DOCTORAL THESIS

FAST K-NEAREST NEIGHBORS FOR BIG DATA AND SMART DATA

AUTHOR: Jesús Maillo

ADVISORS: Isaac Triguero

Francisco Herrera

UNIVERSIDAD DE GRANADA

University of Granada

Department of Computer Science
and Artificial Intelligence

PhD Program in Information and
Communication Technologies

PhD THESIS DISSERTATION

FAST K-NEAREST NEIGHBORS FOR BIG DATA AND SMART DATA

PhD STUDENT
JESÚS MAILLO

PhD ADVISORS
Isaac Triguero
Francisco Herrera

Granada, March 2020

Editor: Universidad de Granada. Tesis Doctorales
Autor: Jesús Maillo Hidalgo
ISBN: 978-84-1306-526-7
URI: http://hdl.handle.net/10481/62892

El doctorando / *The doctoral candidate* **Jesús Maillo** y los directores de la tesis / *and the thesis supervisors*: **Isaac Triguero** and **Francisco Herrera**

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

Guarantee, by signing this doctoral thesis, that the work has been done by the doctoral candidate under the direction of the thesis supervisors and, as far as our knowledge reaches, in the performance of the work, the rights of the other authors to be cited (when their results or publications have been used) have been respected.

Lugar y fecha / *Place and date*:

The PhD student:	The PhD advisor:	The PhD advisor:
MAILLO HIDALGO JESUS - 50617814N Firmado digitalmente por MAILLO HIDALGO JESUS - 50617814N Fecha: 2020.02.13 10:54:16 +01'00'	Isaac Triguero Digitally signed by Isaac Triguero DN: cn=Isaac Triguero, o=University of Nottingham, ou=School of Computer Science, email=Isaac.Triguero@nottingham.ac.uk, c=GB Date: 2020.02.13 12:14:17 Z	HERRERA TRIGUERO FRANCISCO - 26473617L Firmado digitalmente por HERRERA TRIGUERO FRANCISCO - 26473617L Fecha: 2020.02.20 09:25:50 +01'00'
Sgd.: Jesús Maillo	Sgd.: Isaac Triguero	Sgd.: Francisco Herrera

This doctoral thesis has been supported by the Spanish National Research Project TIN2017-89517-P. The PhD Student, J. Maillo, holds a FPU scholarship from the Spanish Ministry of Education FPU14/03081.

It's not so much staying alive,
it's staying human that's important.

George Orwell, 1984

The question of whether a computer can think
is no more interesting than the question of
whether a submarine can swim.

Edsger Dijkstra

Agradecimientos

Quiero comenzar agradeciendo a mis directores, Isaac y Paco, su esfuerzo y dedicación a la investigación es admirable. Gracias a ambos por el tiempo dedicado y el conocimiento que han compartido conmigo mediante innumerables consejos y reuniones, gracias a ellos concluyo esta etapa con éxito. Agradecer especialmente a Isaac su cercanía durante mi estancia en Nottingham.

También extiendo el agradecimiento a los compañeros del proyecto Repsol, dando un enfoque más práctico al trabajo realizado, y permitiéndome descubrir mi gusto por enfrentarme a problemas reales.

Agradecer a los compañeros y amigos de trabajo, por compartir el duro camino del doctorado y hacerlo más llevadero con el baloncesto, el rol, las tapas, y demás planes de ocio tan necesarios. Especialmente a Sergio, comenzamos el grado juntos, y siempre ha sido un apoyo en la formación, el trabajo y lo más importante, en lo personal. En la etapa de doctorado, se han disfrutado los largos debates sobre las labores de investigación, y aunque no tan agradables, se han hecho más llevaderas las horas compartidas entre las frías paredes del CPD.

Como no, agradecer a mis válvulas de escape del trabajo, queriendo destacar a Noemí, Carlos, Kako, Aitor y Lucia. Compartir tiempo con vosotros es un gusto y siempre me sabe a poco.

En último lugar, agradecer al apoyo más cálido y más necesario, mis dos pilares fundamentales. Noemí, sí otra vez, gracias por apoyar cada decisión y por estar tan cerca a pesar de la distancia que hay entre nosotros. Gracias a madre, padre y hermano por los valores que me han transmitido y la paciencia infinita que tienen. Sin estos dos pilares, el esfuerzo no tiene sentido, y este trabajo nunca se habría realizado.

A todos los que no he mencionado de forma explícita, no los estoy olvidando, este último párrafo es para todas aquellas personas que están en mi día a día, haciéndola más sencilla.

MUCHAS GRACIAS

Table of Contents

Chapter I

PhD dissertation

1 Introduction

Advances in technology in an increasingly digital world place data science at the forefront of extracting valuable knowledge in areas as diverse as medical applications [1], new challenges relating to social networks [2], implications for political elections [3] or theoretical physics studies developed at CERN [4], promoting the need to store as much data as possible. Thus, huge amounts of data are currently available from a wide variety of sources, exceeding standard storage and processing capacities, which is known as Big Data problems. This generates new challenges for researchers and companies, which are associated with problems of Volume, Velocity, Variety and Veracity, among others [5]. Note that the benefits are not found in the data itself, but in the capacity to extract patterns and information hidden in the data, in a process called Knowledge Discovery in Databases (KDD) [6]. In Big Data problems, it is commonly assumed that having bigger datasets will provide better performance of data mining techniques. However, conventional techniques are not able to handle so much data, affecting all stages of the KDD process [7].

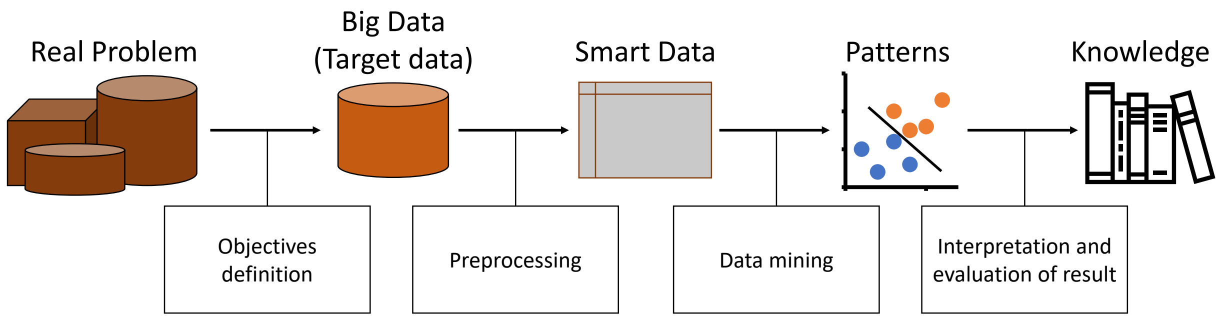


Figure 1: Stages of KDD process.

The KDD process is responsible for detecting hidden associations and patterns in the data, which provide useful knowledge. Although there are some differences in the definition of the stages of the KDD process [8], in this thesis we have adopted the one most widely used by the scientific community, illustrated in Figure 1 and composed of the following four stages:

- Objectives definition: selection of the data that constitutes the problem demanded by the end user, with the help of expert knowledge during data collection.

- Preprocessing: preparation of the data in order to facilitate the work in the following stages. Specifically, this stage deals with solving problems with integration, transformation, noise, missing values (MVs) and data reduction. For this purpose, the multiple data sources are unified into a single one and noise filtering, missing values imputation, and data reduction techniques are applied. Finally, after performing the different transformations, quality data is obtained and used in the next step.

- Data mining: first, the most appropriate area to address the problem must be identified, which could be association, clustering, regression or classification. Then, the technique is selected belonging to one of the previous families, and the corresponding algorithm is adapted by optimizing the parameters according to a procedure of validation. The algorithm extracts the valuable patterns that will be evaluated in the next stage.

- Interpretation and evaluation of result: the last stage is responsible to describe the patterns obtained, extracting valuable knowledge, so that they can be useful for the users.

This thesis focuses on two stages of the KDD process, when tackling Big Data problems. Big Data problems do not modify the KDD process, but they incorporate a number of difficulties that must be addressed and will be discussed later. Specifically, we focus on the data preprocessing [9] stage, with the purpose of obtaining quality data, and the data mining stage, with the objective of enabling the use of classification algorithms that obtain good results.

The literature refers to quality data as Smart Data [10], highlighting the importance of transforming raw data into quality data to facilitate the achievement of valuable knowledge and quality results [11]. The two major contributions of Smart Data are reducing storage and computation, and improving the performance of data mining techniques. Traditionally, the process of obtaining Smart Data is called preprocessing and although it is a less explored and known stage than data mining, it is usually the most time consuming stage of the KDD process.

Previously to the stages of preprocessing and data mining, it is necessary to know the maximum possible details of the problem to be faced. The most common is to perform an Exploratory Data Analysis (EDA) [12]. An EDA mainly consists of a study on basic statistics, such as mean, median, mode and their respective standard deviations, and also a comparison of the sample by means of inferential statistics and distribution charts for visual analysis. However, a good EDA includes any information that is useful, and it is common to introduce complexity metrics [13].

Emphasizing the importance of Smart Data and the need for data preprocessing, which is present in Big Data problems [14], the main problems to be faced are described below:

- Data preparation: the first step is to transform the real-world data into a computer-readable format [15, 16]. To do this, it is necessary to consider, among other things, the following problems: integration of the data, unifying several data sources in a common format. Detect and eliminate inconsistency, duplication and clean up the dataset from errors.

- Discretization: Its main objective is to transform continuous characteristics of a dataset into categorical characteristics, mainly by matching intervals of the continuous domain to a given group of categories. It is one of the main techniques for feature extraction, and therefore it is relevant in Big Data problems [17].

- Missing Values: Real data often contains MVs, usually caused by human or sensor mistakes in recording the values. MVs make some data mining techniques impossible to use, and those that can be applied reduce their effectiveness considerably. The most direct techniques include imputation using statistics such as the average value or the statistical mode [18], and the more sophisticated ones apply machine learning techniques to learn patterns about the features affected by MVs, to perform the imputation as a prediction [19].

- Noisy data: The existence of noise in the datasets is unavoidable, and affects the results obtained by data mining techniques. Noise detection is already a challenge, and once it is detected there are two main ways to reduce its impact. Through modification and cleaning of noisy data [20] or by designing new noise-robust data mining algorithms [21].
- Data reduction: reducing the dataset without losing information has a high impact on large datasets, alleviating the two main problems present in Big Data: runtime and storage. Reduction can be done in terms of the number of instances or features. As examples the frameworks for instance reduction [22] and feature selection [23].

Once the preprocessing stage is completed and Smart Data is obtained, a positive situation is created to facilitate the best performance of the data mining techniques [24], which are even more necessary in Big Data problems [25]. Depending on the objective variable, three families of techniques can be differentiated:

- Unsupervised learning: characterized by not having the target variable defined. Its purpose is to identify descriptive relationships implicit in the data and it can be differentiated into two problems:
  - Clustering [26]: detect groups of instances based on their similarity, maximizing the distance between groups and the cohesion between instances that belong to the same cluster.
  - Association [27]: identify relationships of interest between variables, such as correlation or independence between them.
- Supervised learning: the objective variable is perfectly defined. The purpose is to predict the value of the target variable for new unseen samples. To do so, the relationships between the input variables and the objective variable are learned, differentiating two branches of methods:
  - Classification [28]: the objective variable is discrete and the values it can take are known. These values are called labels or classes, and could be, for example, about a patient's health: whether he is healthy, or has diabetes of type 1, 2, gestational or other factors.
  - Regression [29]: the target variable is in continuous domain. An example of a regression problem would be the variation between two currencies over time, or the variation in a business' share price.
- Semi-supervised learning [30]: this is the intermediate case between those described above. The target variable is known for a reduced amount of data, and unknown for the remaining majority. In these problems, techniques must be applied that obtain valuable information from supervised and unsupervised learning. An example of a classic approach in this area would be self-labelling [31], starting from a small number of manually labelled examples, automating the labelling process in the remaining examples in the sample.

In this thesis we focus on classification problems. Classification techniques aim to predict an unseen instance in a particular, finite domain of known categories. Generally, a classifier learns a model with the information of the input dataset, called a training set. The model is then used to predict the value of the objective variable of an instance or set of instances not used in learning, called a test set.

The k Nearest Neighbors (kNN) [32] algorithm is recognized as one of the top ten most influential algorithms in data mining because of its effectiveness on classification and regression problems and its intuitive design [33]. In addition, it is used as the basis for numerous preprocessing techniques, making its sensitivity to noise a strength for detecting and eliminating noisy instances or applying data reduction techniques. The kNN belongs to the family of lazy learning algorithms, whose main distinctive feature is that it does not have a model training stage. Its workflow postpones all computation to the prediction stage. For this, it stores the training set, and when it has to predict a new example, it calculates the similarity with respect to the whole training set, selecting the k nearest instances. To decide on its prediction, in the case of classification it selects the most repeated class among the selected k samples, and for regression, it averages them.

There are numerous proposals to improve the runtime or accuracy of the kNN algorithm. Regarding the execution time, we can find two approaches: 1) Exact approach, that assures to find the nearest neighbors, where we can emphasize kd-tree [34] and metric-tree [35]. 2) Approximate approach, which reduces the execution time without ensuring to find the nearest neighbors, such as approximate nearest neighbors [36], locality sensitive hashing [37] or a variation of kd-tree called best bin first [38]. With regard to the improvement of accuracy, we can highlight the fuzzy variant, Fuzzy k Nearest Neighbor (FkNN) [39]. There are numerous proposals, however, the experiments carried out in [40], show how one of the most effective approaches is the original proposal FkNN. An evolutionary and fuzzy approach can be highlighted [41], which improves the effectiveness of FkNN, but increases the computation. FkNN consists of two stages: class membership and classification. FkNN consists of two stages: class membership and classification. The first stage changes the class label to a vector of class membership degree, according to the nearest training instances. The second stage calculates the kNN with the information incorporated in the first stage. Thus, class boundaries are detected more accurately, being less affected by noise and improving the kNN algorithm in most classification problems.

Despite the usefulness of the kNN algorithm in the KDD process, it encounters great difficulty in addressing problems with large amounts of data. The three challenges would be: 1) High computational cost, due to postpone all the computation to the classification stage, due to the no trained model. 2) Heavy demand for storage, in order to speed up the necessary computation, it is better to have the data in main memory. 3) Low robustness, kNN suffers from noise and badly defined boundaries.

The increase of available data and the need to process them to extract valuable knowledge through the KDD process, generates a series of problems in all its stages included under the name Big Data. Due to this need, cloud-based technologies [42] have been developed to alleviate and generate solutions to the required high computing and storage loads, as well as network traffic, or fault tolerance in computing machines. The most outstanding solution is MapReduce [43], compared to other distributed computing paradigms [44], where it shows a better scalability and more facilities for the development of distributed algorithms. There are numerous proposals for implementation of the MapReduce paradigm, the first open source implementation was Apache Hadoop [45], but currently Apache Spark [46] is recognized as more competent than Hadoop. Both use Hadoop's original distributed file system, called the Hadoop Distributed Files System (HDFS). The three most significant differences are improved main memory usage and the ability to iterate over the data, as well as a richer API that simplifies the work of the developer.

With the creation of the Hadoop and Spark frameworks, libraries of machine learning have been developed. We can highlight the Mahout [47] library for Hadoop, and MLlib [48] for Spark, which are supported by the creators of the respective frameworks. In addition, there are many external libraries: on the one hand, Spark-packages [49] is developed by the community and maintained by the company Databricks[1]. On the other hand, MMLSpark[50, 51] maintained and developed by the company Microsoft, both being open source.

In this thesis, we propose to address the following issues, keeping the focus on the relevance of the kNN algorithm in the KDD process, and its need in Big Data problems: 1) Enabling the kNN and FkNN classifiers to be able to scale and address Big Data problems. 2) The development of preprocessing techniques for obtaining Smart Data using the kNN algorithm. 3) Finally, two proposed metrics to study the complexity and density of Big Data problems.

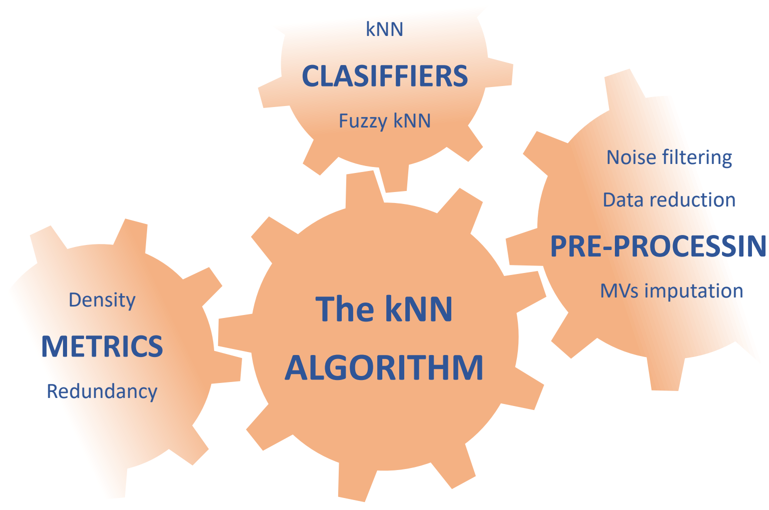


Figure 2: Objectives of the PhD thesis.

For the development of distributed algorithms under the MapReduce paradigm, two approaches from the divide and conquer paradigm [52] can be followed: local and global. The local approach does not consider the information belonging to each partition and machine in the cluster, nor is it considered a posteriori. This characteristic of the local approaches causes that the designed algorithms are obligatorily of approximate character. The global approach knows the information belonging to each partition. Despite having the ability to know all the samples, it does not imply that the algorithms designed are accurate, because they can use other techniques to speed up the computing and obtain approximate results.

At the time of writing this thesis, the proposals of kNN for Big Data are approximated by a local approach, which reduces the quality of the results obtained. For example, the algorithm proposed in [53] the athors apply k-means [54], the most well-known known similarity-based clustering algorithm, to separate the dataset and then calculates kNN by obtaining an approximate result according to the cluster to which it belongs. The algorithm iHMR-kNN [55], iteratively calculates kNN for image classification. There are also algorithms that use kNN internally, such as the Spitfire algorithm [56], used to perform queries on SQL databases. As far as FkNN as a classifier is concerned, the only proposal [57] is local and is limited to calculating FkNN on each

[1]Founded by the creators of Apache Spark. They provide distributed computing services through Microsoft Azure or Amazon Web Services. They organize courses on Big Data technologies

partition, obtaining very limited results.

