DOS	0.9998	0.9998	0.9998	0.9996	0.9996	0.9998	0.9996	0.9996	0.9998	0.9997	0.9996	0.9998	**1.0000**
kddcup_DOS_vs_R2L	0.9756	0.9934	0.9912	0.9976	0.9997	0.9998	0.9934	**1.0000**	0.9978	0.0000	**1.0000**	0.9976	0.9978
Average	0.2502	0.1823	0.2694	0.7226	0.7451	0.7596	0.7285	0.7362	0.7553	0.6265	0.6645	0.7735	**0.8194**

Table 5: Average results for the imbalanced Big Data cases of study using the AUC measure. The highest AUC value per dataset is stressed in bold.

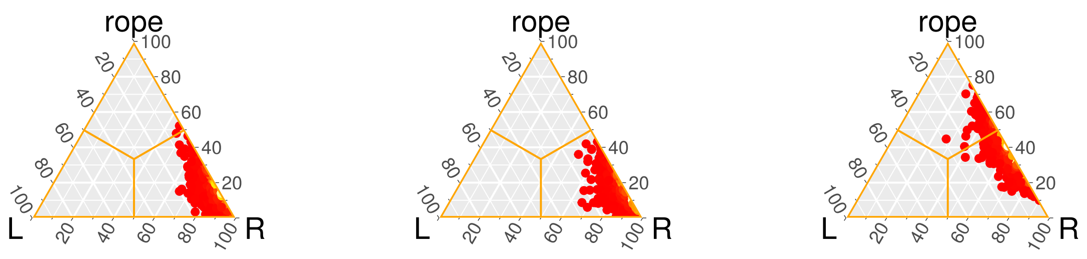
Dataset	Baseline			RUS			ROS			SMOTE_BD			ICE_BD
	DT	RF	PCARDE	DT	RF	PCARDE	DT	RF	PCARDE	DT	RF	PCARDE	
poker0_vs_2	0.5197	0.5000	0.5000	0.5456	0.6045	0.6152	0.6148	0.7093	0.6145	0.5997	0.5919	0.6653	**0.8274**
poker0_vs_3	0.5080	0.5000	0.5000	0.6946	0.7191	0.6651	0.5733	0.7151	0.6648	0.6174	0.6174	0.7030	**0.8326**
poker0_vs_4	0.5284	0.5000	0.5000	0.8477	0.8500	0.9029	0.8431	0.8440	0.8440	0.7787	0.7785	0.9102	**0.9880**
poker_0_vs_5	0.5000	0.5000	0.7451	0.8824	0.8822	0.9565	0.8824	0.8638	0.9738	0.5028	0.5177	**1.0000**	0.9974
poker0_vs_6	0.5000	0.5000	0.5000	0.5935	0.7015	0.6408	0.6920	0.7160	0.7284	0.6928	0.6543	0.7427	**0.8167**
poker1_vs_2	0.5007	0.5000	0.5000	0.6089	0.5887	0.5521	0.6174	0.5872	0.5513	0.5016	0.4937	0.5453	**0.6647**
poker1_vs_3	0.5030	0.5000	0.5008	0.6129	0.6307	0.5763	0.6034	0.6337	0.5701	0.5596	0.5386	0.5734	**0.6430**
poker1_vs_4	0.5000	0.5000	0.5000	0.7751	0.7551	0.8790	0.7678	0.7607	0.8646	0.6647	0.7173	0.8384	**0.9372**
poker1_vs_5	0.5000	0.5000	0.7452	0.5833	0.6073	0.9999	0.5516	0.8068	0.9532	0.4979	0.5226	0.9964	**1.0000**
poker1_vs_6	0.5000	0.5000	0.5000	0.6455	0.6269	0.5960	0.6383	0.6047	0.6200	0.6440	0.5912	0.6380	**0.6927**
rlcp	0.5038	0.5043	0.5043	0.9318	0.9322	0.9328	0.9322	0.9319	0.9320	0.9325	0.9315	0.9320	**0.9327**
susy_ir4	0.7260	0.6856	0.7115	0.7687	0.7665	0.7768	0.7689	0.7668	0.7763	0.7642	0.7656	0.7769	**0.7854**
susy_ir8	0.6603	0.6477	0.6592	0.7688	0.7670	0.7754	0.7679	0.7678	0.7759	0.7645	0.7664	0.7761	**0.7821**
susy_ir16	0.6315	0.6333	0.6020	0.7667	0.7662	0.7758	0.7671	0.7664	0.7741	0.7677	0.7656	0.7757	**0.7838**
higgs_ir4	0.5535	0.5014	0.5344	0.6638	0.6703	0.6891	0.6636	0.6695	0.6930	0.6542	0.6640	0.6858	**0.7142**
higgs_ir8	0.5270	0.5000	0.5153	0.6640	0.6689	0.6841	0.6633	0.6699	0.6896	0.6484	0.6542	0.6829	**0.7174**
higgs_ir16	0.5091	0.5000	0.5000	0.6640	0.6692	0.6887	0.6638	0.6680	0.6874	0.6519	0.6541	0.6814	**0.7122**
ecbdl14-1.2mill-90	0.5001	0.5000	0.5000	0.7006	0.7034	0.7141	0.7029	0.7056	0.7068	0.6700	0.6939	0.6943	**0.7236**
ecbdl14-10mill-90	0.5000	0.5000	0.5000	0.6977	0.7039	0.7083	0.6979	0.7047	0.7073	0.6799	0.6885	0.6901	**0.7273**
kddcup_normal_vs_DOS	0.9998	0.9998	0.9998	0.9996	0.9996	0.9998	0.9996	0.9996	0.9998	0.9997	0.9996	0.9998	**1.0000**
kddcup_DOS_vs_R2L	0.9759	0.9934	0.9912	0.9934	**1.0000**	0.9978	0.9976	0.9997	0.9998	0.5000	**1.0000**	0.9976	0.9978
Average	0.5784	0.5698	0.5957	0.7337	0.7435	0.7679	0.7338	0.7567	0.7679	0.6711	0.6955	0.7764	**0.8227**