The weakness of the kNN algorithm as a classifier, particularly the impact of data imperfections on it, becomes a strength if used for data preprocessing tasks. Specifically, there are models of instance reduction [58, 59, 60], feature selection [61, 62] for Big Data problems that base their core on the kNN algorithm, using it locally. However, although kNN has been used for the imputation of MVs [63] in relatively small to medium size problems, before this thesis and the proposal we made, no MVs imputation strategies had been developed based on that algorithm.

Although EDA is a field studied in Big Data problems [64, 65], there are no complexity, density or redundancy metrics that study specific Big Data problems. With the metrics it is possible to characterize the datasets and obtain a greater knowledge of the problem, detecting which techniques will have a more favorable behavior. In this thesis we propose specific metrics for Big Data problems and study some of the most used datasets in the research field.

Finally, after the introduction presented in this section, the current thesis is composed of two parts: the doctoral thesis and the publications that constitute it. In the first part, the Section 2 describes the knowledge that forms the necessary background for the elaboration of the thesis: starting with Big Data (Subsection 2.1), continuing with the kNN algorithm (Subsection 2.2), the FkNN algorithm (Subsection 2.3), the preprocessing techniques to obtain Smart Data based on kNN (Subsection 2.4) and finally the complexity and redundancy metrics (Subsection 2.5). Section 3 presents the justification for the work performed, describing the open problems that have been faced. Section 4 details the objectives set to address the problems described. Section5 shows the methodology followed for the development of the thesis. Section 6 summarizes the work that composed this report and then Section 7 shows the results achieved. Finally, the Section 8 compiles the main conclusions reached and highlights the future lines.

The second part of the present report consists of 3 publications in international and indexed journals and one work that is under review. The publications ordered according to the objectives are the following:

- kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for Big Data.
- Fast and Scalable Approaches to Accelerate the Fuzzy k Nearest Neighbors Classifier for Big Data. IEEE Transactions on Fuzzy Systems.
- Transforming Big Data into Smart Data: An insight on the use of the k nearest neighbors algorithm to obtain quality data.
- Redundancy and Complexity Metrics for Big Data Classification: Towards Smart Data.

Introducción

Los avances de la tecnología en un mundo cada vez más digitalizado sitúan la ciencia de datos en la vanguardia de extracción de conocimiento valioso en áreas tan diversas como aplicaciones médicas [1], nuevos retos relativos a redes sociales [2], implicaciones en elecciones políticas [3] o estudios de física teórica desarrollados en el CERN [4], promoviendo la necesidad de almacenar tantos datos como sea posible. Así, actualmente se disponen de ingentes cantidades de datos que provienen de fuentes muy diversas y superan las capacidades de almacenamiento y procesamiento convencionales, recibiendo el nombre de problemas *Big Data*. Esto genera nuevos retos para investigadores y empresas, los cuales están asociados a problemas de Volumen, Velocidad, Variedad y Veracidad entre otros [5]. Cabe destacar, que los beneficios no se encuentran en los datos por sí solos, sino en la capacidad de extraer patrones e información oculta en los datos, en un proceso que recibe el nombre de descubrimiento del conocimiento en base de datos (*Knowledge Discovery in Databases* - KDD) [6]. En los problemas Big Data, se asume que tener un número elevado de datos favorece un mejor rendimiento de las técnicas de minería de datos. Sin embargo, las técnicas clásicas no son capaces de manejar tantos datos, afectando a todas las etapas del proceso KDD [7].

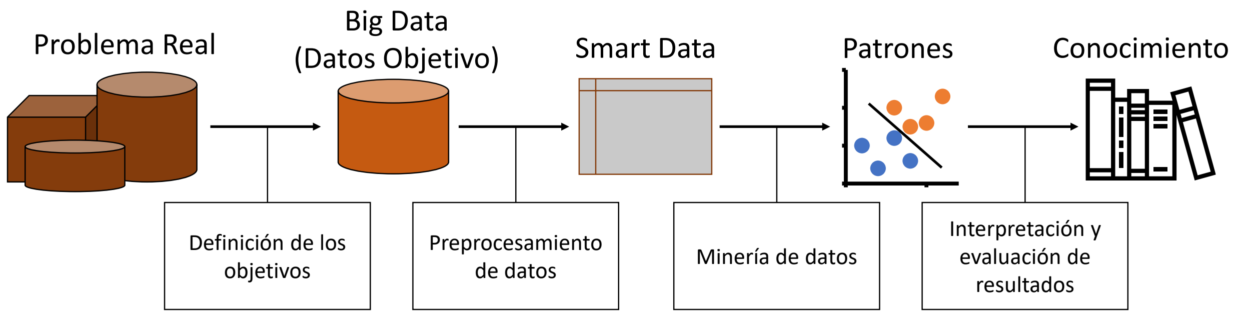

Problema Real
Big Data (Datos Objetivo)
Smart Data
Patrones
Conocimiento
Definición de los objetivos
Preprocesamiento de datos
Minería de datos
Interpretación y evaluación de resultados

Figura 3: Etapas del proceso KDD.

El proceso KDD es responsable de detectar relaciones y patrones ocultos en los datos, que aporten conocimiento de utilidad. Aunque existen algunas diferencias en cuanto a la definición de las etapas del proceso KDD [8], en esta tesis hemos tratado de adoptar la más extendida por la comunidad científica, ilustrada en la Figura 3 y compuesta por las siguientes cuatro etapas:

- Definición de los objetivos: selección de los datos que constituyen el problema demandado por el usuario final, con la ayuda del conocimiento experto durante la recogida de los datos.
- Preprocesamiento de datos: preparación de los datos con el objetivo de facilitar el trabajo en las siguientes etapas. Concretamente, esta etapa trata de solucionar problemas con integración, transformación, ruido, MVs y reducción de datos principalmente. Para ello, se unifican las múltiples fuentes de datos en una sola y se aplican técnicas de limpieza de ruido, imputación de MVs, y reducción de datos. Finalmente, tras aplicar las distintas transformaciones se obtienen datos de calidad que pasarán a utilizarse en la siguiente etapa.
- Minería de datos: en primer lugar, se debe identificar el área más adecuada para afrontar el problema, que podría ser asociación, clustering, regresión o clasificación. A continuación, se selecciona la técnica perteneciente a alguna de las familias anteriores, y se adaptará el algoritmo en cuestión optimizando los parámetros de acuerdo a un procedimiento de validación. El algoritmo extrae los patrones válidos que serán evaluados en la siguiente etapa.

- Interpretación y evaluación de resultados: la última etapa se encarga de describir los patrones obtenidos, extrayendo conocimiento de valor, de forma que puedan ser útiles para los usuarios.

Esta tesis se centra en dos etapas del proceso KDD, con el añadido de tratar problemas Big Data. Los problemas Big Data no modifican el proceso KDD, pero si incorporan una serie de dificultades que deben ser tratadas y serán expuestas más adelante. Concretamente, se centra en la etapa de preprocesamiento de datos [9] con el objetivo de obtener datos de calidad y la etapa minería de datos, con el objetivo de habilitar el uso de algoritmos de clasificación que obtengan buenos resultados.

La literatura denomina a estos datos de calidad *Smart Data* [10], fijando en ellos la importancia de la transformación de datos crudos en datos de calidad para facilitar la obtención de conocimiento valioso y resultados de calidad [11]. Las dos grandes contribuciones de Smart Data son la disminución de almacenamiento y cómputo, y la mejora del rendimiento de las técnicas de minería de datos. Tradicionalmente, el proceso de obtención de Smart Data recibe el nombre de preprocesamiento y aunque es una etapa menos explorada y conocida que la minería de datos, es usual que sea la etapa que más tiempo consuma del proceso KDD.

Previo a las etapas de preprocesamiento y minería de datos, es necesario conocer los máximos detalles posibles del problema que tenemos que afrontar. Lo más común es realizar un análisis exploratorio de datos [12] (*Exploratory Data Analysis* - EDA). Un EDA se compone principalmente de un estudio sobre estadísticos básicos, como la media, mediana, moda y sus respectivas desviaciones típicas, y también de una comparación de la muestra mediante estadística inferencial y gráficas de distribución para un análisis visual. Sin embargo, un buen EDA incluye toda información que sea de utilidad, siendo habitual introducir métricas de complejidad [13].

Remarcada la importancia de Smart Data y la necesidad del preprocesamiento de datos, encontrándose presente en problemas Big Data [14], a continuación se describen los principales problemas a enfrentar:

- Preparación de los datos: el primer paso consiste en convertir los datos del mundo real en un formato legible para el ordenador [15, 16]. Para ello, es necesario considerar, entre otros, los siguientes problemas: integración de los datos, unificando varias fuentes de datos en un formato común. Detectar y eliminar la inconsistencia, duplicidad y limpiar el conjunto de datos de errores.

- Discretización: Su principal objetivo es transformar características continuas de un conjunto de datos, en características categóricas, principalmente relacionando intervalos del dominio continuo a un conjunto determinado de categorías. Es una de las principales técnicas de extracción de características, y por ello es relevante en problemas Big Data [17].

- Valores perdidos: los datos de problemas reales suelen contener valores perdidos (*Missing Values - MVs*), provocados generalmente por errores humanos o de sensores a la hora de anotar los valores. Los MVs imposibilitan la utilización de algunas técnicas de minería de datos, y aquellos que pueden aplicarse reducen considerablemente su eficacia. Las técnicas más directas incluyen la imputación mediante estadísticos como el valor promedio o la moda [18], y las más sofisticadas aplican técnicas de machine learning para aprender patrones sobre las características afectadas con MVs, para realizar la imputación como se realizaría una predicción [19].

- Datos ruidosos: la existencia de ruido en los conjuntos de datos es inevitable, y afecta a los resultados obtenidos por las técnicas de minería de datos. Detectar el ruido ya es una tarea compleja, y una vez detectado existen dos formas principales de reducir su impacto. Mediante la modificación y limpieza de los datos ruidosos [20] o con el diseño de nuevos algoritmos de minería de datos robustos al ruido [21].

- Reducción de datos: reducir el conjunto de datos sin perder información tiene un alto impacto en grandes conjuntos de datos, aliviando los dos principales problemas presentes en Big Data: el tiempo de ejecución y el almacenamiento. En cuanto a la reducción, se puede realizar respecto al número de instancias o de características, como ejemplos los framework de reducción de prototipos [22] y selección de características [23].

Tras la etapa de preprocesamiento y la obtención de Smart Data, se consigue generar una situación favorable para facilitar el mejor funcionamiento de las técnicas de minería de datos [24], aún más necesarias en problemas Big Data [25]. Con dependencia de la variable objetivo se pueden diferenciar tres familias de técnicas:

- Aprendizaje no supervisado: se caracteriza por no tener definida la variable objetivo. Su propósito es identificar relaciones descriptivas implícitas en los datos y pueden diferenciarse dos problemas:
  - Agrupamiento [26]: detectar grupos de instancias basadas en su similitud, maximizando la distancia entre grupos y la cohesión entre instancias que pertenecen al mismo grupo.
  - Asociación [27]: identificar relaciones de interés entre variables, como correlación o independencia entre las mismas.
- Aprendizaje supervisado: la variable objetivo está perfectamente definida. La finalidad es predecir el valor de la variable objetivo para nuevas muestras no vistas. Para ello, se aprenden las relaciones entre las variables de entrada y la variable objetivo, diferenciando dos ramas de métodos:
  - Clasificación [28]: la variable objetivo es discreta y los valores que puede tomar son conocidos. Estos valores reciben el nombre de etiquetas o clases, y podrían ser por ejemplo, sobre la salud de un paciente, si está sano, o posee diabetes de tipo 1, 2, gestacional u otros factores.
  - Regresión [29]: la variable objetivo se encuentra en un dominio continuo. Un ejemplo de problema de regresión sería, la variación del valor entre dos divisas en el tiempo, o la variación del precio de las acciones de una empresa.
- Aprendizaje semi-supervisado [30]: es el caso intermedio entre los descritos anteriormente. La variable objetivo es conocida para una parte reducida de los datos, y desconocida para la mayoría restante. En estos problemas se deben aplicar técnicas que obtengan información valiosa del aprendizaje supervisado y el no supervisado. Un ejemplo de problema clásico de este área sería el auto-etiquetado [31], partiendo de un número reducido de ejemplos etiquetados manualmente, automatizar el proceso de etiquetado en el resto de ejemplos de la muestra.

En esta tesis nos centramos en problemas de clasificación. Las técnicas de clasificación tienen como objetivo predecir una instancia no vista dentro de un dominio determinado y finito de categorías conocidas. Generalmente, un clasificador aprende un modelo con la información del conjunto de entrada, nombrado conjunto de entrenamiento. Posteriormente, se utiliza el modelo para predecir el valor de la variable objetivo de una instancia o conjunto de instancias no utilizadas en el aprendizaje, denominado conjunto de prueba.

El algoritmo de los k vecinos más cercanos (*k-Nearest Neighbors* - kNN) [32] está reconocido como uno de los diez algoritmos más influyentes de la minería de datos [33], debido a su efectividad en problemas de clasificación, regresión y su diseño intuitivo. Además, es utilizado como base de numerosas técnicas de preprocesamiento, convirtiendo su sensibilidad al ruido en una fortaleza para detectar y eliminar instancias ruidosas o aplicar técnicas de reducción de datos. El kNN pertenece a la familia de algoritmos de aprendizaje perezoso, cuya mayor singularidad es que no poseen una etapa de entrenamiento de modelo. Su flujo de trabajo pospone todo el computo a la etapa de predicción. Para ello, almacena el conjunto de entrenamiento, y cuando tiene que predecir un nuevo ejemplo, calcula la similitud con respecto a todo el conjunto de entrenamiento, seleccionando las k instancias más cercanas. Para decidir su predicción, en el caso de clasificación se selecciona la clase más repetida entre las k muestras seleccionadas, y para regresión, se realiza el promedio de las mismas.

Son numerosas las propuestas para mejorar el tiempo de ejecución o la precisión del algoritmo kNN. Respecto al tiempo de ejecución, podemos encontrar dos enfoques: 1) Enfoque exacto, que asegura encontrar los vecinos más cercanos, dónde se pueden destacar kd-tree [34] y metric-tree [35]. 2) Enfoque aproximado, reduce el tiempo de ejecución sin asegurar encontrar los vecinos más cercanos, como podrían ser *approximate nearest neighbors* [36], *locality sensitive hashing* [37] o una variación de kd-tree denominada *best bin first* [38]. Respecto a la mejora de precisión, podemos destacar la variante difusa (*Fuzzy k Nearest Neighbor* - FkNN) [39]. Existen numerosas propuestas, sin embargo, los experimentos realizados en [40], muestran como uno de los enfoques más efectivos es la propuesta original FkNN. Se puede destacar una aproximación evolutiva y difusa [41] que mejora la eficacia de FkNN, pero incrementando el computo. FkNN está formado por dos etapas: grado de pertenencia a clase y clasificación. La primera etapa cambia la etiqueta de clase por un vector de grado de pertenencia a cada clase, de acuerdo a las instancias de entrenamiento más cercanas. La segunda etapa calcula los kNN con la información incorporada en la primera etapa. Así, se detectan las fronteras de clase con mayor precisión, viéndose menos afectado por el ruido y mejorando al algoritmo de clasificación kNN en la mayoría de los problemas.

A pesar de la utilidad del algoritmo kNN en el proceso KDD, se encuentra con grandes dificultades al abordar problemas con grandes cantidades de datos. Los tres retos serían: 1) Elevado coste computacional, debido a posponer todo el cómputo a la etapa de clasificación, por la falta de un modelo entrenado. 2) Alta necesidad de almacenamiento, para agilizar el cómputo necesario, es mejor disponer de los datos en memoria principal. 3) Poca robustez, kNN sufre de ruido y fronteras mal definidas.

El aumento de los datos disponibles y la necesidad de procesarlos para extraer conocimiento de valor mediante el proceso KDD, genera una serie de problemas en todas sus etapas englobados bajo el nombre Big Data. Debido a esta necesidad, se han desarrollado tecnologías basadas en cloud [42] para aliviar y generar soluciones a las altas cargas de cómputo y almacenamiento necesarias, así como el tráfico en red, o la tolerancia a errores en las máquinas de cómputo. La solución más destacada es MapReduce [43], comparado con otros paradigmas de computación distribuida [44], donde destaca con una mejor escalabilidad y mayor facilidades para el desarrollo de algoritmo distribuidos. Existen numerosas propuestas de implementación del paradigma MapReduce, la primera implementación de código abierto fue Apache Hadoop [45], pero actualmente Apache Spark [46] es reconocido como más competente que Hadoop. Ambos utilizando el sistema de archivos distribuido original de Hadoop, denominado *Hadoop Distributed Files System* (HDFS). Las tres diferencias más significativas son la mejora del uso de memoria principal y la capacidad de iterar sobre los datos, así como una API más rica que facilita el trabajo del desarrollador.

Con la creación de los framework Hadoop y Spark, se han desarrollado bibliotecas de minería de datos. Se pueden destacar las bibliotecas Mahout [47] para Hadoop, y MLlib [48] para Spark, que cuentan con soporte de los creadores de los respectivos framework. Además, existen multitud de bibliotecas externas: por un lado, Spark-packages [49] es desarrollada por la comunidad y mantenida por la empresa Databricks[2], por otro lado, MMLSpark[50, 51] mantenida y desarrollada por la empresa Microsoft, siendo ambos casos de código abierto.

En esta tesis se plantea abordar los siguientes temas, manteniendo el foco de atención en la relevancia del algoritmo kNN en el proceso KDD, y su necesidad en problemas Big Data: 1) Habilitación de los clasificadores kNN y FkNN para ser capaz de escalar y abordar problemas Big Data. 2) El desarrollo de técnicas de preprocesamiento para la obtención de Smart Data mediante la utilización del algoritmo kNN. 3) Finalmente, dos propuestas de métricas para estudiar la complejidad y densidad de problemas Big Data.