In Table 4 we can see the average results for the GM measure using the three classifiers combined with the three data preprocessing strategies, compared with ICE_BD. As can be observed, the *Baseline* with no data imbalanced handling often results in a GM value of 0. That value represents that one of the classes (the minority in particular) is being missclassified completely. All classifiers are benefiting from the data balancing done by RUS and ROS. All three classifiers achieve very similar results when using either RUS or ROS. This can be explained by the high data redundancy present in Big Data datasets. SMOTE_BD is able to achieve an improvement in the GM measure when using PCARDE as a classifier. ICE_BD is be the best performing method for almost every tested dataset. On average, ICE_BD achieves an improvement of nearly 0.5 points in the GM measure. This shows the good performance of the clustering-based data oversampling of ICE_BD.

The AUC average results are depicted in Table 5. Again, the *Baseline* with no preprocessing achieves low values of AUC. The first difference when comparing AUC with the GM measure, is that AUC shows a value of 0.5 when a class is completely missclassified. RUS and ROS methods are producing very similar results in terms of AUC measure. Regarding SMOTE_BD, as observed with the GM measure, only PCARDE is able to achieve an AUC improvement with respect to RUS and ROS. The same improvement seen with the GM measure can be seen with the AUC measure for ICE_BD. It is the best performing data preprocessing and ensemble method among the different strategies tested.

For a deeper analysis of the results, we have performed a Bayesian Sign Test in order to analyze if ICE_BD is statistically better than the rest of the methods [39]. Bayesian Sign Tests obtain a distribution of the differences between two algorithms, and make a decision when 95% of the distribution is in one of the three regions: left, rope (region of practical equivalence), or right [50].

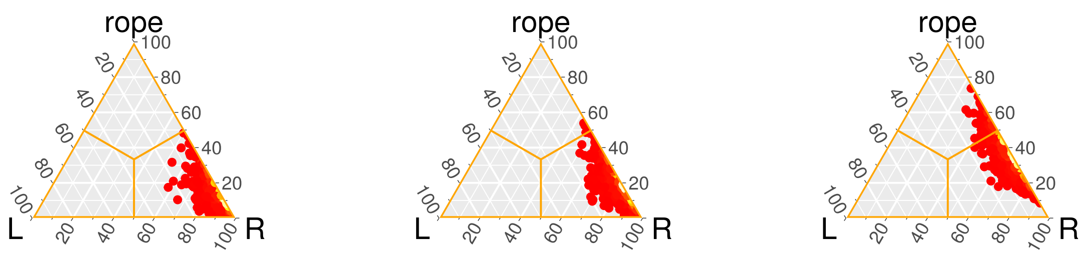
The Bayesian Sign Test is applied to the mean GM and AUC measures of each dataset. We have selected the best performing scenario for each classification method depending on the measure employed. In Figure 2 we can see a comparison of ICE_BD against the decision tree with ROS, Random Forest with RUS, and PCARDE with SMOTE_BD, all using the GM measure. On the other hand, for AUC measure (showed in Figure 3), the decision tree is combined with RUS, Random Forest with ROS, and PCARDE with SMOTE_BD. As we can observe, both GM and AUC Bayesian Sign Tests are showing very similar results. The probability of the difference being to the left is minimal for ICE_BD. This means that the Bayesian Sign Test is assigning a probability of 0 to these classification methods performing better than our proposal.



DT (L) vs ICE_BD (R) RF (L) vs ICE_BD (R) PCARDE (L) vs ICE_BD (R)

Figure 2: Bayesian Sign Test heatmap of DT, RF and PCARDE best results, against ICE_BD for GM measure

In order to assess the performance in Big Data scenarios, we shall analyze the computing times for ICE_BD and the rest of the methods. In classification tasks, prediction times are more important than learning times, since models are only learned once. Such times can be seen in Table 6. As expected, the decision tree is the fastest in prediction, since it only requires to predict a simple tree. ÇRandom Forest also achieve good



DT (L) vs ICE_BD (R) RF (L) vs ICE_BD (R) PCARDE (L) vs ICE_BD (R)

Figure 3: Bayesian Sign Test heatmap of DT, RF and PCARDE best results, against ICE_BD for AUC measure

predictions times, since neither the decision tree nor Random Forest use data preprocessing when predicting. In spite of this, ICE_BD is very competitive in prediction. It is less than one second slower than PCARDE in predicting imbalanced Big Datasets.

In view of these results, we can conclude that:

- The combination of RD and PCA for creating highly diverse ensembles proposed in PCARDE achieves excellent performance in imbalanced Big Datasets.
- The proposed addition of hierarchical clustering and ROS for balancing the data has proven to be able to effectively produce balanced datasets, while adding another level of diversity to the ensemble.
- ICE_BD has shown to be the best performing method for the majority of tested datasets.
- ICE_BD has proven to be able to create Smart Data base classifiers and to address Big Data imbalanced problems effectively.

Table 6: Average prediction times (in seconds) for the imbalanced Big Data cases of study.

Dataset	Baseline			RUS			ROS			SMOTE_BD			ICE_BD
	DT	RF	PCARDE	DT	RF	PCARDE	DT	RF	PCARDE	DT	RF	PCARDE	
poker0_vs_2	0.07	1.96	3.38	0.03	1.98	2.32	0.03	1.91	2.74	0.03	1.85	2.58	3.60
poker0_vs_3	0.07	1.92	3.28	0.03	1.68	2.23	0.03	1.65	2.61	0.03	1.84	2.51	3.19
poker0_vs_4	0.08	1.72	3.19	0.03	1.72	2.12	0.04	1.80	2.61	0.03	1.63	2.18	3.07
poker_0_vs_5	0.59	1.59	3.10	0.03	1.67	2.41	0.03	1.50	2.53	0.03	1.59	2.12	2.38
poker0_vs_6	0.08	1.82	3.09	0.03	1.88	2.45	0.03	1.72	2.55	0.03	1.66	2.16	2.75
poker1_vs_2	0.08	1.71	2.95	0.02	1.66	1.98	0.03	1.57	2.29	0.03	1.59	2.00	3.62
poker1_vs_3	0.08	1.82	3.13	0.03	1.67	2.21	0.03	1.70	2.19	0.02	1.77	2.02	3.16
poker1_vs_4	0.08	1.67	2.75	0.03	1.59	2.18	0.03	1.54	2.19	0.03	1.59	1.84	2.34
poker1_vs_5	0.61	1.47	2.97	0.03	1.45	1.85	0.03	1.43	2.31	0.03	1.48	1.95	2.50
poker1_vs_6	0.08	1.54	2.89	0.03	1.41	2.09	0.04	1.36	2.26	0.03	1.34	1.95	2.44
rlcp	0.14	2.30	13.28	0.04	2.28	12.29	0.04	2.30	12.37	0.04	2.26	12.64	15.35
susy_ir4	0.08	1.37	9.05	0.04	1.40	8.31	0.04	1.48	8.14	0.04	1.54	8.18	11.34
susy_ir8	0.10	1.07	8.18	0.04	1.17	8.30	0.05	1.09	7.79	0.04	1.11	7.96	10.77
susy_ir16	0.10	1.14	7.62	0.04	1.25	7.22	0.04	1.20	7.22	0.04	1.23	7.61	9.94
higgs_ir4	0.23	2.97	17.31	0.06	2.65	16.76	0.06	2.89	16.81	0.06	2.87	16.39	22.86
higgs_ir8	0.12	2.28	16.59	0.09	2.35	15.21	0.07	2.21	15.01	0.06	2.29	15.00	21.73
higgs_ir16	0.24	2.45	14.71	0.26	2.43	13.80	0.06	2.43	14.19	0.06	2.54	13.57	18.96
ecbdl14-1.2mill-90	0.19	0.62	5.15	0.04	0.67	4.72	0.04	0.65	4.64	0.04	0.87	4.30	6.66
ecbdl14-10mill-90	0.21	5.24	30.81	0.05	5.56	31.42	0.06	5.57	31.17	0.06	5.66	30.87	44.36
kddcup_normal_vs_DOS	0.21	0.99	6.44	0.04	0.86	5.99	0.04	0.95	5.99	0.04	0.94	6.15	9.01
kddcup_DOS_vs_R2L	0.12	2.48	9.67	0.03	2.09	8.94	0.04	2.02	9.04	0.04	2.19	8.99	9.83
Average	0.10	2.22	6.52	0.03	2.04	5.63	0.03	1.96	5.89	0.03	2.02	5.78	6.72