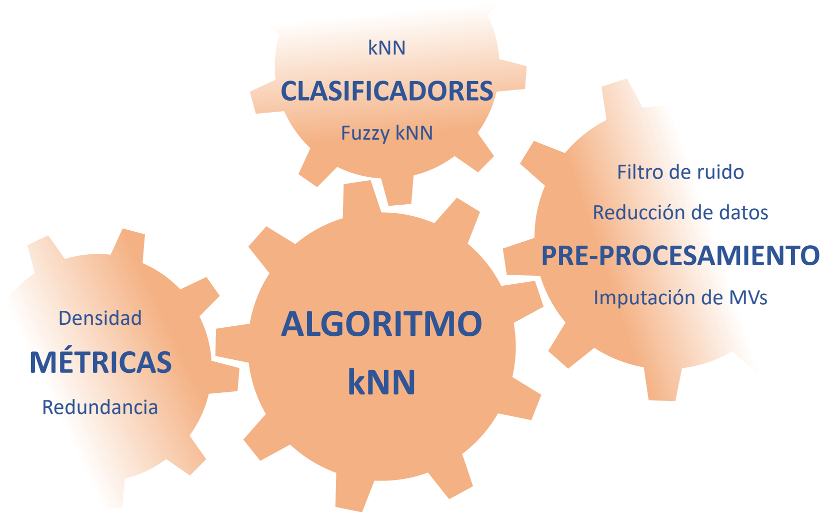


Figura 4: Objetivos de la tesis doctoral.

Para el desarrollo de algoritmos distribuidos bajo el paradigma MapReduce, se pueden seguir dos enfoques provenientes del paradigma divide y vencerás [52]: local y global. El enfoque local no actúa con conocimiento sobre la información perteneciente a cada partición y máquina del cluster, ni tampoco es considerada a posteriori. Esta característica de los enfoques locales provoca que los algoritmos diseñados sean obligatoriamente de carácter aproximado. El enfoque global si conoce la información perteneciente a cada partición. A pesar de tener la capacidad de conocer todas las muestras, no implica que los algoritmos diseñados sean exactos, pues pueden utilizar otras técnicas para agilizar el cómputo y obtener resultados aproximados.

Hasta la escritura de esta tesis, las propuestas de kNN para Big Data son aproximadas mediante un enfoque local, lo que reduce la calidad de los resultados obtenidos. Por ejemplo, el algoritmo propuesto en [53] aplica k-means [54], el algoritmo más conocido de agrupamiento basado en similitud, para separar el conjunto de datos, y posteriormente calcula kNN obteniendo un resultado aproximado según el cluster al que pertenece. El algoritmo *iHMR-kNN* [55], calcula iterativamente kNN para clasificación de imagenes. También existen algoritmos que utilizan kNN de forma interna, como el algoritmo *Spitfire* [56], utilizado para realizar consultas en bases de datos

[2]Fundada por los creadores de Apache Spark. Proveen servicio de cómputo distribuido mediante Microsoft Azure o Amazon Web Services. Organizan cursos sobre tecnologías Big Data.

SQL. En lo que respecta a FkNN como clasificador, la única propuesta [57] es local y está limitada a calcular FkNN en cada partición, obteniendo resultados muy limitados.

La debilidad del algoritmo kNN como clasificador, concretamente el impacto que tienen en él las imperfecciones de los datos, se convierte en una fortaleza si se utiliza para tareas de preprocesamiento de datos. Específicamente, existen modelos de reducción de instancias [58, 59, 60], selección de características [61, 62] para problemas Big Data que basan su núcleo en el algoritmo kNN, utilizándolo de forma local. Sin embargo, a pesar de que kNN ha sido utilizado para la imputación de MVs [63] en problemas de tamaño clásico, antes de esta tesis y la propuesta que realizamos en ella, no se habían desarrollado estrategias de imputación de MVs basados en dicho algoritmo.

En último lugar, aunque el EDA si es un campo estudiado en problemas Big Data [64, 65], no existen métricas de complejidad, densidad o redundancia que estudien los problemas específicos de Big Data. Con las métricas se consigue caracterizar los conjuntos de datos y se obtiene un mayor conocimiento del problema, detectando que técnicas tendrán un comportamiento más favorable. En esta tesis proponemos métricas específicas para problemas Big Data y estudiamos algunos de los conjuntos de datos más utilizados en el ámbito investigador.

Finalmente, tras la introducción planteada en esta sección, la presente tesis está compuesta de dos partes: la tesis doctoral y las publicaciones que la forman. En la primera parte, la Sección 2 describe en detalle los conocimientos que forman la base necesaria para el desarrollo de la tesis: comenzando por Big Data (Subsección 2.1), continuando con el algoritmo kNN (Subsección 2.2), el algoritmo FkNN (Subsección 2.3), las técnicas de preprocesamiento para obtener Smart Data basadas en kNN (Subsección 2.2) y finalmente las métricas de complejidad y redundancia (Subsección 2.5). La Sección 3 presenta la justificación del trabajo llevado a cabo, describiendo los problemas abiertos que han sido enfrentados. La Sección 4 detalla los objetivos fijados para abordar los problemas descritos. La Sección 5 muestra la metodología seguida para el desarrollo de la tesis. La Seccion 6 resume los trabajos que componen esta memoria y posteriormente la Sección 7 muestra los resultados alcanzados. Finalmente, la Sección 8 recopila las principales conclusiones alcanzadas y destaca las líneas futuras.

La segunda parte de la memoria lo conforman 3 publicaciones en revista internacionales e indexadas y una publicación que se encuentra en el proceso de revisión. Las publicaciones ordenadas según los objetivos son las siguientes:

- kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for Big Data.
- Fast and Scalable Approaches to Accelerate the Fuzzy k Nearest Neighbors Classifier for Big Data. IEEE Transactions on Fuzzy Systems.
- Transforming Big Data into Smart Data: An insight on the use of the k nearest neighbors algorithm to obtain quality data.
- Redundancy and Complexity Metrics for Big Data Classification: Towards Smart Data.

2 Preliminaries

This section describe in detail the main concepts involved in this thesis. First of all, Section 2.1 describes the Big Data problem, including the most popular paradigm and tools. Sections 2.2 and 2.3 expose the kNN and FkNN algorithms respectively, and outline the main problems to scale those algorithm to tackle big datasets. Section 2.4 presents the different groups of data preprocessing methods, focusing to MVs. Finally, Section 2.5 gives a brief overview of the different families of complexity metrics, paying attention to the overlapping measures and the class balance measure.

2.1 Big Data technologies: MapReduce and Spark

We are in the era of information and data, where the companies with the highest market value in the world generate their fortunes around technology and information, highlighting the first three in 2019[3]: Microsoft, Apple and Amazon. Most of the information is generated through the Internet[4], where an estimated 56.1% of the world's population (4.39 billion people), actively contributes. For this reason, the ability to make data-driven decisions is important to any business or government.

The group of technologies, infrastructures, techniques and algorithms specifically designed to store and process such massive amounts of data, are collected under the term Big Data. One of the most common definitions of Big Data is based on 3 Vs [5]: Volume, Velocity, Variety. Nevertheless, it has been extended to 7 [66], outstanding Value for its importance. Value refers to the improvement that can be obtained by addressing the problem, the extraction of valuable information.

Distributed computing approaches become mandatory if we need to work with large amounts of data. A single-node architecture is not capable of storing and processing such quantities. In the beginning of Big Data, the MapReduce paradigm (MR) [43] was used to process PageRank [67], an algorithm in charge of classifying the pages searched in Google. After that, Doug Cutting developed an open source implementation called Hadoop [45]. MR was designed to facilitate the use and development of distributed techniques across a cluster of computer machines. To that end, MR provides in a transparent way, distributed computing, fault tolerance, automatic job-resource scheduling, automatic data partition and management. To obtain the benefit of using MR, the algorithms developed have to follow a workflow represented in the Figure 5.

The MR scheme can be described according to the following three steps:

- Map: the data is first read and formatted in key-value pairs. They are partitioned and distributed among the computer machines that constitute the cluster. Specific processing is applied to each data partition on each machine, and finally the data is transmitted via network to the computer machines responsible for the Reduce stage.
- Shuffle: groups the values associated with the same key, matching on the same machine those values that are labelled with the identical key.
- Reduce: finally, the merging of the values with the same key is computed using some specific and problem-dependent technique. As a good practice, use a reduction technique as the last step of the Map stage, to reduce the network traffic generated and group the least possible pairs key-value pairs.

[3]The World's Largest Public Companies - `https://www.forbes.com/global2000/list/#header:marketValue_sortreverse:true`

[4]Data Never Sleeps 7.0 - `https://www.domo.com/learn/data-never-sleeps-7`

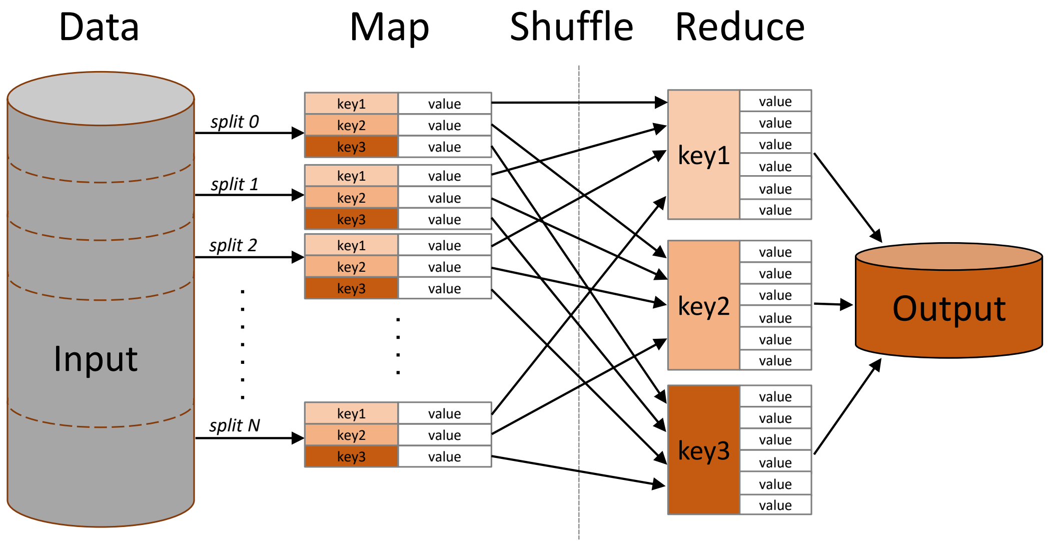


Figure 5: Workflow overview of MapReduce

Despite the great popularity that Hadoop reached in its beginnings, the emergence of new implementations of the MR paradigm has brought numerous criticisms to Hadoop. Its main weaknesses are data streaming, iterative processing or the use of hard disk during computing. For this reason, currently the best known is Spark [46], which provides a solution to the problems presented in Hadoop, and provides an API with greater functionality, with a new data structure called Resilient Distributed Datasets (RDDs). RDDs are immutable, can be stored in main memory or on disk, for fast access or greater capacity respectively. They can also be re-used to iterate over the data and apply multiple stage maps and reductions.

In relation to this thesis, it is necessary to highlight the Spark machine learning libraries, called MLlib [48]. The purpose of MLlib is to provide native functionalities and algorithms, which consider, the statistics calculation, and algorithm implementations in classification, regression, clustering, dimensionality reduction, among others. However, despite the breadth of the library, they do not have the kNN algorithm, and therefore do not have kNN-based preprocessing algorithms available either. It is remarkable that neither it has any specific complexity metric for Big Data problems.

2.2 The k Nearest Neighbors Classifier

As mentioned in the Introduction Section, kNN [32] stands out as one of the top ten most influential algorithms in data mining [33]. Its recognition comes from its good performance as a classification and regression algorithm, and from the simplicity of its design. In addition, it is used in a multitude of real problems, such as in cancer classification gene expression [68], traffic prediction [69], or for intrusion detection in cybersecurity systems [70], among others.

A large number of preprocessing techniques have the kNN algorithm as the core. Among other algorithms, can be listed: the SMOTE [71] algorithm for class balancing throught oversampling, the FCNN [72] algorithm for noise filtering or the kNN-I [63] algorithm for imputation of MVs.

The kNN is a non-parametric algorithm, and has the particularity of not having a training stage. Instead, it stores the instances of the training set and postpones all the computation to the classification stage. When it arrives at unseen case to classify, it calculates the similarity [73] or distance [74] with each sample of the training set. The similarity calculation is usually obtained with the Euclidean distance for real variables, Hamming for integer variables. Then, sort according to similarity and select the nearest k instances. With these samples, a majority vote is taken to decide which will be the predicted class.

An example of the classification of a new unseen case is shown in Figure reffig:kNN. Specifically, it shows a 2D example taking the Euclidean distance as a measure of similarity. If we set the value of k to 3, it is classified as class B, with the result expanded: 2 votes for class B and 1 for class A.

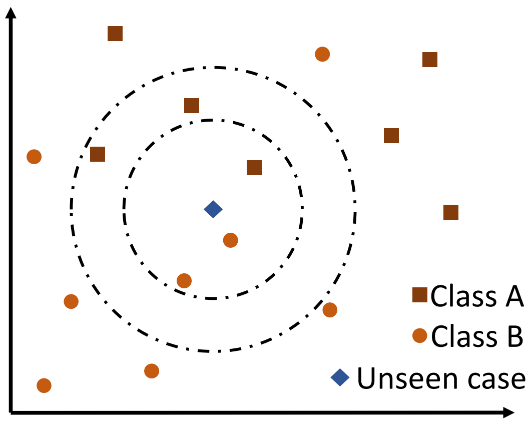


Figure 6: 2D classification example with the kNN algorithm. k = 3 and 5

Despite the good performance and efficiency of the kNN algorithm in a wide variety of problems, it encounters scalability problems in dealing with large-scale data. These problems come from its lazy behavior, that leads to storing all the instances of the training set and having a high computer load in the classification stage.

- Runtime: the computational complexity for calculating the nearest neighbor is $O((I \cdot F))$, where I is the number of instances that compose the training set and F is the number of features. This process is repeated for each instance that we have to sort. Furthermore, if we increase the value of k, the computational complexity of sorting distances is added, which implies an extra $O(n \cdot log(n))$.
- Memory consumption: to speed up the runtime of the algorithm, it is necessary to have the training and test set in the main memory.Thus, avoiding as much as possible the swap between hard disk and RAM memory. For big datasets, it may be easy to overcome the main memory available in a single machine.

To alleviate these problems in the development of the algorithm, there are two approaches that can be followed: exact and approximate. The approximate approach speeds up the execution time and sacrifices the quality, for that it reduces the computation of ensuring the obtaining of the nearest instance. The exact proposals spend time to ensure that the nearest instance selected, is indeed.

Among the exact proposals, we can highlight kd-tree [34] and metric-tree [35], both proposals build a tree to speed up the search time. However, the size of the tree is dependent on the number of features in the dataset, and this disables its scalability in large datasets with a moderate number of features.

Among the approximate algorithms we can highlight, the approximate nearest neighbors algorithm [36] and the locality sensitive hashing algorithm [37], among other. Both are sequential designs and although they speed up the computation in traditional datasets, they are not capable of scaling to big datasets because they are not distributed designs. The algorithm presented in [55] proposes iteratively repeating a map phase process, without an explicitly defined reduce phase, which is very time consuming repeated for each test sample. The approximate proposal that stands out from the others is Hybrid-spill tree [75], which combines the exact search of metric-tree, with the approximate search of spill-tree [76]. In this way, they achieve a balance in execution time and accuracy of high quality, capable of scaling in large datasets.

2.3 The Fuzzy k Nearest Neighbors Classifier

The FkNN [39] algorithm stands out as one of the most effective according to the experimental study conducted by Derrac et al. [40]. Although there are proposals that improve its performance, such as the evolutionary and diffuse algorithm proposed in [41], they also increase its computational complexity. For this reason, we focused on FkNN to be able to perform Big Data problems.

The FkNN algorithm classifies based on similarity, just like its kNN parent. However, its workflow is composed of two stages. The first stage is to change the class label to the class membership degree. To do this, it calculates for each instance of the training set its k nearest samples maintaining a leave-one-out scheme and computes the equation I.1. The result of this first stage is the training set by changing the class label to a vector class membership degree of each class. We name this new training set Fuzzy Training Set.

$$u_j(x) = \begin{cases} 0.51 + (n_j/k) \cdot 0.49 & if \quad j = i \\ \\ (n_j/k) \cdot 0.49 & if \quad j \neq i \end{cases} \tag{I.1}$$

where: n_j is the number of neighbors of the class j, and if the class matches the one owned by the instance to classify n_i, 0.51 will be added to enhance the original class.

In the second stage its kNN is calculated on the Fuzyz Training Set. Then, it adds the results of the candidates according to the Equation I.2. The predicted class is the one with the highest membership value.

$$u_i(x) = \frac{\sum_{j=1}^{K} u_{ij}(1/|x - x_j|^{2/(m-1)})}{\sum_{j=1}^{K}(1/|x - x_j|^{2/(m-1)})} \tag{I.2}$$

where m is a parameter to adjust the influence of the neighbors with respect to the inverse of the distance. If m is 2, each neighbors is weighted by the reciprocal of the distance. If m is close to 1, the closer neighbors have a much greater influence than the more distant neighbors. If we increase m, the neighbors affect more uniformly.

The first stage of the FkNN algorithm, which is an extra stage compared to the kNN, increases computational complexity and generates the computational bottleneck. The computational complexity is equal to the magnitude order of the kNN algorithm, however, the first stage is about the training set and this is usually much larger than the test set. For this same reason, the main memory needed to store the entire dataset is even more demanding than in the kNN.

To the best of our knowledge, the only proposal to apply FkNN to Big Data problems is proposed by Bakry et al. [77]. The authors proposes to apply a MapReduce scheme, in which

FkNN is computed in each partition, losing the information of all those instances that do not share a partition. Later, it groups the class label in a reduce phase, to choose the predicted class by majority vote. It is a very simplistic approach, in which any classification method can be applied and that affects negatively the results.

2.4 kNN-based preprocesing algorithms to obtain Smart Data

The most important step in the KDD process for quality results is data preprocessing. Since raw data is obtained, there are different imperfections in the data that make it difficult to obtain patterns and knowledge in the data mining step. Quality data obtained after the data preprocessing stage is known as Smart Data [10].