5 Conclusions

Imbalanced data binary classification is a challenging task to which many researchers have devoted their efforts. In Big Data scenarios, this problem is aggravated due to the amount of data available. Although popular data balancing approaches like RUS, ROS and SMOTE have proven to be effective for normal-sized problems, in Big Data environments they are less so. These techniques can be combined to create ensembles of classifiers for improving the discrimination of both classes. The huge data redundancy that characterizes Big Data problems, hinders the performance of these algorithms, since they are replicating already redundant data points.

In this paper, we have proposed a Smart Data based ensemble for dealing with the imbalanced class classification problem in Big Data, namely ICE_BD. ICE_BD makes use of the combination of RD and PCA for achieving a highly diverse dataset. We have proposed a novel combination of clustering and oversampling with ROS for achieving a balanced dataset while adding another level of diversity. Our proposal has been tested using several Big Datasets with different characteristics, and two metrics focused on imbalanced classification, GM and AUC. ICE_BD has achieved statistically the best performance in both GM and AUC for almost every tested dataset, proving its efficiency when dealing with Big Data imbalanced datasets.

Acknowledgments

This work is supported by the Spanish National Research Project TIN2017-89517-P.

References

[1] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. Soong, J. C. Zhang, What will 5g be?, IEEE Journal on selected areas in communications 32 (6) (2014) 1065–1082.

[2] A. Botta, W. De Donato, V. Persico, A. Pescapé, Integration of cloud computing and internet of things: a survey, Future generation computer systems 56 (2016) 684–700.

[3] 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.

[4] J. Luengo, D. García-Gil, S. Ramírez-Gallego, S. García, F. Herrera, Big Data Preprocessing. Enabling Smart Data, Springer, 2020, 978-3-030-39104-1.

[5] N. Khan, A. Naim, M. R. Hussain, Q. N. Naveed, N. Ahmad, S. Qamar, The 51 v's of big data: Survey, technologies, characteristics, opportunities, issues and challenges, in: Proceedings of the International Conference on Omni-Layer Intelligent Systems, COINS '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 19–24.

[6] I. Cordón, J. Luengo, S. García, F. Herrera, F. Charte, Smartdata: Data preprocessing to achieve smart data in R, Neurocomputing 360 (2019) 1 – 13.

[7] Y. Sun, H. Song, A. J. Jara, R. Bie, Internet of things and big data analytics for smart and connected communities, IEEE Access 4 (2016) 766–773.

[8] X. Wu, X. Zhu, G.-Q. Wu, W. Ding, Data mining with big data, IEEE transactions on knowledge and data engineering 26 (1) (2013) 97–107.

[9] I. H. Witten, E. Frank, M. A. Hall, C. J. Pal, Data Mining: Practical machine learning tools and techniques, Morgan Kaufmann, 2016.

[10] B. Krawczyk, Learning from imbalanced data: open challenges and future directions, Progress in Artificial Intelligence 5 (4) (2016) 221–232.

[11] F. Thabtah, S. Hammoud, F. Kamalov, A. Gonsalves, Data imbalance in classification: Experimental evaluation, Information Sciences 513 (2020) 429 – 441.

[12] A. Fernández, S. García, M. , R. C. Prati, B. Krawczyk, F. Herrera, Learning from imbalanced data sets, Springer, 2018.

[13] J. Sun, H. Li, H. Fujita, B. Fu, W. Ai, Class-imbalanced dynamic financial distress prediction based on adaboost-svm ensemble combined with smote and time weighting, Information Fusion 54 (2020) 128 – 144.

[14] F. Carcillo, A. D. Pozzolo, Y.-A. L. Borgne, O. Caelen, Y. Mazzer, G. Bontempi, SCARFF: A scalable framework for streaming credit card fraud detection with spark, Information Fusion 41 (2018) 182 – 194.