This thesis is focused on the ubiquity of the kNN algorithm in the whole KDD process, and therefore, we are going to focus on those kNN-based preprocessing techniques for obtaining Smart Data:

- Missing Values: the existence of MVs reduces the efficiency of data mining methods, becoming impossible to apply some algorithms. The most prominent method for kNN-based MVs imputation is called kNN-Imputation [63], which takes the neighborhood given a value for the parameter k to perform the imputation, using the mean if it is a numerical value, or the statistical mode if it is a categorical value.
- Data Reduction: the purpose of data reduction techniques is to decrease the size of the dataset to alleviate data storage and processing requirements. At the same time, noisy, redundant or irrelevant information is reduced to strengthen the performance of data mining techniques. There are two main groups of techniques. Feature reduction, which aims to reduce the number of features [61]. Instance reduction, whose goal is to reduce the number of instances, which may differ in the following aspects: instance selection [58], which selects a subset of existing instances and instance generation [59], which creates new instances representative of the problem. A kNN-based instance generation algorithm recognized for its good results is [78], which takes the SSMA memetic algorithm as core to achieve a good reduction rate and accuracy.
- Noisy data: noise data are present in all datasets, reducing the quality of the results of data mining techniques. As for noise cleaning, there are numerous proposals for kNN-based algorithms. One of the most popular methods is the Edited Nearest Neighbour (ENN) [79], which removes all incorrectly labelled instances that do not agree with their k nearest neighbours.

As far as preprocessing of big datasets [14] based on kNN is concerned, we can find proposals in instance reduction [22], feature selection [22], and noise data filtering [80], among other techniques. However, as far as we know, there are no proposals for the imputation of MVs.

2.5 Complexity and redundancy Metrics

It is very important to know the problem that we are going to face in order to properly select the preprocessing and classification algorithms that will obtain a good result. Ho and Basu [81] propose the descriptors extracted from a learning dataset of a given classification problem. It also guides the development of new preprocessing and data mining techniques [82].

The most commonly used metrics in the characterization of classification problems are geometrically or statistically based. In addition, there are other families of metrics that focus on the separability of classes according to borders. Lorena et al. [13] differentiates 6 families of metrics: 1) Feature overlapping measure, which use feature statistics to measure separability between classes. 2) Neighborhood measures, which characterize the density between classes by proximity. 3) Class balance measure, which reports the ratio between the number of instances of each class. 4) Linearity measures, which determine whether a problem can be discerned by its classes through linear separation. 5) Network measures, modeling the data as a network and extracting information from its structure. 6) Dimensionality measures, evaluate data sparsity based on the number of samples relative to the number of features.

For the scope of this thesis, we will focus on the feature overlapping metrics and class balance metrics:

- F1. Maximum Fisher's discriminant ratio [83] measures the overlap between the features of the different classes of the problem. Specifically, it calculates the overlap of each feature separately, and takes the highest.
- F2. Volume of overlapping region [84] calculates the overlap between the samples of the different classes. In this case, it considers the domain by considering the maximum and minimum values of all features.
- F3. Maximum Individual Feature Efficiency [85] calculates the ratio of examples that are not in the overlap area and the total number of examples regarding the most discriminatory feature.
- F4. Collective Feature Efficiency [86] extends the F3 metric applying iteratively the same filter in order to get a more restrictive overlapping measure.

Finalize, indicating that there are no specific metrics for Big Data problems, which address the problem of redundancy, focusing on the number of instances needed and the existence of data redundancy in big datasets.

3 Justification

As it has been reflected in the previous sections, the kNN algorithm is very influential in the KDD process, being part of multiple data preprocessing techniques, as well as being a well known classifier used in real problems. The absence of the kNN algorithm capable of dealing with large datasets creates a gap in multiple areas of the KDD process in the Big Data domain. There are specific tools for the development of distributed computing software, as well as ML libraries with a variety of algorithms capable of scaling-up Big Data problems.

For the wide variety of preprocessing techniques, as well as complexity metrics based on the kNN algorithm and the classifier itself, the followings major issues should be properly addressed:

- Lazy learning algorithms, and kNN in particular, have a major impact in data mining. Before this Ph.D. thesis, there were only exact single-node and parallel proposals, as well as approximate proposals with a more accelerated computation. However, there were no quality distributed proposals that obtain the same result as the sequential version. For the design and development of the kNN algorithm as a distributed proposal capable of handling large datasets, and later to improve its effectiveness, specifically with its fuzzy version, we have considered:
    - First, we study and analyze the tools available for distributed development, paying attention to the data mining algorithm libraries for Big Data. With this acquired knowledge, and in order to meet the needs present in the development of kNN, we decided the paradigm and its implementation that fits best to our task.
    - Secondly, we study the state of the art to identify the strengths and weaknesses of sequential and parallel proposals. We also analyzed the Big Data versions with a local approach, which increases scalability but significantly reduces their effectiveness. Thus, we detected the need to design a scalable proposal with large datasets following a global approach that ensures the same result as its sequential version. Once the need is discovered, we apply the knowledge acquired for its design and development.
    - Finally, in the same way that a distributed global version of kNN is needed, it is widely known that its fuzzy variant improves efficiency in most classification problems. However, FkNN presents major scalability problems. For this release, we focus on using an accelerated approach to improve the efficiency and effectiveness of FkNN.
- Big Data preprocessing is an under-explored area compared to the numerous data mining techniques proposed, such as classifiers or regressors.

  The numerous kNN-based data preprocessing techniques have not been adapted to Big Data, or have been adapted using a local approach, considering the data by random chunck, leaving an important gap unattended. In order to fill this gap, we proceed as follows:
    - We studied the state of the art of kNN-based preprocessing techniques in traditional data mining problems, learning about their scalability and usefulness to be adapted to Big Data problems.
    - We design and implement techniques of different subcategories of preprocessing, specifically data reduction, noise detection and imputation of MVs. We conducted an in-depth study of Smart Data acquisition using the kNN algorithm to improve data quality.
- At a time prior to preprocessing, it is very important to know about the characteristics and particularities of the problem we are going to face. The study of characteristics includes

complexity metrics. However, when working with large datasets there are no metrics adapted to the particularities of Big Data problems, such as proposals on redundancy or complexity. To face this problem, we propose:

- We start by studying the state of the art of data complexity metrics in conventional problems, detecting the strengths and weaknesses of each to select those that can be re-designed for Big Data. In turn, we take ideas for the development of new metrics capable of scaling to large datasets.
- Once we acquired the necessary knowledge and detected the need for Big Data metrics, we re-designed those from the literature with a low computational cost, and designed two metrics, one based on Decission Tree Classifier for complexity, and another based on kNN for redundancy or density.

All these problems are interrelated and included within the theme of this doctoral thesis: Fast k-Nearest Neighbors for Big Data and Smart Data

4 Objectives

Once the main concepts of the state of the art have been introduced, we present the aim of this thesis. The overarching aim of this Ph.D. thesis is to accelerate the kNN algorithm in the context of Big Data and Smart Data. Specifically, to address it we have focused on the design, implementation, analysis, and evaluation of new algorithms capable of dealing with large datasets and perform a fast classification and obtaining quality data by preprocessing techniques. In addition, it is necessary to study the most commonly used tools in distributed computing, such as Apache Spark. The objectives are described in detail below:

1. **Enable the kNN algorithm to handle big datasets:**
   a. Study of the Big Data state-of-the-art, technology and paradigm: study of the current Big Data tools that will allow us to have the necessary knowledge about the efficiency and operation of the main platforms. The objective is to select and introduce the most appropriate technologies for the thesis development, along with the paradigm used to design and development of novel algorithms for Big Data.
   b. Design and implementation of the kNN algorithm in Hadoop and Spark: after analyzing the main tools and algorithms to handle large datasets, our objective is to design an exact and distributed kNN algorithm, which have an added difficulty due to the lack of training stage and storage of all instances. The implementation is on the Hadoop and Spark platforms to compare the behaviour of the algorithm.
2. **Fast approaches of the FkNN algorithm to improve the efficiency of kNN:**
   a. Study approximate approaches of kNN search to accelerate the compute without losing efficiency: to do this we review the literature with exact and approximate proposals that accelerate the kNN algorithm, both in sequential designs and in distributed designs.
   b. Scalable design and implementation of the FkNN algorithm in spark: once we have analyzed the possibilities for accelerating the kNN algorithm, our objective is to tackle the problem with three designs that cover local, global, exact and approximate approaches.
3. **Design of scalable algorithms to transform Big Data into Smart Data:**
   a. Study of the Big Data state-of-the-art on Smart Data and Data Preprocessing: theoretical and empirical study of the preprocessing techniques and Smart Data. The objective is to know the current methods that improve data quality.
   b. Design and implementation of MVs imputation based on kNN: once the needs for data preprocessing into Big Data problems are known, the goal is to propose instance-based algorithms for MVs imputation capable of handling large datasets.
4. **Design of metrics to study the complexity and redundancy in Big Data problems:**
   a. Study the state-of-the-art of complexity and redundancy metrics: knowing the complexity of a dataset before applying classification or preprocessing techniques provides valuable information to address the problem. The objective is to know the existing metrics until the moment of the thesis, focusing on those that can be adapted to large datasets.
   b. Design and implementation of specific complexity and redundancy metrics for Big Data problems: as far as we know, there are no specific metrics for Big Data problems. Thus, we aim to develop metrics capable of providing valuable information on the complexity and density of large datasets.

5 Methodology

This thesis has been constructed following the scheme of the traditional scientific method. Furthermore, it requires the integration of both practical and theoretical methodologies during its development. In particular, the following guidelines apply for research work and experiments:

1. **Observation**: through the study of machine learning problems with large-scale datasets and the study of Big Data technologies to develop distributed computing algorithms. Focusing specifically on instance-based learning algorithms as classifier and its application in data preprocessing, as a fundamental stage in the process of knowledge discovery in databases.

2. **Hypothesis formulation**: design of new classification algorithms and preprocessing techniques capable of scaling to address large-scale problems. The models designed and developed are within the framework defined by the objectives previously described, faced with Big Data problems.

3. **Observation gathering**: taking the results obtained by the proposed models, in large datasets and using different performance measures. In order to analyse their effectiveness and scalability, mainly through accuracy and execution time.

4. **Contrasting the hypothesis**: comparison of the results obtained with other models in the literature with the aim of analysing the quality of the proposals in terms of both efficiency and effectiveness. Thus, the most widespread machine learning libraries today are considered, MLlib as the official library of the Apache Spark project, and Spark-packages as a community library on the same platform.

5. **Hypothesis proof or refusal**: validation of the hypothesis raised through the experiments performed and the results obtained. If negative, rejection and modification of the hypothesis repeating the previous steps to ensure quality results.

6. **Scientific thesis**: Extraction and acceptance of conclusions according to the research process. Redaction of the conclusions obtained compiling the whole process and results into journal publications and the memory of the thesis.

6 Summary

This section summarizes the proposals and studies carried out in the publications associated with the thesis. Subsequently, Section 7 shows the main results obtained in each research paper. The journal publications are listed below:

- J. Maillo, S. Ramírez-Gallego, I. Triguero, F. Herrera. kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for Big Data. Knowledge-Based Systems. 117 (2017), 3-15. (Objective 1)
- J. Maillo, J. Luengo, S. García, F. Herrera, I. Triguero. Fast and Scalable Approaches to Accelerate the Fuzzy k Nearest Neighbors Classifier for Big Data. IEEE Transactions on Fuzzy Systems. Accepted 2019 (In press) DOI:10.1109/TFUZZ.2019.2936356. (Objective 2)
- I. Triguero, D. García-Gil, J. Maillo, J. Luengo, S. García, F. 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. 9:3 (2018). e1289. (Objective 3)
- J. Maillo, I. Triguero, F. Herrera. Redundancy and Complexity Metrics for Big Data Classification: Towards Smart Data. Submitted (Objective 4)

The remainder of this summary is organized according to the publications listed and the objectives detailed in Section 4. Firstly, Section 6.1 presents an exact and distributed proposal of the kNN algorithm capable of handle large-datasets. Then, Section 6.2 details how to improve the accuracy and scalability of the kNN algorithm with a fuzzy and fast version of it. After that, Section 6.3 provides an approach to generate quality data in Big Data problems, using as core the kNN classifier. Finally, Section 6.4 studies the massive amounts of data needed and the redundancy contained in large datasets by proposing two scalable metrics about complexity and density contained in large datasets.

6.1 Enabling the kNN classifiers to handle with Big Data problems

The kNN algorithm is widely known for its use in many areas of data science and for its simplicity and effectiveness when facing real problems. It belongs to the family of lazy learning algorithms, whose main characteristic is that they do not have a training phase, causing the storage of the complete training set and delaying the entire computer to the classification stage. This particularity produces a difficulty for the kNN algorithm when handling large datasets.

With this problem in view, our main objective is to enable the use of the kNN algorithm in the Big Data environment, as a classifier and as a tool to build the core of data preprocessing techniques. To do this, we design and develop a proposal of the kNN algorithm capable of handling large datasets and following the exact behavior, which means obtaining the same result as its sequential version. In the design process to adapt the algorithm to the Big Data environment, we identified two main problems:

- Runtime consumption: The computational complexity for calculating the nearest neighbor of an unseen sample is $O(n \cdot D)$, being n the number of intancias of the training set and D the number of features. In addition to this complexity, we have to add the repeated process for each instance of the test set. When both sets are very large, it is impractical to tackle it with a sequential approach.

- Memory consumption: for a quick calculation, it is necessary to have the information in main memory. When the sets are very large, we easily exceed the available main memory.

First, we designed and implemented the kNN algorithm following the MapReduce and specifically its Apache Hadoop implementation. The algorithm distributes the compute and storage of the instances that compose the dataset among a cluster of compute machines. Subsequently, the information from each machine is collected and the right k nearest neighbors of the set are obtained. This proposal of the kNN algorithm was published in the following conference paper:

- J. Maillo, I. Triguero, F. Herrera. A MapReduce-based k-Nearest Neighbor Approach for Big Data Classification. 9th International Conference on Big Data Science and Engineering (IEEE BigDataSE-15), Helsinki (Finland), 167-172, August 20-22, 2015

The results are promising and there is acceptable scalability. However, we focus on improve the design to accelerate it even more, using in this case the Apache Spark platform. The algorithm is called k Nearest Neighbors - Iterative Spark (kNN-IS).

Finally, to prove the efficiency and effectiveness of the kNN-IS algorithm, a full experimental study is conducted with datasets up to 11 million instances. In addition, the version designed in Spark is compared with the version designed in Hadoop, showing an improvement in runtime and scalability with the Spark implementation.

The jorunal article associated to this part is:

J. Maillo, S. Ramírez-Gallego, I. Triguero, F. Herrera. kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for Big Data. Knowledge-Based Systems. 117 (2017), 3-15.

6.2 FkNN: proposals to accelerate and improve the efficiency of kNN in Big Data

Once we have an exact version of the kNN algorithm, we aim to improve accuracy by designing a fuzzy version of the kNN classifier. FkNN is composed of two stages. The first, called class memberhisp degree, calculates kNN for each instance of the training set, and modifies the class label by a vector of membership in each class. Thus, it modifies the training set by introducing the specified change. The second, called classification, calculates kNN, with the difference that instead of applying a majority vote to decide the classification, it takes the class that has a higher degree of belonging. Again, we find runtime and memory consumption problems, but this time they are even stronger due to the need of the two stages. This algorithm is called Exact FkNN (EF-kNN), work collected in the following conference paper:

- J. Maillo, J. Luengo, S. Garcia, F. Herrera, I. Triguero. Exact Fuzzy k-Nearest Neighbor Classification for Big Datasets. IEEE Conference on Fuzzy Systems (FUZZ-IEEE 2017), Naples (Italy), July 9-12.

In this work the experimental study presented promising results in accuracy and high scalability. However, the exact approach produces high runtimes in the class membership degree stage, causing a pronounced bottleneck. We aim to alleviate the compute without affecting its effectiveness.

To address the new goal, we explore two alternatives, one local and one global:

- Local Hybrid Spill Tree FkNN (LHS-FkNN): the local approach divides the dataset into different parts and calculates the class membership degree internally in each partition, without considering other partitions.

- Global Approximate Hybrid Spill Tree FkNN (GAHS-FkNN): the global approach is based on the HS model. It generates a tree with the instances of the training set and distributes it among all the computation nodes, considering all the instances for the calculation of the class membership degree.

The class membership degree stage of the GAHS-FkNN algorithm and the classification stage of both proposed models are based on the Hybrid Spill Tree (HS) algorithm, proposed for the approximate and distributed search of kNN. HS is composed by the hybridization of two models based on trees built according to the distance between the samples: Metric-Tree (MT) ensures to find the nearest one, performing backtracking in the search if necessary, which causes slower runtimes. Spill-Tree (SP) does not ensure the nearest, but a representative one. To do this, it allows the duplication of instances in its branches, which receives the name of overlapping area, and does not perform backtracking in its search becoming faster than MT.

Finally, an experimental study is carried out to corroborate the quality of the proposed models. The complete study analyzes the accuracy, runtime, scalability, influence of the k-value and finishes by comparing the proposed models with their crips versions in a total of 8 datasets of up to 11 million instances.

The jorunal article associated to this part is:

J. Maillo, J. Luengo, S. García, F. Herrera, I. Triguero. Fast and Scalable Approaches to Accelerate the Fuzzy k Nearest Neighbors Classifier for Big Data. IEEE Transactions on Fuzzy Systems. Accepted 2019 (In press) DOI:10.1109/TFUZZ.2019.2936356.

6.3 On the use of kNN-based algorithm to transform Big Data into Smart Data

Although the kNN algorithm is known for its effectiveness as a classifier, it is negatively influenced by data imperfection. The high influence caused by data imperfection on the kNN algorithm is its main disadvantage as a classifier, but at the same time its strength to be used in preprocessing techniques, to detect and correct data imperfection and obtain quality data.

However, using kNN in large datasets presents problems with runtime scalability and memory consumption, as described in the previous section. Once the kNN proposal capable of dealing with large datasets has been designed, our main objective is to propose and study preprocessing techniques based on kNN capable of transforming large datasets into quality data, that is, converting Big Data into Smart Data.

Before the preprocessing proposals, a brief overview of Smart Data and current and future trends of the kNN in Big Data problems is made. After that, the main areas of preprocessing are selected to improve data quality:

- Data reduction: the aim of data reduction techniques is to discard instances while keeping as much information as possible. Thus, it reduces the necessary storage requirements, and reduces the training time of the data mining algorithms.
- Data noise and MVs: most data mining algorithms assume that data is perfect, and in reality there are imperfections. These imperfections range from taking values using sensors, labeling them in a manual process, corruption during storage or transmission, among other factors. In order to improve the quality of the knowledge obtained by the algorithms, techniques of data imperfection correction work by eliminating noisy instances, that contain erroneous information, or imputation of MVs, allowing the use of corrupt or wrongly taken samples.