[15] S. Fotouhi, S. Asadi, M. W. Kattan, A comprehensive data level analysis for cancer diagnosis on imbalanced data, Journal of Biomedical Informatics 90 (2019) 103089.

[16] Q. Chen, A. Zhang, T. Huang, Q. He, Y. Song, Imbalanced dataset-based echo state networks for anomaly detection, Neural Computing and Applications (2018) 1–10.

[17] S. García, J. Luengo, F. Herrera, Data Preprocessing in Data Mining, Springer, 2015.

[18] G. E. A. P. A. Batista, R. C. Prati, M. C. Monard, A study of the behavior of several methods for balancing machine learning training data, SIGKDD Explor. Newsl. 6 (1) (2004) 20–29.

[19] N. Chawla, K. Bowyer, L. Hall, W. Kegelmeyer, SMOTE: Synthetic minority over-sampling technique, Journal of Artificial Intelligence Research 16 (2002) 321–357.

[20] A. Fernández, S. García, F. Herrera, N. V. Chawla, SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary, Journal of artificial intelligence research 61 (2018) 863–905.

[21] S. Nejatian, H. Parvin, E. Faraji, Using sub-sampling and ensemble clustering techniques to improve performance of imbalanced classification, Neurocomputing 276 (2018) 55–66.

[22] W.-C. Lin, C.-F. Tsai, Y.-H. Hu, J.-S. Jhang, Clustering-based undersampling in class-imbalanced data, Information Sciences 409 (2017) 17–26.

[23] Y.-P. Zhang, L.-N. Zhang, Y.-C. Wang, Cluster-based majority under-sampling approaches for class imbalance learning, in: 2010 2nd IEEE International Conference on Information and Financial Engineering, IEEE, 2010, pp. 400–404.

[24] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, F. Herrera, A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42 (4) (2012) 463–484.

[25] D. García-Gil, J. Luengo, S. García, F. Herrera, Enabling Smart Data: Noise filtering in Big Data classification, Information Sciences 479 (2019) 135 – 152.

[26] D. García-Gil, F. Luque-Sánchez, J. Luengo, S. García, F. Herrera, From Big to Smart Data: Iterative ensemble filter for noise filtering in Big Data classification, International Journal of Intelligent Systems 34 (12) (2019) 3260–3274.

[27] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, ACM, 2016, pp. 785–794.

[28] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, Lightgbm: A highly efficient gradient boosting decision tree, in: Advances in Neural Information Processing Systems, 2017, pp. 3146–3154.

[29] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, A. Gulin, Catboost: unbiased boosting with categorical features, in: Advances in Neural Information Processing Systems, 2018, pp. 6638–6648.

[30] J. Bi, C. Zhang, An empirical comparison on state-of-the-art multi-class imbalance learning algorithms and a new diversified ensemble learning scheme, Knowledge-Based Systems 158 (2018) 81 – 93.

[31] D. García-Gil, S. Ramírez-Gallego, S. García, F. Herrera, Principal Components Analysis Random Discretization Ensemble for Big Data, Knowledge-Based Systems 150 (2018) 166 – 174.

[32] A. Fernández, S. del Río, N. V. Chawla, F. Herrera, An insight into imbalanced big data classification: outcomes and challenges, Complex & Intelligent Systems 3 (2) (2017) 105–120.

[33] M. Basgall, W. Hasperué, M. Naiouf, A. Fernández, F. Herrera, SMOTE-BD: An Exact and Scalable Oversampling Method for Imbalanced Classification in Big Data, Journal of Computer Science and Technology 18 (2018) e23.

[34] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al., Mllib: Machine learning in apache spark, The Journal of Machine Learning Research 17 (1) (2016) 1235–1241.

[35] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.

[36] M. Hamstra, H. Karau, M. Zaharia, A. Konwinski, P. Wendell, Learning Spark: Lightning-Fast Big Data Analytics, O'Reilly Media, 2015.

[37] D. Dua, C. Graff, UCI machine learning repository (2017).
URL `http://archive.ics.uci.edu/ml`

[38] 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, Journal of Machine Learning Research 17 (34) (2016) 1–7.

[39] J. Carrasco, S. García, M. del Mar Rueda, F. Herrera, rNPBST: An R Package Covering Non-parametric and Bayesian Statistical Tests, in: F. J. Martínez de Pisón, R. Urraca, H. Quintián, E. Corchado (Eds.), Hybrid Artificial Intelligent Systems, Springer International Publishing, Cham, 2017, pp. 281–292.

[40] R. Barandela, J. Sánchez, V. García, E. Rangel, Strategies for learning in class imbalance problems, Pattern Recognition 36 (3) (2003) 849 – 851.

[41] J. Huang, C. X. Ling, Using auc and accuracy in evaluating learning algorithms, IEEE Transactions on knowledge and Data Engineering 17 (3) (2005) 299–310.

[42] J. Dean, S. Ghemawat, Mapreduce: Simplified data processing on large clusters, in: Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6, OSDI'04, USENIX Association, USA, 2004, p. 10.

[43] T. White, Hadoop: The Definitive Guide, O'Reilly Media, Inc., 2012.

[44] 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.

[45] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Proceedins of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), USENIX, San Jose, CA, 2012, pp. 15–28.

[46] M. Steinbach, G. Karypis, V. Kumar, et al., A comparison of document clustering techniques, in: KDD workshop on text mining, Vol. 400, Boston, 2000, pp. 525–526.

[47] L. Rokach, O. Maimon, Data Mining With Decision Trees: Theory and Applications, 2nd Edition, World Scientific Publishing Co., Inc., USA, 2014.

[48] P. Baldi, P. Sadowski, D. Whiteson, Searching for exotic particles in high-energy physics with deep learning, Nature communications 5 (2014) 4308.

[49] I. Triguero, S. del Río, V. López, J. Bacardit, J. M. Benítez, F. Herrera, ROSEFW-RF: the winner algorithm for the ECBDL'14 big data competition: an extremely imbalanced big data bioinformatics problem, Knowledge-Based Systems 87 (2015) 69–79.

[50] A. Benavoli, G. Corani, J. Demšar, M. Zaffalon, Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis, The Journal of Machine Learning Research 18 (1) (2017) 2653–2688.

Bibliography

[AB13] Ahmad A. and Brown G. (2013) Random projection random discretization ensembles—ensembles of linear multivariate decision trees. *IEEE Transactions on Knowledge and data Engineering* 26(5): 1225–1239.

[AIS93] Agrawal R., Imieliński T., and Swami A. (Junio 1993) Mining association rules between sets of items in large databases. *SIGMOD Rec.* 22(2): 207–216.

[BF99] Brodley C. E. and Friedl M. A. (1999) Identifying mislabeled training data. *Journal of artificial intelligence research* 11: 131–167.

[BPM04] Batista G. E. A. P. A., Prati R. C., and Monard M. C. (Junio 2004) A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explor. Newsl.* 6(1): 20–29.

[Bre96] Breiman L. (1996) Bagging predictors. *Machine learning* 24(2): 123–140.

[Bre01] Breiman L. (2001) Random forests. *Machine learning* 45(1): 5–32.

[BSGR03] Barandela R., Sánchez J., García V., and Rangel E. (2003) Strategies for learning in class imbalance problems. *Pattern Recognition* 36(3): 849 – 851.

[CCGH19] Charte D., Charte F., García S., and Herrera F. (Apr 2019) A snapshot on nonstandard supervised learning problems: taxonomy, relationships, problem transformations and algorithm adaptations. *Progress in Artificial Intelligence* 8(1): 1–14.

[CCS12] Chen H., Chiang R. H., and Storey V. C. (2012) Business intelligence and analytics: From big data to big impact. *MIS quarterly* 36(4).

[CG16] Chen T. and Guestrin C. (2016) Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794. ACM.

[CM07] Cherkassky V. and Mulier F. M. (2007) *Learning from data: concepts, theory, and methods*. John Wiley & Sons.

[Das00] Dasgupta S. (2000) Experiments with random projection. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, UAI '00, pp. 143–151. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[DG04] Dean J. and Ghemawat S. (2004) Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*, OSDI'04, page 10. USENIX Association, USA.

[DHS12] Duda R. O., Hart P. E., and Stork D. G. (2012) *Pattern classification*. John Wiley & Sons.

[Die00a] Dietterich T. G. (2000) Ensemble methods in machine learning. In *Multiple Classifier Systems*, pp. 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Die00b] Dietterich T. G. (2000) Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pp. 1–15. Springer.

[FdRCH17] Fernández A., del Río S., Chawla N. V., and Herrera F. (Jun 2017) An insight into imbalanced big data classification: outcomes and challenges. *Complex & Intelligent Systems* 3(2): 105–120.

[FGG$^+$18] Fernández A., García S., Galar M., Prati R. C., Krawczyk B., and Herrera F. (2018) *Learning from imbalanced data sets*. Springer.

[FGHC18] Fernández A., Garcia S., Herrera F., and Chawla N. V. (2018) Smote for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. *Journal of artificial intelligence research* 61: 863–905.

[FHL14] Fan J., Han F., and Liu H. (2014) Challenges of big data analysis. *National science review* 1(2): 293–314.