To alleviate the three problems briefly described, we study and design kNN-based techniques capable of working with large datasets to obtain quality data:

- Data reduction: for the dataset reduction, we have used 4 algorithms based on kNN.
  - Fast Condensed Nearest Neighbor - MapReduce (FCNN_MR) incorporates the centroids of each class of the dataset and through an iterative process, adds the nearest instance of the opposite class to the centroid. When an iteration is completed without finding instances of the opposite class, it ends. The version used is local and executed through the MRPR framework.
  - Steady-State Memetic Algorithm - Scale Factor Local Search in Differential Evolution (SSMA-SFLSDE) makes a combination between selection and generation of prototypes. It starts based on SSMA for the selection of prototypes. Specifically, it uses a local search, and the selected instances become part of the population that will be optimized through the generation of prototypes of the differential evolution algorithm SFLSDE.
  - Democratic Instance Selection - MapReduce (MR-DIS) is a methodology for instance selection. MR-DIS divides the set into m disjointed parts of equal size, and makes a vote, according to the desired instance selector. This process is repeated a certain number N of times, and finally it is counted. Those instances that have received the more votes are eliminated. It is a process that is likely to be executed in a distributed mode.
  - Random Mutation Hill Climbing (RMHC_MR) to perform data reduction selects a random subset and through an iterative process selects the best subset using the kNN classifier. This is global, considering all the instances in the kNN application.
- Data noise: for noise reduction, we have selected 4 recognized algorithms in the literature based on kNN.
  - Edited Nearest Neighbor MapReduce (ENN_MR) to remove noisy instances, discards those that do not match the class label of its k nearest neighbors. The proposal for this publication is global, considering all instances of the dataset.
  - All k Nearest Neighbors MapReduce (All-kNN_MR) sigue exactamente el mismo funcionamiento que ENN_MR, pero se aplica en repetidas ocasiones para asegurar la eliminación de más instancias ruidosas. Al igual que ENN_MR, también es de carácter global.
  - Nearest Centroid Neighbor Edition (NCN-Edit_MR) detects and deletes noise as ENN does, but changes the neighborhood rule by including the centroid calculation in an iterative process. This causes a higher computational complexity, and to alleviate the compute a local approach has been followed in its implementation, through the MRPR framework.

  - Relative Neighborhood Graph Edition (RNG_MR) builds a neighborhood graph. To declare a noisy instance, it must be erroneously classified by its neighborhood graph. When it comes to the construction of graphs, the computational complexity is high and to alleviate it we make use of the MRPR framework and obtain a local approach.

- Missing values: for the imputation of MVs, a distributed and local version of the kNN-Imputation algorithm has been designed. kNN-I takes the missing value to be imputed as the new value to be predicted, in this sense, the nearest instances are calculated and the value used for the imputation is decided by majority vote if it is a categorical feature and the average if it is a continuous feature.

Finally, to study the effectiveness of the kNN algorithm for transforming Big Data into Smart Data, a study has been carried out using all the techniques described and a total of 7 datasets of up to 11 million instances and more than 600 features. In addition, all the models designed and developed are accessible in an open source software package.

The jorunal article associated to this part is:

I. Triguero, D. García-Gil, J. Maillo, J. Luengo, S. García, F. 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. 9:3 (2018). e1289.

6.4 Analyzing the complexity and redundancy of Big Data problems

Knowing the characteristics of a dataset is important to address a data science problem. Have information about the complexity of the problem allows us to make decisions about which data preprocessing or data mining techniques are the most suitable, achieving a more efficient and higher quality work.

There are many metrics that characterize complexity in classification problems, however, these metrics do not scale properly when faced with big datasets. In addition, there are no metrics to specifically address the particularities of Big Data problems, such as: When is Big Data too much data for machine learning and data mining algorithms? The main objective is to propose specific Big Data metrics capable of scaling to large datasets, in order to characterize and study a dataset before applying any preprocessing or data mining technique.

In this publication, we highlight the fact that the literature ignores the fact that there is redundancy in the datasets. To analyze the datasets and characterize them, we propose two metrics:

- Nighborhood Density (ND): it presents the proximity of samples by calculating the percentual difference of the Euclidean distance, which is calculated with all available data, and with the half of them randomly chosen.

- Decision Tree Progression (DTP): it measures complexity and redundancy by training two decision trees with the totality of the data, and discarding half of them randomly. After that, the percentual difference of the accuracy obtained with each model is calculated to reflect the loss of information.

In addition, we desing and adapt the main class imbalance and complexity metrics based on overlapping instances of each class to be able to handle large datasets.

Finally, an extensive experimental study has been carried out with the main big datasets of the literature to validate the quality of the proposed metrics and to highlight the imperative need to work on complexity and density metrics. In the study, an experiment is carried out reducing the dataset randomly and gradually, discarding up to 75% of the instances.

J. Maillo, I. Triguero, F. Herrera. Redundancy and Complexity Metrics for Big Data Classification: Towards Smart Data. Submitted.

7 Discussion of results

This section summarizes the analysis and results obtained in each stage of the thesis, in relation to the publications that form the compendium and have been summarized in the previous section. In addition, the Section 7.5 collects the source code of all implemented techniques associated with each jorunal publication.

7.1 Enabling the kNN classifiers to handle with Big Data problems

As described previously, we have designed an accurate and comprehensive model of the kNN classifier that is able to deal with Big Data problems. In addition, the development of kNN-IS will be used as a basis for preprocessing algorithms.

According to the experiments performed, the proposed solution to enable the kNN classifier with exact and global approach, called kNN-IS, is capable of running large datasets of up to 11 million instances and more than 600 features while maintaining the same result as in its sequential version. kNN-IS gets a speedup of approximately 1000 compared to its sequential version and 10 compared to the Hadoop implementation. In addition, the design achieves that the k number of neighbors to consider in the classification, does not drastically affect the runtime. Moreover, when the dataset exceeds the main memory capacity available in the cluster, the design of the algorithm allows, in a transparent way for the user, to iterate over the dataset scaling up to big datasets

In addition, an open source package has been developed and is available in spark packages at: `http://spark-packages.org/package/JMailloH/kNN_IS`

7.2 FkNN: proposals to accelerate and improve the efficiency of kNN in Big Data

We have proposed two models of FkNN, studying the local and global approach, as well as exact and approximate with the aim of improving accuracy and scalability as a classifier algorithm.

The experiments carried out to study the FkNN proposals show great scalability, as well as improving the results of the kNN and FkNN versions of the literature. Although there are no statistically significant differences in accuracy between the proposed GAHS-FkNN and LHS-FkNN models, there are differences in their scalability and efficiency.

On the one hand, LHS-FkNN obtains hardware-dependent scalability, and therefore we recommend its use if we have a powerful hardware infrastructure. On the other hand, GAHS-FkNN is more affected with a high number of features and its scalability is not so dependent on the available hardware, so we recommend its use if we have hardware limitations or datasets with a low number of features and a high number of instances.

All models have been published in the open source spark packages repository and are available at: `https://spark-packages.org/package/JMailloH/HS_FkNN`

7.3 On the use of kNN-based algorithm to transform Big Data into Smart Data

We have discussed the importance of the kNN algorithm in the area of data preprocessing, with the aim of transforming Big Data into Smart Data. For this purpose, we have studied in depth the problems faced by kNN to work with large datasets. In addition, we have reviewed the literature on

Smart Data. Thus, we have selected, designed and developed the main techniques of noise reduction, elimination of redundant information and imputation of MVs.

According to the experiments done we can suggest some guidelines and comments:

- Data redundancy is latent in all experiments. To reduce the size of datasets without eliminating relevant information, it drives to less demand of memory and also of compute. This allows for more effective application of data mining techniques, producing better results
- As with classic sized datasets, the existence of noisy data negatively affects the quality of data mining models. It can be cleaned by applying kNN-based filters, improving the results of other classifiers used as Decision Tree.
- The existence of MVs deteriorates or disables the application of any data mining algorithm. The imputation of MVs is necessary in order to avoid losing those affected instances and, therefore, to avoid losing the information contained in them. However, the experiments show a high redundancy of information, and this causes that discarding the instances affected by MVs does not drastically affect the results obtained by the classifiers.

Finally, it is important to note that all the software has been generated on the Apache Spark platform, and is available in the repository generated by the Spark Packages community. In this way, some of the main preprocessing problems can be addressed and the results obtained by data mining techniques can be improved.

All Smart Data algorithm have been published in three open source spark packages and are available at:

- Smart Reduction `https://spark-packages.org/package/djgarcia/SmartReduction`
- Smart Filtering `https://spark-packages.org/package/djgarcia/SmartFiltering`
- Smart Imputation `https://spark-packages.org/package/JMailloH/Smart_Imputation`

7.4 Analyzing the complexity and redundancy of Big Data problems

We have proposed two specific metrics for Big Data problems, measuring complexity and density. ND and DTP calculate, respectively, the percentage difference of density and accuracy with the complete dataset and by discarding half of the instances randomly. In addition, some complexity metrics from the literature based on overlap have been designed and adapted to handle big dataset.

The most relevant conclusion we can draw from the experimental study is the existence of high information redundancy in big datasets, and that reducing the number of instances randomly does not drastically affect the accuracy obtained by the classifiers. Thus, we can reduce the runtime and the memory it takes up, which are the two major issues of Big Data problems.

The experimental study shows a high scalability of all metrics. In addition, by reducing the size of the dataset, a significant reduction in run times can be seen in the three classifiers used. Thus, we can dedicate more time to applying preprocessing algorithm and optimize classifier parameters, in order to achieve a higher quality result.

All metrics have been published in the open source spark packages repository and are available at: `https://spark-packages.org/package/JMailloH/ComplexityMetrics`

7.5 Spark-based repositories associated with journal publications

This section presents all the source code generated during the development of the doctoral thesis. It is important to have transparency during the research process and to facilitate the replication of the experiments carried out. For this reason, and to give visibility to the research performed, each publication and the associated software package in Apache Spark is listed below:

- J. Maillo, S. Ramírez-Gallego, I. Triguero, F. Herrera. kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for Big Data. Knowledge-Based Systems. 117 (2017), 3-15.
  - kNN-IS: k Nearest Neighbors - Iterative Spark:
    `http://spark-packages.org/package/JMailloH/kNN_IS`
- J. Maillo, J. Luengo, S. García, F. Herrera, I. Triguero. Fast and Scalable Approaches to Accelerate the Fuzzy k Nearest Neighbors Classifier for Big Data. IEEE Transactions on Fuzzy Systems. Accepted 2019 (In press) DOI:10.1109/TFUZZ.2019.2936356.
  - HS-FkNN: Hybrid Spill Tree Fuzzy k Nearest Neighbors.
    `https://spark-packages.org/package/JMailloH/HS_FkNN`
- I. Triguero, D. García-Gil, J. Maillo, J. Luengo, S. García, F. 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. 9:3 (2018). e1289.
  - Smart Reduction:
    `https://spark-packages.org/package/djgarcia/SmartReduction`
  - Smart Filtering:
    `https://spark-packages.org/package/djgarcia/SmartFiltering`
  - Smart Imputation:
    `https://spark-packages.org/package/JMailloH/Smart_Imputation`
- J. Maillo, I. Triguero, F. Herrera. Redundancy and Complexity Metrics for Big Data Classification: Towards Smart Data. Submitted
  - Complexity Metrics:
    `https://spark-packages.org/package/JMailloH/ComplexityMetrics`

8 Conclusions and future work

This section concludes the whole thesis, collects the publications with their number of citations and provides some future research lines.

8.1 Conclusions

In this thesis, we have presented an extensive study of the kNN algorithm in Big Data problems and its application to transform Big Data into Smart Data. The objective has been to the design, implementation, analysis and evaluation of the proposed algorithms. This thesis started by enabling the original kNN classifier to tackle Big Data problems, and then we extended that proposal to allow its fuzzy variation, in order to improve the scalability and accuracy. Afterwards, the implication of the kNN algorithm in obtaining Smart Data is analysed, highlighting the proposal as an imputation of MVs. Finally, two specific complexity and density metrics for Big Data problems are proposed in order to study the redundancy information in large scale datasets.

In the first objective, a proposal of the kNN algorithm has been designed and developed in Spark, capable of dealing with large datasets. In addition, it is compared with an own implementation developed in Hadoop, making a comparative study between both implementations, Spark and HAdoop, of the MapReduce paradigm. The design of the algorithm makes the impact of the k parameter negligible in practice in terms of runtime. In addition, due to its design it allows, in a transparent way for the user, to execute datasets that exceed the available main memory, through the automatic management of hard disk and main memory by an iterative process.

As a conclusion to this objective, we recommend the use of the MapReduce paradigm, and the implementation of Spark instead of Hadoop, obtaining an acceleration of approximately 1000 times compared to the sequential version, and up to 10 times compared to the distributed version implemented in Hadoop, the MR-kNN algorithm.

Regarding the second objective, two FkNN models have been designed with the purpose of improving the accuracy and scalability of the kNN-IS algorithm. Two approaches have been followed: local and global. LHS-FkNN has a local approach, and through the experiments performed, there is a strong hardware dependent scalability. GAHS-FkNN has a global approach, and the experimentation shows a slight dependency on hardware features and is more affected by the number of features of the datasets.

As conclusion for this objective, we can highlight the good performance of the global approximate models, which improve both the accuracy and the runtime. In addition, we recommend the use of the LHS-FkNN algorithm if a valid hardware infrastructure is available, and the GAHS-FkNN algorithm when we are faced with a high number of features or with hardware limitations.

Thirdly, our objective is to highlight the importance of the kNN algorithm in data preprocessing in order to obtain Smart Data. In the present research we reviewed the literature, collected and developed the main techniques for missing values imputation, data reduction and noisy data cleaning.

As a conclusion to this objective, we highlight the following three points: 1) Although the existence of MVs is a disadvantage for data mining techniques, the experimental study reveals that their imputation does not show a significant improvement. This reflects the existing high data redundancy. 2) Data reduction is essential to alleviate the need for storage and preprocessing, enabling the use of high quality data mining techniques. 3) Filtering noisy instances improves the performance of most data mining algorithms, enabling the main kNN-based noise filtering techniques.

Finally, the fourth objective is the characterization of datasets previous to the preprocessing and data mining stage. For this purpose, two specific Big Data metrics have been designed to measure complexity and density: ND and DTP. In addition, the main complexity metrics based on feature overlapping and class unbalance have been adapted to Big Data problems.

According to the experimental study carried out, we concluded that it is very common to appreciate redundancy information in the samples of the big datasets. This redundancy allows to randomly reduce the dataset up to 25% of the original size, without drastically affecting the accuracy obtained by the classifiers. Thus, we highlight the need for new complexity metrics specific to Big Data problems, and the importance of data preprocessing to obtain smaller and better quality datasets.

Conclusiones

En esta tesis hemos presentado un amplio estudio sobre el algoritmo kNN en problemas Big Data y su uso para transformar grandes conjuntos de datos en Smart Data. Como objetivo se ha planteado el diseño, implementación, análisis y evaluación de los algoritmos propuestos. Comienza con la propuesta escalable y exacta del algoritmo kNN y se extiende con la mejora aproximada de FkNN. Posteriormente, se ha analizado la implicación del algoritmo kNN en la obtención de Smart Data, destacando la propuesta como imputador de MVs. Finalmente, se han propuesto dos métricas de complejidad y densidad específicas para problemas Big Data con el propósito de estudiar la redundancia de información en conjuntos de datos de gran escala.

En el primer objetivo planteado, se ha logrado diseñar y desarrollar en Spark una propuesta del algoritmo kNN, capaz de abordar grandes conjuntos de datos. Además, se compara con una implementación propia desarrollada en Hadoop, realizando un estudio comparativo entre ambas implementaciones del paradigma MapReduce. El diseño del algoritmo hace despreciable en la práctica el impacto de la variable k en cuanto a tiempo de ejecución. Además, debido a su diseño permite, de forma transparente para el usuario, ejecutar conjuntos de datos que exceden la memoria principal disponible, mediante la gestión automátizada del disco duro y la memoria principal a través de un proceso iterativo.

Como conlusión a este objetivo, recomentamos el uso del paradigma MapReduce, y la implementación de Spark por enima de Hadoop, obteniendo una aceleración de aproximadamente 1000 veces comparado con la versión secuencial, y hasta 10 veces respecto a la versión distribuida implementada en Hadoop, el algoritmo MR-kNN.

Respecto al segundo objetivo, se han diseñado dos modelos de FkNN con el propósito de mejorar el accuracy y la escalabilidad del algoritmo kNN-IS. Para ello se han seguido dos enfoques: local y global. LHS-FkNN tiene un enfoque local, y mediante los experimentos realizados se aprecia una escalabilidad fuertemente dependiente del hardware. GAHS-FkNN tiene un enfoque global, y la experimentación muestra una dependencia leve con las características hardware y se ve más afectado por conjuntos de datos con alto número de variables.

Como conclusión principal para este objetivo, podemos destacar el buen funcionamiento de los modelos aproximados globales, que mejoran tanto la precisión como el tiempo de ejecución. Además, recomendamos el uso de LHS-FkNN si se dispone de una infraestructura hardware adecuada, y de GAHS-FkNN cuando nos encontramos con un número elevado de variables o con limitaciones hardware.

En tercer lugar, se ha planteado como objetivo la importancia del algoritmo kNN en el preprocesamiento de datos para la obtención de Smart Data. En el trabajo realizado revisamos la literatura, recopilamos y desarrollamos las principales técnicas de imputación de MVs, reducción de datos y limpieza de datos ruidosos.

Como conclusión para este objetivo, podemos destacar los siguientes tres aspectos: 1) Aunque la existencia de MVs perjudica a las técnicas de minería de datos, el estudio experimental muestra que su imputación no muestra una mejora significativa. Esto refleja la alta redundancia de datos existente. 2) La reducción de datos es primordial para aliviar la necesidad de almacenamiento y preprocesamiento, habilitando la utilización de técnicas de minería de datos de alta calidad. 3) Filtrar las instancias ruidosas mejora el rendimiento de la mayoría de los algoritmos de minería de datos, quedando habilitadas las principales técnicas de filtro de ruido basadas en kNN.

Finalmente, el cuarto objetivo es la caracterización de conjuntos de datos previo a la etapa de preprocesamiento y minería de datos. Para ello se han diseñado dos métricas específicas de Big Data para medir la complejidad y densidad: ND y DTP. Además, se han adaptado para problemas Big Data las principales métricas de complejidad basadas en solapamiento de variables, y de desequilibrio de clases.

Con este objetivo concluimos de acuerdo al estudio experimental realizado, que es muy común apreciar redundancia de información en las muestras de los grandes conjuntos de datos. Esta redundancia permite reducir aleatoriamente el conjunto de datos hasta el 25 % del tamaño original, sin afectar drásticamente a la precisión obtenida por los clasificadores. Así, destacamos la necesidad de nuevas métricas de complejidad específicas para problemas Big Data, y la importancia del preprocesamiento de datos para obtener conjuntos de datos más reducidos y de calidad.

8.2 Publications

This section lists journal papers and conference contributions published during the PhD study period. The publications ordered by date of publishing and differentiated in two groups, indicating the number of received citations in Google Scholar are listed below:

- Journal papers:

  1. J. Maillo, S. Ramírez-Gallego, I. Triguero, F. Herrera. kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data. Knowledge-Based Systems 117 (2017) 3-15. CITED BY: 144.
  2. I. Triguero, D. García-Gil, J. Maillo, J. Luengo, S. García, F. 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. 9:3 (2018). e1289. CITED BY: 8.
  3. J. Maillo, J. Luengo, S. García, F. Herrera, I. Triguero. Fast and Scalable Approaches to Accelerate the Fuzzy k Nearest Neighbors Classifier for Big Data. IEEE Transactions on Fuzzy Systems. Accepted 2019 (In press)
  4. J. Maillo, I. Triguero, F. Herrera. Redundancy and Complexity Metrics for Big Data Classification: Towards Smart Data. Submitted

- Conference contributions:

  1. J. Maillo, I. Triguero, F. Herrera. A MapReduce-based k-Nearest Neighbor Approach for Big Data Classification. 9th International Conference on Big Data Science and Engineering (IEEE BigDataSE-15), Helsinki (Finland), 167-172, August 20-22, 2015. CITED BY: 61.
  2. I. Triguero, J. Maillo, J. Luengo, S. García, F. Herrera. From Big data to Smart Data with the K-Nearest Neighbours algorithm. The 2016 IEEE International Conference on Smart Data (SmartData 2016), Chengdu (China), Dec 16-19, 2016. CITED BY: 9.
  3. J. Maillo, J. Luengo, S. Garcia, F. Herrera, I. Triguero. Exact Fuzzy k-Nearest Neighbor Classification for Big Datasets. IEEE Conference on Fuzzy Systems (FUZZ-IEEE 2017), Naples (Italy), July 9-12. CITED BY: 13
  4. J. Maillo, J. Luengo, S. Garcia, F. Herrera, I. Triguero. A preliminary study on Hybrid Spill-Tree Fuzzy k-Nearest Neighbors for big data classification. IEEE Conference on Fuzzy Systems (FUZZ-IEEE 2018), Rio de Janeiro (Brazil), July 8-13. CITED BY: 4.
  5. B. Montesdeoca, J. Luengo, J. Maillo, D. García-Gil, S. García, F. Herrera. A First Approach on Big Data Missing Values Imputation. 4th International Conference on Internet of Things, Big Data and Security. 2-4 May 2019. pp 315-323.

8.3 Future work

The results obtained in this PhD thesis enable new lines of research, presenting new challenges to address in Big Data problems. In this section we present some problems that can be addressed based on the research work done:

- Missing Values Imputation: The missing values imputation [19] is responsible for alleviating a real problem whose origin is in the wrong collection of the data. Although a kNN-based model has been proposed for MVs imputation, this is a local and approximate model. In addition, a first approach has been designed for the missing values imputation based on clustering [87] that improves the imputation with the average or discards those instances that have been affected. During the development of the thesis, we showed two FkNN proposals based on Hybrid-spill tree, which show promising scalability and accuracy.

  An interesting research line could be focused on using these algorithms to investigate whether the use of fuzzy models could improve the imputation. Another important line of research is the influence of MVs with a bias for a specific class, which is a common situation in real problems. If they are not addressed through the appropriate imputation, it generates an imbalance between classes that can significantly affect the result obtained by data mining techniques.

- Semi-supervised learning: semi-supervised learning problems [30] requires supervised and unsupervised learning techniques, and it is an under-explored line of research in the Big Data context. Specifically, is interesting in large-data sets due to the difficulty of having properly labelled samples because the process of labelling manually the classes is tedious and expensive. The techniques used to alleviate this problem is known as self-labelling [31].

  The kNN algorithm can be a solid base for self-labelling techniques, automatically labeling from a small set of manually labeled samples. The developed FkNN models represent a proposal which is worth to study in depth.

- Complexity and redundancy metrics: the complexity metrics [13] provides valuable information to address a problem. Thus, knowing some relevant characteristics facilitates the decision on which algorithms to use.

  Although we have developed two proposals on complexity and redundancy metrics for Big Data problems, it is clear that there is a need for metrics capable of identifying the necessary size of the dataset, the existence of noise, or other specific approaches for special characteristics of Big Data problems.

- Auto-machine learning : is a novel area of research that is becoming trending and brings the usefulness of the entire KDD process to non-expert users. For this purpose, auto-machine learning [88] is involved in the automation of the whole pipeline from raw data, going through the selection of data preprocessing and data mining algorithms and parameter optimization, until the generation of reports about the analysis or visualization of the results.

  In this new area of research, we can provide the integration of the proposed complexity metrics, as well as propose new metrics in the context of Big Data. Focusing on improving the selection of data preprocessing and data mining techniques, with the aim of reducing the commonly high runtime and memory consumption in large datasets.

- Big data stream: the main characteristic of the data stream problems [89] is a continuous supply of raw data, which need to be processed in a limited time. This area needs preprocessing techniques capable of select the samples that contribute usefull information to the problem, capable of descarding noisy or redundancy data among others difficulties.

  In this research line, the kNN algorithm has a horizontal behavior, similar to the KDD process described during this thesis, covering the core of several preprocessing and classification techniques. As a tentative proposal, an incremental model based on kNN can be provided, which through prototype generation obtains a smart dataset, reduced and without noisy or redundant information that may provide fast and accurate results.

Chapter II

Publications

1 kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data

- J. Maillo, S. Ramírez-Gallego, I. Triguero, F. Herrera. kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data. Knowledge-Based Systems. 117 (2017), 3-15.
  - Status: **Published**.
  - Impact Factor (JCR 2017): **4.396**
  - Subject Category: **Computer Science, Artificial Intelligence**
  - Rank: **14/132**
  - Quartile: **Q1**

kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors Classifier for Big Data

Jesus Maillo
Department of Computer Science
and Artificial Intelligence
University of Granada, Granada, Spain 18071
jesusmh@decsai.ugr.es

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

Isaac Triguero
Department of Internal Medicine
Ghent University, Ghent, Belgium, 9000
Data Mining and Modelling for Biomedicine group
VIB Inflammation Research, Zwijnaarde, Belgium, 9052
School of Computer Science, University of Nottingham
Jubilee Campus, Nottingham NG8 1BB, United Kingdom
Isaac.Triguero@nottingham.ac.uk

Francisco Herrera
Department of Computer Science
and Artificial Intelligence
University of Granada, Granada, Spain 18071
Faculty of Computing and Information Technology
University of Jeddah, Jeddah, Saudi Arabia, 21589
herrera@decsai.ugr.es

Abstract

The k-Nearest Neighbors classifier is a simple yet effective widely renowned method in data mining. The actual application of this model in the big data domain is not feasible due to time and memory restrictions. Several distributed alternatives based on MapReduce have been proposed to enable this method to handle large-scale data. However, their performance can be further improved with new designs that fit with newly arising technologies.

In this work we provide a new solution to perform an exact k-nearest neighbor classification based on Spark. We take advantage of its in-memory operations to classify big amounts of unseen cases against a big training dataset. The map phase computes the k-nearest neighbors in different training data splits. Afterwards, multiple reducers process the definitive neighbors from the list obtained in the map phase. The key point of this proposal lies on the management of the test set, keeping it in memory when possible. Otherwise, it is split into a minimum number of pieces, applying a MapReduce per chunk, using the caching skills of Spark to reuse the previously partitioned training set. In our experiments we study the differences between Hadoop and Spark implementations with datasets up to 11 million instances, showing the scaling-up capabilities of the proposed approach. As a result of this work an open-source Spark package is available.

***K*eywords** K-nearest neighbors · Big data · Apache Hadoop · Apache Spark · MapReduce

1 Introduction

Over the last few years, gathering information has become an automatic and relatively inexpensive task, thanks to technology improvements. This has resulted in a severe increment of the amount of available data. Social media, biomedicine or physics are just a few examples of areas that are producing tons of data every day [1]. This data is useless without a proper knowledge extraction process that can somehow take advantage of it. This fact poses a significant challenge to the research community because standard machine learning methods can not deal with the volume, diversity and complexity that this data brings [2]. Therefore, existing learning techniques need to be remodeled and updated to deal with such volume of data.

The k-Nearest Neighbor algorithm (kNN) [3] is an intuitive and effective nonparametric model used for both classification and regression purposes. In [4], the kNN was claimed to be one of the ten most influential data mining algorithms. In this work, we are focused on classification tasks. As a lazy learning model, the kNN requires that all the training data instances are stored. Then, for each unseen case and every training instance, it performs a pairwise computation of a certain distance or similarity measure [5, 6], selecting the k closest instances to them. This operation has to be repeated for all the input examples against the whole training dataset. Thus, the application of this technique may become impractical in the big data context. In what follows, we refer to this original algorithm as the exact kNN method, w.r.t. partial and approximate variants of the kNN model that reduce the computational time, assuming that distances are computed using any class of approximation error bound [7].

Recent cloud-based technologies offer us an ideal environment to handle this issue. The MapReduce framework [8], and its open-source implementation in Hadoop [9], were the precursor tools to tackle data-intensive applications [10] based on the principle of data locality [11], which is implemented through its distributed file system. Its application in data mining has been widely spread [12, 13, 14], to the detriment of other parallelization schemes such as Message Passing Interface [15], because of its fault-tolerant mechanism (recommendable for time-consuming tasks) and its ease of use [16]. Despite its unquestionable breakthrough, researchers have found several limitations in Hadoop Mapreduce to design scalable machine learning tools [17]. MapReduce is inefficient for applications that share data across multiple steps, including iterative algorithms or interactive queries. Multiple platforms for large-scale processing have recently emerged to overcome the issues presented by Hadoop MapReduce [18, 19]. Among them, Spark [20] highlights as one of the most flexible and powerful engines to performed faster distributed computing in big data by using in-memory primitives. This platform allows user programs to load data into memory and query it repeatedly, making it more suitable for online, iterative or data streams algorithms [21].

The use of the kNN algorithm and similar approaches has been already considered in the big data context. On the one hand, some works incorporate a kNN classifier in a MapReduce process [22], but their purpose is not to carry out an exact kNN classification, but use a partial kNN (kNN is applied over subsets of the training data) as part of a larger pipeline of experiments. In [23] the authors proposed a novel approach for clustering in large datasets by adding kNN and Principal Component Analysis as part of the technique proposed. The method proposed in [24] have two different stages. The first stage used a k-means in order to separate the whole dataset in different parts. The second stage computes a kNN in each split providing approximate results. On the other hand, without aiming at classification or regression tasks, several approaches have been proposed to perform a distributed computation of kNN join queries in MapReduce. For example, in [25] the authors apply kNN-join (exact or approximate) queries within a two-stage MapReduce process. In [26] the authors proposed *Spitfire*, an efficient and scalable kNN queries model composed of multiple distributed stages. We further discuss these methods in Section 2.2. When focused on pure classification, the MapReduce process can be greatly simplified because it is not necessary to provide the k nearest neighbors themselves, but rather their classes. In [27], an iterative Hadoop MapReduce process (iHMR-kNN) was presented for kNN based image classification. This approach iteratively performs MapReduce for every single test instance, with the consequent time consumption of Hadoop-based systems for iterations. In [28], however, we proposed a single Hadoop MapReduce process that can simultaneously classify large amounts of test samples against a big training dataset, avoiding start-up costs of Hadoop. To do so, we read the test set line by line from the Hadoop File System, which make this model fully scalable but its performance can be further improved by in-memory solutions.

In this paper, we propose an iterative MapReduce-based approach for kNN algorithm implemented under Apache Spark. In our implementation, we aim to exploit the flexibility provided by Spark, by using other in-memory operations that alleviate the consumption costs of existing MapReduce alternatives. To manage enormous test sets as well, this method will iteratively address chunks of this set, if necessary. The maximum number of possible test examples, depending on memory limitations, will be used to minimize the number of iterations. In each iteration, a kNN MapReduce process will be applied. The map phase consists of deploying the computation of similarity between a subset of the test examples and splits of the training set through a cluster of computing nodes. As a result of each map, the class label of the k nearest neighbors together with their computed distance values will be emitted to the reduce stage. Multiple reducers will determine which are the final k nearest neighbors from the list provided by the maps. This process is repeated until the whole test set is classified. Through the text, we will denote this approach as a kNN design based on Spark (kNN-IS).

In summary, the contributions of this work are as follows:

- We extend the MapReduce scheme proposed in [28] by using multiples reducers to speed up the processing when the number of maps needed is very high.
- A fully parallel implementation of the kNN classifier that makes use of in-memory Spark operations to accelerate all the stages of the method, including normalization of the data, processing of big test datasets, and computation of pairwise similarities, without incurring in Hadoop startup costs.

To test the performance of the proposed classification model, we will conduct experiments on big datasets with up to 11 millions instances. We investigate the influence of number of maps and reducers and we will establish a comparison among existing Hadoop MapReduce alternatives and the proposed approach. A repository of code with the implementation of this technique can be found at `https://github.com/JMailloH/kNN_IS`.

The remainder of this paper is organized as follows. Section 2 introduces the big data technologies used in this work and the current state-of-art in kNN big data classification. Then, Section 3 details the proposed kNN-IS model. Section 4 describes the experimental setup and Section 5 includes multiple analyses of results. Finally, Section 6 outlines the conclusions drawn in this work. The Appendix provides a quick start guide with the developed Spark package.

2 Preliminaries

This section provides the necessary background for the remainder of the paper. First, Section 2.1 introduces the concept of MapReduce and the platforms Hadoop and Spark. Then, Section 2.2 formally defines the kNN algorithm and its weaknesses to tackle big data problems, presenting the current alternatives to alleviate them.

2.1 MapReduce Programming Model and Frameworks: Hadoop and Spark

The MapReduce programming paradigm [8] is a scale-out data processing tool for Big Data, designed by Google in 2003. This was thought to be the most powerful search-engine on the Internet, but it rapidly became one of the most effective techniques for general-purpose data parallelization.

MapReduce is based on two separate user-defined primitives: Map and Reduce. The Map function reads the raw data in form of key-value ($< key, value >$) pairs, and transforms them into a set of intermediate $< key, value >$ pairs, conceivably of different types. Both key and value types must be defined by the user. Then, MapReduce merges all the values associated with the same intermediate key as a list (shuffle phase). Finally, the Reduce function takes the grouped output from the maps and aggregates it into a smaller set of pairs. This process can be schematized as shown in Figure 1.

This transparent and scalable platform automatically processes data in a distributed cluster, relieving the user from technical details, such as: data partitioning, fault-tolerance or job communication. We refer to [16] for an exhaustive review of this framework and other distributed paradigms.

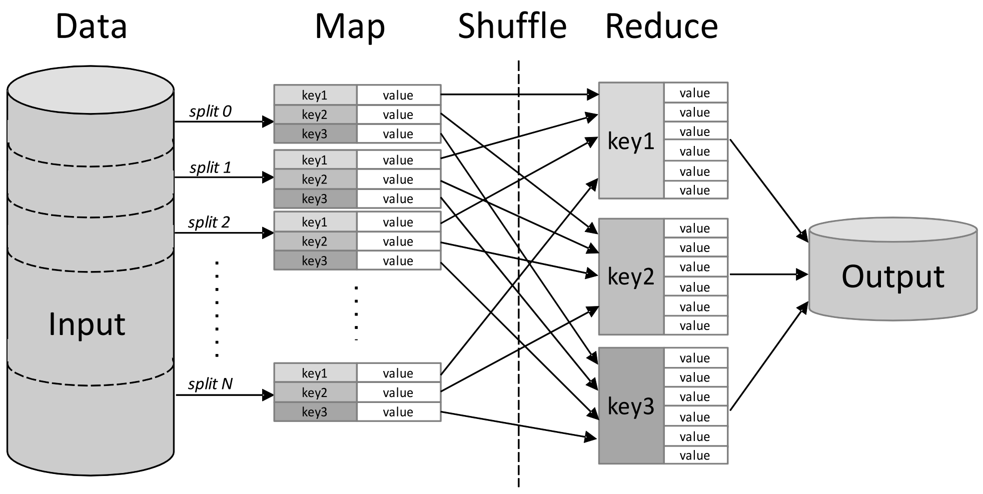


Figure 1: Data flow overview of MapReduce

Apache Hadoop [29, 30] is the most popular open-source implementation of MapReduce for large-scale processing and storage on commodity clusters. The use of this framework has become widespread in many fields because of its performance, open source nature, installation facilities and its distributed file system (Hadoop Distributed File System, HDFS). In spite of its great popularity, Hadoop and MapReduce have shown not to fit well in many cases, like online or iterative computing [31]. Its inability to reuse data through in-memory primitives makes the application of Hadoop for many machine learning algorithms unfeasible.

Apache Spark, a novel solution for large-scale data processing, was thought to be able to solve the Hadoop's drawbacks [32, 33]. Spark was introduced as part of the Hadoop Ecosystem and it is designed to cooperate with Hadoop, specially by using its distributed file system. This framework proposes a set of in-memory primitives, beyond the standard MapReduce, with the aim of processing data more rapidly on distributed environments, up to 100x faster than Hadoop.

Spark is based on Resilient Distributed Datasets (RDDs), a special type of data structure used to parallelize the computations in a transparent way. These parallel structures let us persist and reuse results, cached in memory. Moreover, they also let us manage the partitioning to optimize data placement, and manipulate data using a wide set of transparent primitives. All these features allow users to easily design new data processing pipelines.

A scalable machine learning library (MLlib) [34] was built on top of Spark, thanks to its implicit suitability for iterative processes. The current version of MLlib (v1.6.0) contains a large set of standard learning algorithms and statistic tools, which covers many important fields in the knowledge discovery process, such as: classification, regression, clustering, optimization or data pre-processing. The MLlib is a key component of the MLbase [35] platform. It provides a high-level API that makes easier for the user to connect multiple machine learning algorithms. However, this platform does not include lazy learning algorithms such as the kNN algorithm.