[Fli19] Flink A. (2019) Apache Flink. `http://flink.apache.org/`.

[FM03] Fradkin D. and Madigan D. (2003) Experiments with random projections for machine learning. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 517–522. ACM.

[FS$^+$96] Freund Y., Schapire R. E., *et al.* (1996) Experiments with a new boosting algorithm. In *icml*, volumen 96, pp. 148–156. Citeseer.

[FS97] Freund Y. and Schapire R. E. (1997) A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences* 55(1): 119–139.

[FV14] Frénay B. and Verleysen M. (2014) Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems* 25(5): 845–869.

[GANAV14] Gonzalez-Abril L., Nuñez H., Angulo C., and Velasco F. (2014) GSVM: An SVM for handling imbalanced accuracy between classes inbi-classification problems. *Applied Soft Computing* 17: 23–31.

[GFB$^+$11] Galar M., Fernandez A., Barrenechea E., Bustince H., and Herrera F. (2011) A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42(4): 463–484.

[GLH15] García S., Luengo J., and Herrera F. (2015) *Data preprocessing in data mining*. Springer.

[GU16] Goldstein M. and Uchida S. (2016) A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one* 11(4): e0152173.

[Har75] Hartigan J. A. (1975) *Clustering algorithms.* Wiley series in probability and mathematical statistics. Wiley, New York, NY.

[HK19] Hueske F. and Kalavri V. (2019) *Stream Processing with Apache Flink: Fundamentals, Implementation, and Operation of Streaming Applications.* O'Reilly Media.

[HKZ$^+$15] Hamstra M., Karau H., Zaharia M., Konwinski A., and Wendell P. (2015) *Learning Spark: Lightning-Fast Big Data Analytics.* O'Reilly Media.

[HL05] Huang J. and Ling C. X. (2005) Using auc and accuracy in evaluating learning algorithms. *IEEE Transactions on knowledge and Data Engineering* 17(3): 299–310.

[HZW$^+$19] Hu J., Zhu E., Wang S., Liu X., Guo X., and Yin J. (2019) An efficient and robust unsupervised anomaly detection method using ensemble random projection in surveillance videos. *Sensors* 19(19).

[JL84] Johnson W. B. and Lindenstrauss J. (1984) Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics* 26(189-206): 1.

[Jol11] Jolliffe I. (2011) *Principal component analysis.* Springer.

[KMA$^+$19] Kairouz P., McMahan H. B., Avent B., Bellet A., Bennis M., Bhagoji A. N., Bonawitz K., Charles Z., Cormode G., Cummings R., *et al.* (2019) Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* .

[KMF$^+$17] Ke G., Meng Q., Finley T., Wang T., Chen W., Ma W., Ye Q., and Liu T.-Y. (2017) Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pp. 3146–3154.

[Kra16] Krawczyk B. (2016) Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence* 5(4): 221–232.

[KSW15] Krawczyk B., Schaefer G., and Woźniak M. (2015) A hybrid cost-sensitive ensemble for imbalanced breast thermogram classification. *Artificial intelligence in medicine* 65(3): 219–227.

[Kun14] Kuncheva L. I. (2014) *Combining pattern classifiers: methods and algorithms.* John Wiley & Sons.

[LGGRG$^+$20] Luengo J., García-Gil D., Ramírez-Gallego S., García S., and Herrera F. (2020) *Big Data Preprocessing. Enabling Smart Data.* Springer International Publishing. 978-3-030-39104-1.

[Lin13] Lin J. (2013) Mapreduce is good enough? if all you have is a hammer, throw away everything that's not a nail! *Big Data* 1(1): 28–37.

[LJX19] Li C., Jiang L., and Xu W. (2019) Noise correction to improve data and model quality for crowdsourcing. *Engineering Applications of Artificial Intelligence* 82: 184 – 191.

[LKRH15] Landset S., Khoshgoftaar T. M., Richter A. N., and Hasanin T. (2015) A survey of open source tools for machine learning with big data in the hadoop ecosystem. *Journal of Big Data* 2(1): 24.

[LS95] Liu H. and Setiono R. (1995) Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence*, pp. 388–391. IEEE.

[LSA$^+$18] Luengo J., Shim S.-O., Alshomrani S., Altalhi A., and Herrera F. (2018) Cnc-nos: Class noise cleaning by ensemble filtering and noise scoring. *Knowledge-Based Systems* 140: 27–49.

[LTHJ17] Lin W.-C., Tsai C.-F., Hu Y.-H., and Jhang J.-S. (2017) Clustering-based undersampling in class-imbalanced data. *Information Sciences* 409: 17–26.