2.2 The kNN classifier and big data

The kNN algorithm is a non-parametric method that can be used for either classification and regression tasks. Here, we define the kNN problem, its current trends and the drawbacks to manage big data. A formal notation for the kNN algorithm is the following:

Let TR be a training dataset and TS a test set, they are formed by a determined number **n** and **t** of samples, respectively. Each sample $\mathbf{x}_p$ is a tuple $(\mathbf{x}_{p1}, \mathbf{x}_{p2}, ..., \mathbf{x}_{pD}, \omega)$, where, $\mathbf{x}_{pf}$ is the value of the f-th feature of the p-th sample. This sample belongs to a class ω, given by $\mathbf{x}_p^{\omega}$, and a D-dimensional space. For the TR set the class ω is known, while it is unknown for TS. For each sample $\mathbf{x}_{test}$ included in the TS set, the kNN algorithm searches the k closest samples in the TR set. Thus, the kNN calculates the distances between $\mathbf{x}_{test}$ and all the samples of TR. The Euclidean distance is the most widely-used measure for this purpose. The training samples are ranked in ascending order according to the computed distance, taking the k nearest samples $(\mathbf{neigh}_1, \mathbf{neigh}_2, ..., \mathbf{neigh}_k)$. Then, they are used to compute the most predominant class label. The chosen value of k may influence the performance and the noise tolerance of this technique.

Although the kNN has shown outstanding performance in a wide variety of problems, it lacks the scalability to manage big TR datasets. The main problems found for dealing with large-scale data are:

- Runtime: The complexity to find the nearest neighbor training example of a single test instance is $O((n \cdot D))$, where n is the number of training instances and D the number of features. This becomes computationally more expensive when it involves finding the k closets neighbors, since it requires the sorting of the computed distances, so that, an extra complexity $O(n \cdot log(n))$. Finally, this process needs to be repeated for every test example.
- Memory consumption: For a rapid computation of the distances, the kNN model requires the training data to be stored in memory. When TR and the TS sets are too big, they may easily exceed the available RAM memory.

These drawbacks motivate the use of big data technologies to distribute the processing of kNN over a cluster of nodes.

In the literature, we can find a family of approaches that perform kNN joins with MapReduce. A recent review on this topic can be found in [36]. The kNN joins differs from kNN classifier in the expected output. While the kNN classifier aims to provide the predicted class, the kNN join outputs the neighbors themselves for a single test. Thus, these methods cannot be applied for classification.

For an exact kNN join, two main alternatives have been proposed in [25]. The first one, named H-BkNNJ, consists of a single round of MapReduce in which TR and TS sets are partitioned together, so that, every map task processes a pair TS_i and TR_i, and carries out the pairwise distance comparison between each training and test splits. Let m the number of used partitions, it creates m^2 blocks by performing a linear scan on both sets. The reduce task then processes all computed distances for a given test instance and sorts them in ascending order to output the top k results. A second alternative called H-BNLJ is proposed, by using two MapReduce processes, in order to reduce the complexity of the sort phase. However, it still requires m^2 tasks. The main deficiencies of these approaches are: (1) they generate extra blocks of data, and therefore, they make the size of the problem tackled even bigger; (2) they square the complexity of the solution (m^2 tasks);

(3) it relied on Hadoop MapReduce, so that, the two-stage MapReduce model needs to serialize intermediate data into disk, with its consequent cost. Some other models, such as PGBJ [37], perform a preprocessing phase and distance based partitioning strategy to reduce the number of task to m. Nevertheless, it adds an extra computational cost to carry out this phase. More recently, a new alternative called Spitfire was proposed in [26]. Following its own distributed procedure (i.e. not a MapReduce model), it calculates the k nearest neighbors of all the elements of a single set. To do so, it first partitions the search space, and then, calculates and replicates the k nearest neighbors in each split. The last phase computes a local kNN to provide the final result.

Focusing on classification tasks (also valid for regression), existing methods are simpler than kNN join approaches, since they do not need to provide the neighbors themselves (or reference to them to search for them later), only their classes. Two main approaches have been presented so far, and they are both focused on using the Map phase to split the training data in m disjoint parts. The former was presented in [27], and it proposes iteratively repeating a MapReduce process (without an explicitly defined reduce function) for each single test example, which is very time consuming in both Hadoop and also in Spark (as we will discuss further in the experiment section). The latter was proposed in [28], denoted as MR-kNN, in which a single MapReduce process manages the classification of the (big) test set. To do that, Hadoop primitives are used to read line by line the test data within the map phase. As such, this model is scalable but its performance can be further improved by in-memory solutions.

3 kNN-IS: An Iterative Spark-based design of the kNN classifier for Big Data

In this section we present an alternative distributed kNN model for big data classification using Spark. We will denote our method as kNN-IS. We focus on the reduction of the runtime of the kNN classifier, when both training and test sets are big datasets. As stated in [36], when computing kNN within a parallel framework, many additional factors may impact the execution time, such as number of MapReduce jobs j or number of Map m and Reduce r tasks required. Therefore, writing an efficient exact kNN in Spark is challenging, and multiple key-points must be taken into account to obtain an efficient and scalable model.

Aiming to alleviate the main issues presented by previously MapReduce solutions for kNN, we introduce the following ideas in our proposal:

- As in [28] and [27], a MapReduce process will split the training dataset, as it is usually the biggest dataset, into m tasks. In contradistinction to kNN-join approaches that need m^2 tasks, we reduce the complexity of kNN to m tasks without requiring any preprocessing in advanced.
- To tackle large test datasets, we rely on Spark to reuse the previously split training set with different chunks of the test set. The use of multiple MapReduce jobs over the same data does not imply significant extra costs in Spark, but we keep this number to a minimum. The MR-kNN approach only performs m tasks independently of the test data size, by reading line-by-line the test set within the maps. Here we show how in-memory operations highly reduce the cost of every task.
- It is also noteworthy that none of the alternatives proposed for pure kNN classification (e.g. [28, 27]) discuss the influence of the number of reducers, which can be determinant when the size of the dataset becomes very big (See Section 5.3).
- In addition, every single operation will be performed within the RDD objects provided by Spark. It means that even simple operations such as normalization, are also efficient and fully scalable.

This is the reasoning behind our model. In what follows, we detail its main components. First of all, we will present the main MapReduce process that classifies a subset of the test set against the whole training set

(Section 3.1). Then, we give a global overview of the method, showing the details to carry out the iterative computation over the test data (Section 3.2).

3.1 MapReduce for kNN classification within Spark

This subsection introduces the MapReduce process that will manage the classification of subsets of test data that fit in memory. As such, this MapReduce process is based on our previously proposed alternative MR-kNN, with the distinction that it allows for multiple reducers, checks the iterations required to run avoiding memory swap, and is implemented under Spark.

As a MapReduce model, this divides the computation into two main phases: the map and the reduce operations. The map phase splits the training data and calculates for each chunk the distances and the corresponding classes of the k nearest neighbors for every test sample. The reduce stage aggregates the distances of the k nearest neighbors from each map and makes a definitive list of k nearest neighbors. Ultimately, it conducts the usual majority voting procedure of the kNN algorithm to predict the resulting class. Map and reduce functions are now defined in Sections 3.1.1 and 3.1.2, respectively.

3.1.1 Map Phase

Let us assume that the training set TR and the corresponding subset of test samples TS_i have been previously read from HDFS as RDD objects. Hence, the training dataset TR has already been split into a user-defined number m of disjoint subsets when it was read. Each map task ($Map_1, Map_2, ..., Map_m$) tackles a subset TR_j, where $1 \leq j \leq m$, with the samples of each chunk in which the training set file is divided. Therefore, each map approximately processes a similar number of training instances.

To obtain an exact implementation of kNN, the input test set TS_i is not split together with the training set, but it is read in each map in order to compare every test sample against the whole training set. It implies that both TS_i and TR_j are supposed to fit altogether in memory.

Algorithm 1 contains the pseudo-code of this function. In our implementation in Spark we make use of the *mapPartitions(func)* transformation, which runs the function defined in Algorithm 1 on each block of the RDD separately.

Algorithm 1 Map function

```
Require: $TR_j$ $TS_i$; $k$
1: for $t = 0$ to $size(TS_i)$ do
2:     $CD_{t,j} \leftarrow$ Compute kNN ($TR_j, TS_i(x), k$)
3:     $result_j \leftarrow (< key : t, value : CD_{t,j} >)$
4:     EMIT($result_j$)
5: end for
```

Every map j will constitute a class-distance vector $CD_{t,j}$ of pairs $< class, distance >$ of dimension k for each test sample t in TS_i. To do so, Instruction 2 computes for each test sample the class and the distance to its k nearest neighbors. To accelerate the posterior actualization of the nearest neighbors in the reducers, every vector $CD_{t,j}$ is sorted in ascending order regarding the distance to the test sample, so that, $Dist(neigh_1) < Dist(neigh_2) < .... < Dist(neigh_k)$.

Unlike the MapReduce proposed in [28], every map sends multiple outputs, e.g. one per test instance. The vector $CD_{t,j}$ is outputted as value together with an identifier of test instance t as key (Instruction 3). In this way, we allow this method to use multiple reducers. Having more reducers may be useful when the used training and test datasets are very big.

3.1.2 Reduce Phase

The reduce phase consists of collecting, from the tentative k nearest neighbors provided by the maps, the closest ones for the examples contained in TS_i. After the map phase, all the elements with the same key have been grouped. A reducer is run over a list($CD_{t,0}, CD_{t,1}, .., CD_{t,m}$) and determines the k nearest neighbors of this test example t.

This function will process every element of such list one after another. Instructions 2 to 7 update a resulting list $results_{reducer}$ with the k neighbors. Since the vectors coming from the maps are ordered according to the distance, the update process becomes faster. This consists of merging two sorted lists up to get k values, so that, the complexity in the worst case is O(k). Therefore, this function compares every distance value of each of the neighbors one by one, starting with the closest neighbor. If the distance is lesser than the current value, the class and the distance of this position is updated with the corresponding values, otherwise we proceed with the following value. Algorithm 2 provides the details of the reduce operation.

Algorithm 2 Reduce by key operation

```
Require: $result_{key}$, $k$
1: cont=0
2: for $i = 0$ to $k$ do
3:     if $result_{key}(cont).Dist < result_{reducer}(i).Dist$ then
4:         $result_{reducer}(i) = result_{key}(cont)$
5:         cont++
6:     end if
7: end for
```

In summary, for every instance in the test set, the reduce function will aggregate the values according to function described before. To ease the implementation of this idea, we use the transformation *ReduceByKey(func)* from Spark. Algorithm 2 corresponds to the function required in Spark.

3.2 General scheme of kNN-IS

When the size of the test set is very large, we may exceed the memory allowance of the map tasks. In this case, we also have to split the test dataset and carry out multiple iterations of the MapReduce process defined above. Figure 2 depicts the general work-flow of the method. Algorithm 3 shows the pseudo-code of the whole method with precise details of the functions utilized in Spark. In the following, we describe the most significant instructions, enumerated from 1 to 13.

As input, we receive the path in the HDFS for both training and test data as well as the number of maps m and reducers r. We also dispose of the number of neighbors k and the memory allowance for each map.

Firstly, we create an RDD object with the training set TR formed by m blocks (Instruction 1). The test set TS is also read as an RDD without specifying a number of partitions. As this is read, we establish the key of every single test instance according to its position in the dataset (Instruction 2, function *zipWithIndex()* in Spark).

Since we will use Euclidean distance to compute the similarity between instances, normalizing both datasets becomes a mandatory task. Thus, Instructions 3 and 4 both perform a parallel operation to normalize the data into the range [0,1]. Both datasets are also cached for future reuse.

Even though Spark can be iteratively applied with the same data without incurring excessive time consumption, we reduce the number of iterations to a minimum because the fewer iterations there are, the better the performance will be. Instruction 5 calculates the minimum number of iterations $\#Iter$ that we will have to

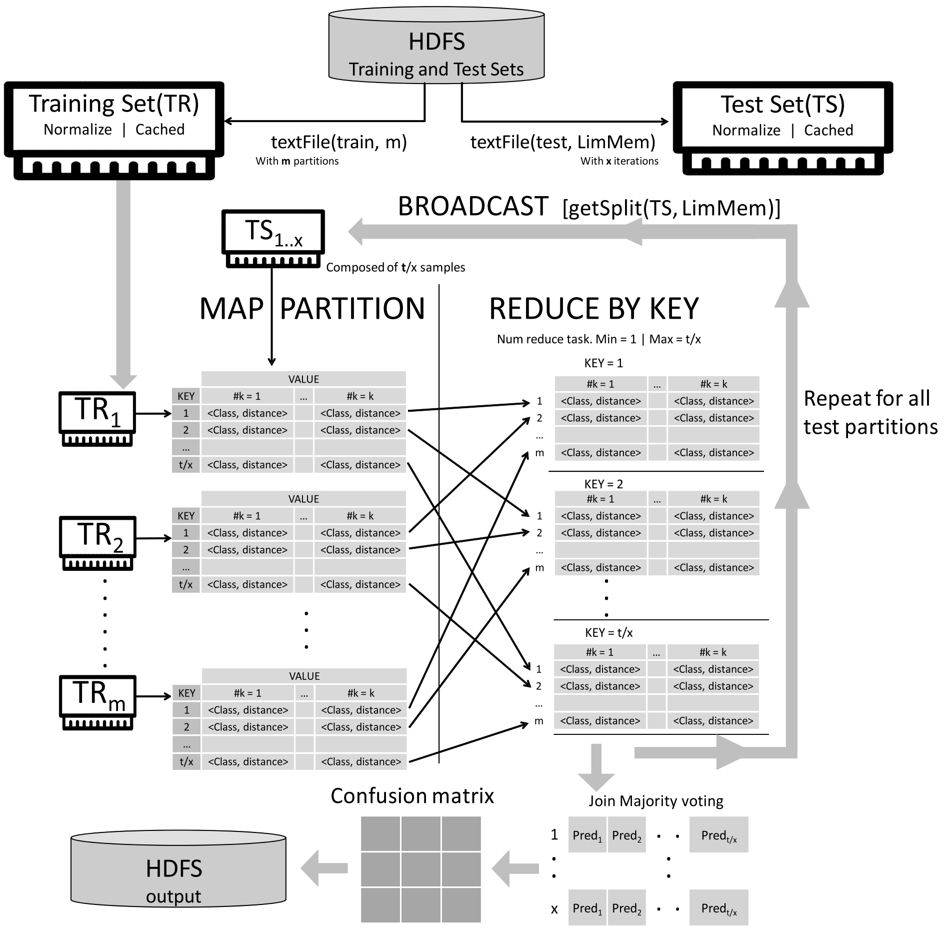


Figure 2: Flowchart of the proposed kNN-IS algorithm

perform to manage the input data. To do so, it will use the size of every chunk of the training dataset, the size of the test set and the memory allowance for each map.

With the computed number of iterations $\#Iter$, we can easily split the test dataset into subsets of a similar number of samples. To do that, we make use of the previously established keys in TS (in Instruction 2). Instruction 6 will perform the partitioning of the test dataset by using the function *RangePartitioner*.

Next, the algorithm enters into a loop in which we classify subsets of the test set (Instructions 7-12). Instruction 7 firstly gets the split corresponding to the current iteration. We use the transformation *filterByRange(lowKey, maxKey)* to efficiently take the corresponding subset. This function takes advantage of the split performed in Instruction 6, to only scan the matching elements. Then, we broadcast this subset TS_i into the main memory of all the computing nodes involved. The *broadcast* function of Spark allows us to keep a read-only variable cached on each machine rather than copying it with the tasks.

After that, the main map phase starts in Instruction 9. As stated before, the *mapPartition* function computes the kNN for each partitions of TR_j and TS_i and emits a pair RDD with key equals to the number of instance and value equals to a list of *class-distance*. The reduce phase joins the results of each map grouping by key (Instruction 9). As a result, we obtain the k neighbors with the smallest distance and their classes for each test input in TS_i. More details can be found in the previous section.

The last step in this loop collects the right and predicted classes storing them as an array in every iteration (Instruction 11).

Finally, when the loop is done, Instruction 13 computes the resulting confusion matrix and outputs the desired performance measures.

Algorithm 3 kNN-IS

```
Require: $TR$; $TS$; $k$; $\#Maps$; $\#Reduces$; $\#MemAllow$
1: $TR\text{-}RDD_{raw}$ ← textFile($TR$, $\#Maps$)
2: $TS\text{-}RDD_{raw}$ ← textFile($TS$).zipWithIndex()
3: $TR\text{-}RDD$ ← $TR\text{-}RDD_{raw}$.map(normalize).cache
4: $TS\text{-}RDD$ ← $TS\text{-}RDD_{raw}$.map(normalize).cache
5: $\#Iter$ ← callIter($TR\text{-}RDD$.weight(), $TS\text{-}RDD$.weight, $MemAllow$)
6: $TS\text{-}RDD$.RangePartitioner($\#Iter$)
7: for $i = 0$ to $\#Iter$ do
8:     $TS_i$ ← broadcast($TS\text{-}RDD$.getSplit(i))
9:     resultKNN ← $TR\text{-}RDD$.mapPartition($TR_j \rightarrow$ kNN($TR_j$, $TS_i$, $k$))
10:    result ← $resultKNN.reduceByKey(combineResult$,#Reduces$)$.collect
11:    right-predictedClasses[i] ← calculateRightPredicted(result)
12: end for
13: cm ← calculateConfusionMatrix(right-predictedClasses)
```

4 Experimental set-up

In this section, we show the factors and points related to the experimental study. We provide the performance measures used (Section 4.1), the details of the problems chosen for the experimentation (Section 4.2) and the involved methods with their respective parameters (Section 4.3). Finally, we specify the hardware and software resources that support our experiments (Section 4.4).

4.1 Performance measures

In this work we assess the performance and scalability with the following three measures:

- *Accuracy:* Represents the number of correct classifications against the total number of classified instances. This is calculated from a resulting confusion matrix, dividing the sum of the diagonal elements between the total of the elements of the confusion matrix. This is the most commonly used metric for assessing the performance of classifiers for years in standard classification ([38] [39]).

- *Runtime:* We will collect the total time spent by the kNN classifier to classify a given test set against the training dataset. Moreover, we will take intermediate times from the map phase and the reduce phase to better analyze the behavior of our proposal. The total runtime for the parallel approach includes reading and distributing all the data, in addition to calculating k nearest neighbors and majority vote.

- *Speed up:* Proves the efficiency of a parallel algorithm comparing against the sequential version of the algorithm. Thus, it measures the relation between the runtime of sequential and parallel versions. In a fully parallelism environment, the maximum theoretical speed up would be the same as the number of used cores, according to the Amdahl's Law [40].

$$Speedup = \frac{base\_line}{parallel\_time} \tag{1}$$

where $base\_line$ is the runtime spent with the sequential version and $parallel\_time$ is the total runtime achieved with its improved version.