[MBY$^+$16] Meng X., Bradley J., Yavuz B., Sparks E., Venkataraman S., Liu D., Freeman J., Tsai D., Amde M., Owen S., *et al.* (2016) Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research* 17(1): 1235–1241.

[MCJ$^+$19] Mao S., Chen J.-W., Jiao L., Gou S., and Wang R. (2019) Maximizing diversity by transformed ensemble learning. *Applied Soft Computing* 82: 105580.

[ML14] Maldonado S. and López J. (2014) Imbalanced data classification using second-order cone programming support vector machines. *Pattern Recognition* 47(5): 2070–2079.

[MMR$^+$17] McMahan B., Moore E., Ramage D., Hampson S., and y Arcas B. A. (2017) Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, pp. 1273–1282.

[NPF18] Nejatian S., Parvin H., and Faraji E. (2018) Using sub-sampling and ensemble clustering techniques to improve performance of imbalanced classification. *Neurocomputing* 276: 55–66.

[Pac19] Packages S. (2019) 3rd Party Spark Packages. https://spark-packages.org/.

[PGV$^+$18] Prokhorenkova L., Gusev G., Vorobev A., Dorogush A. V., and Gulin A. (2018) Catboost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, pp. 6638–6648.

[Pyl99] Pyle D. (1999) *Data Preparation for Data Mining.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

[RGFG$^+$18] Ramírez-Gallego S., Fernández A., García S., Chen M., and Herrera F. (2018) Big data: Tutorial and guidelines on information and process fusion for analytics algorithms with mapreduce. *Information Fusion* 42: 51–61.

[RGKG$^+$17] Ramírez-Gallego S., Krawczyk B., García S., Woźniak M., and Herrera F. (2017) A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing* 239: 39–57.

[RMBG18] Rao T. R., Mitra P., Bhatt R., and Goswami A. (2018) The big data system, components, tools, and technologies: a survey. *Knowledge and Information Systems* pp. 1–81.

[Rok10] Rokach L. (2010) *Pattern classification using ensemble methods*, volumen 75. World Scientific.

[RTR18] Rekha G., Tyagi A. K., and Reddy V. K. (2018) A novel approach to solve class imbalance problem using noise filter method. In *International Conference on Intelligent Systems Design and Applications*, pp. 486–496. Springer.

[SCZ+14] Shao Y.-H., Chen W.-J., Zhang J.-J., Wang Z., and Deng N.-Y. (2014) An efficient weighted lagrangian twin support vector machine for imbalanced data classification. *Pattern Recognition* 47(9): 3158–3167.

[SGLH16] Sáez J. A., Galar M., Luengo J., and Herrera F. (2016) INFFC: an iterative class noise filter based on the fusion of classifiers with noise sensitivity control. *Information Fusion* 27: 19–32.

[VVA03] Verbaeten S. and Van Assche A. (2003) Ensemble methods for noise elimination in classification problems. In *International workshop on multiple classifier systems*, pp. 317–325. Springer.

[WFHP16] Witten I. H., Frank E., Hall M. A., and Pal C. J. (2016) *Data Mining: Practical machine learning tools and techniques.* 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.

[Wol92] Wolpert D. H. (1992) Stacked generalization. *Neural networks* 5(2): 241–259.

[WZWD13] Wu X., Zhu X., Wu G.-Q., and Ding W. (2013) Data mining with big data. *IEEE transactions on knowledge and data engineering* 26(1): 97–107.

[ZCD+12] Zaharia M., Chowdhury M., Das T., Dave A., Ma J., McCauly M., Franklin M. J., Shenker S., and Stoica I. (2012) Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedins of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pp. 15–28. USENIX, San Jose, CA.

[Zer16] Zerhari B. (2016) Class noise elimination approach for large datasets based on a combination of classifiers. In *2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*, pp. 125–130. IEEE.

[Zho12] Zhou Z.-H. (2012) *Ensemble Methods: Foundations and Algorithms.* Chapman & Hall/CRC.

[ZW04] Zhu X. and Wu X. (2004) Class noise vs. attribute noise: A quantitative study. *Artificial intelligence review* 22(3): 177–210.

[ZWL+18] Zhu Z., Wang Z., Li D., Zhu Y., and Du W. (2018) Geometric structural ensemble learning for imbalanced problems. *IEEE Transactions on Cybernetics* pp. 1–13.

[ZWM14] Zaki M. J. and Wagner Meira J. (May 2014) *Data Mining and Analysis: Fundamental Concepts and Algorithms.* Cambridge University Press.

[ZZL16] Zhou Q., Zhou H., and Li T. (2016) Cost-sensitive feature selection using random forest: Selecting low-cost subsets of informative features. *Knowledge-based systems* 95: 1–11.