4.2 Datasets

In this experimental study we will use four big data classification problems. PokerHand, Susy and Higgs are extracted from the UCI machine learning repository [41]. Moreover, we take an extra dataset that comes from the ECBDL'14 competition [42]. This is a highly imbalanced problem (Imbalanced ratio > 45), in which the kNN may be biased towards the negative class. Thus, we randomly sample said dataset to obtain more balance. The point of using said dataset, is that apart from containing a substantial number of instances, it has a relatively high number of features, so that, we can see how this fact affects the proposed model.

Table 1 summarizes the characteristics of these datasets. We show the number of examples ($\#Examples$), number of features ($\#Features$), and the number of classes ($\#\omega$). Note that with a fewer number of instances, the ECBLD'14 datasets become the larger datasets in terms of size because of its number of features.

Table 1: Summary description of the used datasets

Dataset	#Examples	#Features	$\#\omega$
PokerHand	1,025,010	10	10
ECBDL'14	2,063,187	631	2
Susy	5,000,000	18	2
Higgs	11,000,000	28	2

For the experimental study all datasets have been partitioned using a 5 fold cross-validation (5-fcv) scheme. It means that the dataset is partitioned into 5 folds, each one including 80% training samples and the rest test examples. For each fold, the kNN algorithm computes the nearest neighbors from the TS against TR.

In the presented MapReduce scheme, the number of instances of a dataset and the number of maps used have a direct relation, so that, the greater the number of maps is, the fewer number of instances there are in them. Table 2 presents the number of instances in each training set according to the number of maps used. In italics we represent the settings that are not used in our experiments because there are either too few instances or too many.

Table 2: Approximate number of instances in the training subset depending on the number of mappers

Dataset	**Number of maps**						
	32	64	128	256	512	1024	2048
PokerHand	25,626	12,813	6,406	*3,203*	*1,602*	*800*	*400*
ECBDL'14	*51,580*	*25,790*	12,895	6,448	3,224	1,612	*806*
Susy	*62,468*	31,234	15,617	7,809	3,905	1,953	976
Higgs	*275,000*	*137,500*	68,750	34,375	17,188	8,594	4,297

The number of reducers also plays an important role in how the test dataset is managed in kNN-IS. The larger the number of reducers, the smaller the number of test instances that have to be processed for each reducer. Table 3 shows this relation, assuming that the test set is not split because of memory restrictions (so, number of iterations = 1). Once again, we point out in italics those settings that have not been explored.

Table 3: Approximate number of instances in the test subset depending on the number of reducers

Dataset	**Number of reducers**			
	1	32	64	128
PokerHand	205,002	*102,501*	*51,250*	*25,625*
ECBDL'14	412,637	12,895	6,448	*3,224*
Susy	1,000,000	31,250	15,625	*7,813*
Higgs	2,200,000	*68,750*	34,375	17,188

4.3 Methods and Parameters

Among the existing distributed kNN models based on MapReduce, we establish a comparison with the model proposed in [28], MR-kNN, as the most promising alternative proposed so far, which is based in Hadoop MapReduce.

As stated in Section 2.2, kNN-join methods [36] were originally designed for other purposes rather than classification. They also require the data size to increase and even a squared number of Map tasks. Therefore, their theoretical complexity is so much higher than the proposed technique that we have discarded a comparison of such models, as it would be very time consuming.

We have also conducted preliminary experiments in order to apply the iterative method proposed iHMR-kNN [27]. However, the iterative processing becomes so slow that we have not been able to apply it to any of the datasets considered in a timely manner.

This work is mainly devoted to testing the scalability capabilities of the proposed model, showing how it palliates the weaknesses of previously proposed models stated in Section 2.2. To do so, we will analyze the effect of the number of neighbors, and the number of maps and reducers. Table 4 summarizes the parameters used for both MR-kNN and kNN-IS models.

Table 4: Parameter settings for the used methods.

Method	**Parameter values**
MR-kNN [28]	k=1,3,5,7; Number Of Maps = 32/64/128; Number Of Reducers:1
	Implementation: Hadoop MapReduce; Euclidean Distance
kNN-IS	k=1,3,5,7; Number Of Maps = 32/64/128/256/512/1024/2048;
	Number Of Reducers: 1/32/64/128
	Implementation: Spark; Euclidean Distance
	Memory allowance per Map: 2GB
	Multiple iterations: Automatically determined or Fixed.

4.4 Hardware and software used

All the experiments have been executed on a cluster which is composed of sixteen nodes: the master node and sixteen computing nodes. All the nodes have the following features:

- Processors: 2x Intel Xeon CPU E5-2620

- Cores: 6 cores (12 threads)
- Clock speed: 2 GHz
- Cache: 15 MB
- Network: Infiniband (40Gb/s)
- RAM: 64 GB

The specific details of the software used and its configuration are the following:

- MapReduce implementations: Hadoop 2.6.0-cdh5.4.2 and Spark 1.5.1
- Maximum number of map tasks: 256
- Maximum number of reduce tasks: 128
- Maximum memory per task: 2GB.
- Operating System: Cent OS 6.5

Note that the total number of available cores is 192, which becomes 384 by using hyper-threading technology. Thus, when we explore a number of maps greater than 384, we cannot expect linear speedups, since there will be queued tasks. For these cases, we will focus on analyzing the map and reduce runtimes.

5 Analysis of results

In this section, we study the results collected from different experimental studies. Specifically, we analyze the next four points:

- First, we establish a comparison between kNN-IS and MR-kNN (Section 5.1).
- Second, we deeply analyze the influence of the number of neighbors k value in the performance proposed model (Section 5.2).
- Third, we check the impact of the number of reducers in relation to the number of maps when tackling very large datasets (Section 5.3).
- Finally, we study the behavior of kNN-IS with huge test datasets, in which the method is obliged to perform multiple iterations (Section 5.4).

5.1 Comparison with MR-kNN

This section compares kNN-IS with MR-kNN, as the potentially fastest alternative proposed so far. To do this, we make use of PokerHand and Susy datasets. We could not go further than these datasets in order to obtain the results of the sequential kNN. In these datasets, kNN-IS only needs to conduct one iteration, since the test datasets fits in the memory in a every map. The number of reducers in kNN-IS has been also fixed to 1, to establish a comparison between very similar MapReduce alternatives under Hadoop (MR-kNN) or Spark (kNN-IS).

First of all, we run the sequential version of kNN over these datasets as a baseline. As in [28], this sequential version reads the test set line by line, as done by MR-kNN, as a straightforward solution to avoid memory problems. We understand that this scenario corresponds to the worst possible case for the sequential version, and better sequential versions could be designed. However, our aim here is to compare with the simplest sequential version, assuming that large test sets do not fit in memory together with the training set.

Table 5 shows the runtime (in seconds) and the average accuracy (AccTest) results obtained by the standard kNN algorithm, depending on the number of neighbors.

Table 5: Sequential kNN performance

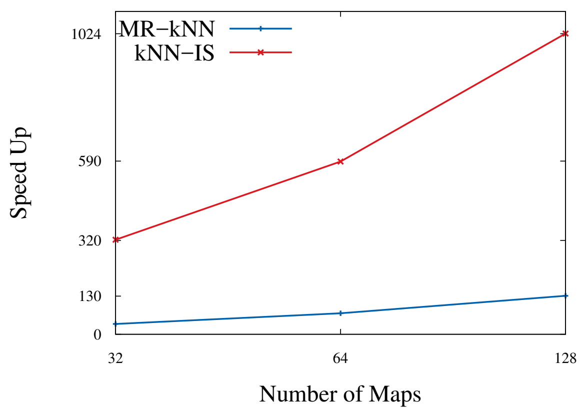
Dataset	Number of Neighbors	Runtime(s)	AccTest
PokerHand	1	105475.0060	0.5019
	3	105507.8470	0.4959
	5	105677.1990	0.5280
	7	107735.0380	0.5386
Susy	1	3258848.8114	0.6936
	3	3259619.4959	0.7239
	5	3265185.9036	0.7338
	7	3325338.1457	0.7379

Table 6 summarizes the results obtained with both methods with k=1. The next Section will detail the influence of the value of k. It shows, for each number of maps (#Maps) the average total time (AvgRuntime) and the speedup achieved against the sequential version. As stated before, both methods correspond to exact implementation of the kNN, so that, we obtain exactly the same average accuracy as presented in Table 5.

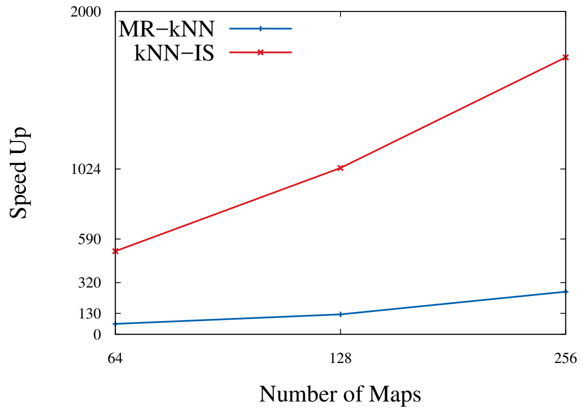
Table 6: Results obtained by MR-kNN and kNN-IS algorithms in PokerHand dataset

Dataset	#Map	**MR-kNN**		**kNN-IS**	
		AvgRunTime	Speedup	AvgRuntime	Speedup
PokerHand	128	804.4560	131.1135	102.9380	1024.6460
	64	1470.9524	71.7052	179.2381	588.4631
	32	3003.3630	35.1190	327.5347	322.0270
Susy	256	12367.9657	263.4911	1900.0393	1715.1481
	128	26438.5201	123.2614	3163.9710	1029.9869
	64	50417.4493	64.6373	6332.8108	514.5975

Figure 3 plots speed up comparisons of both approaches against the sequential version as the number of maps is increased (k = 1).



(a) Poker-Hand



(b) Susy

Figure 3: Speedup comparisons between MR-kNN and kNN-IS against the sequential kNN

According to all these tables and figures, we can make the following analysis:

- As we can observe in Table 5, that the required runtime for this sequential version of the kNN method is considerably high in both datasets. However, Table 6 shows how this runtime can be greatly reduced in both approaches as the number of maps is increased. As stated above, both alternatives always provide the same accuracy as the sequential version.

- According to Figure 3, a linear speed up for the hadoop-based kNN model has been achieved since both models read the test dataset set line-by-line, which is sometimes even superlinear what is related to memory-consumption problems of the original kNN model to manage the training set. However, kNN-IS presents a faster speed up than a linear speed up in respect to this sequential version. This is because of the use of in-memory data structures that allowed us to avoid reading test data from HDFS line-by-line.

- Comparing MR-kNN and kNN-IS, the results show how Spark has allowed us to reduce the runtime needed almost 10-fold in comparison to Hadoop.

5.2 Influence of the number of neighbors

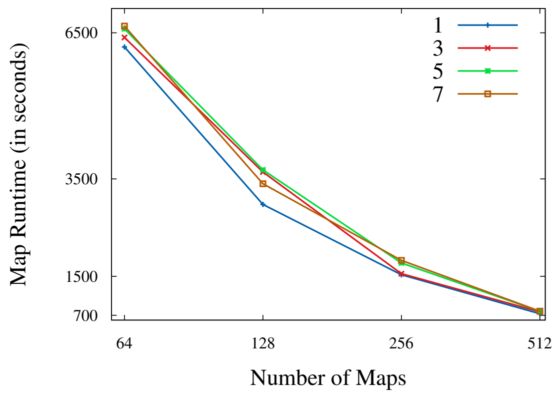
To deeply analyze the influence of the number of neighbors we focus on the Susy dataset, and we set the number of reducer tasks to one again. We analyze its effect in both map and reduce phases.

Table 7 collects for each number of neighbors (#Neigh) and number of maps (#Maps), the average map execution time (AvgMapTime), the average reduce time (AvgRedTime) and the average total runtime (AvgTotalTime). Recall that in our cluster the maximum number of map tasks is set to 256. Thus, the total runtime for 512 maps does not show a linear reduction, but it can be appreciated in the reduction of the map runtime.

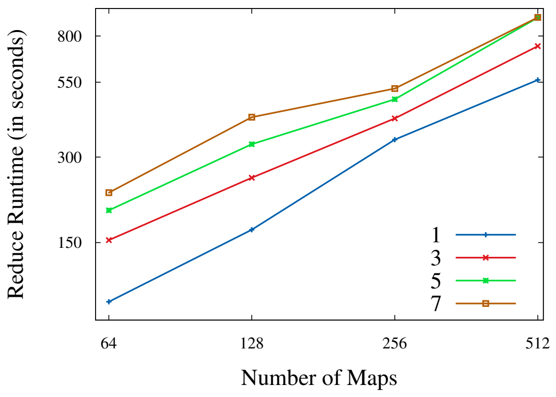
Table 7: Results obtained with Susy dataset

k	#Map	AvgMapTime	AvgRedTime	AvgTotalTime
1	512	730.3893	560.4334	2042.2533
	256	1531.6145	345.5664	1900.0393
	128	2975.3013	166.3976	3163.9710
	64	6210.6177	92.8188	6332.8108
3	512	770.3924	736.8489	2298.3384
	256	1553.4222	410.2235	2615.0150
	128	3641.9363	253.5656	3921.3640
	64	6405.3132	152.9890	6593.5531
5	512	781.3855	928.2620	2511.8909
	256	1773.3579	479.0801	2273.6377
	128	3685.3194	332.9783	4042.1755
	64	6582.0373	194.7054	6802.8159
7	512	782.5756	930.5107	2516.5011
	256	1827.9189	522.6219	2372.4100
	128	3401.2547	414.2961	3838.2360
	64	6637.8837	224.7191	6890.8242

Figure 4a presents how the value of k influences in map runtimes. It depicts the map runtimes in terms of number of maps for $k = 1, 3, 5$ and 7. Figure 4b plots the reduce runtime in relation to the k value and number of maps.



(a) Influence of parameter k in the map phase



(b) Influence of parameter k in the reduce phase.

Figure 4: Result of Susy dataset

According to these tables and plots, we can conclude that:

- Even though larger values of k imply that the data transferred from the maps to the reducers is bigger, this value does not drastically affect the total runtimes. In Table 7, we can appreciate that, in general, the total runtime slightly increments.
- Comparing Figures 4a and 4b, we can see that the number of neighbors seem to have more influence on the reduce runtime than on the map phase. This is because the number of neighbors does not affect the main computation cost (computing the distances between test and training instances) of the map phase, while it may affect the updating process performed in the reducers since its complexity is $O(k)$.

Finally, as a general appreciation, Figure 4b reveals that when a larger number of maps is used, which is clearly necessary to deal big datasets, the reduce runtimes increase considerably. This has motivated the study carried out in the next Section.

5.3 Influence of the number of reducers

As we just saw in the previous section, a high number of maps greatly increases the load of the reduce phase. However, the use of a large number of maps may be absolutely necessary to tackle very big datasets. This section investigates how the proposed idea of managing different test instances in multiple reducers may help to alleviate such an issue.

In this experiment, we involve the three biggest datasets: ECBLD'14, Susy, Higgs. Once again, kNN-IS does not require multiple iterations for these test datasets' sizes. To be concise, in this study we only focus on k=1.

Figure 5 plots the reduce time required with a single reducer for all these problems. It confirms, as pointed out in the previous section, the drastic increment when the number of maps is very high. It is actually even more accentuated as Higgs and ECBLD'14 are larger datasets that require a greater number of maps.

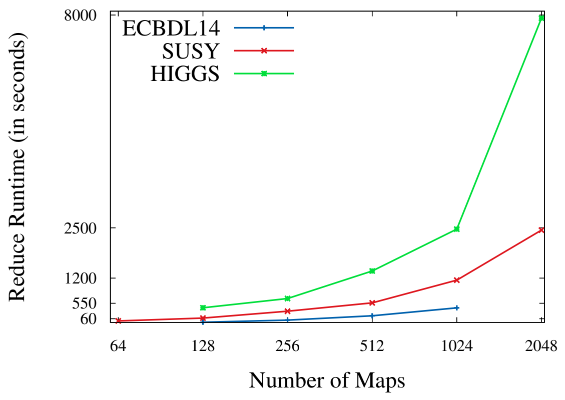


Figure 5: Reduce runtime required against the number of map tasks, k=1, Number of reducers = 1.

For sake of clarity, we do not present the associated tables of results for the three considered problems, but we visually present such results in Figures 7a, 7b and 7c. These figures plot the map and reduce runtimes spent in ECBLD'14, Susy and Higgs, respectively, in terms of the number of maps and reduces utilized.

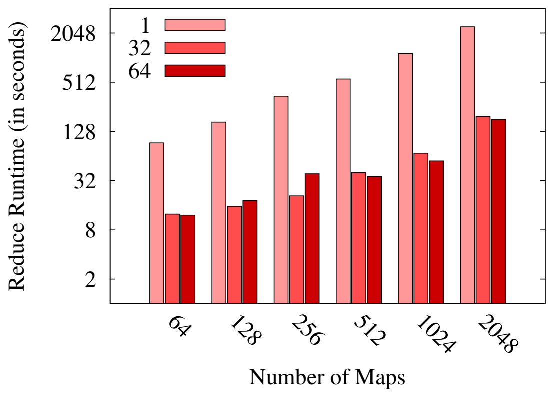


Figure 6: Reduce Runtime vs. number of maps and reducers | SUSY

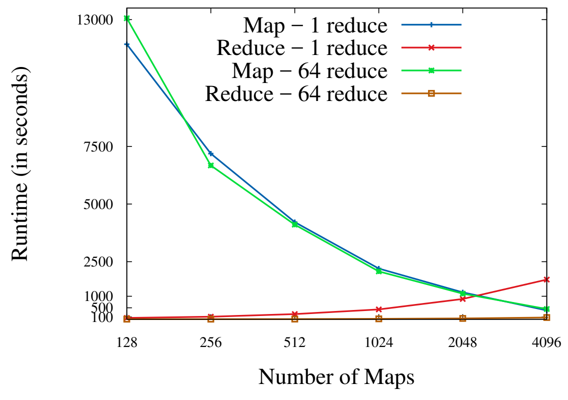
These figures reveal that:

- Using multiple reducers effectively softens the runtime spent in the reduce phase when it is necessary to use a larger number of maps. In the previous plots, we can see how the version with a single reducer rapidly increases its computational cost in comparison to the version with more reducers.

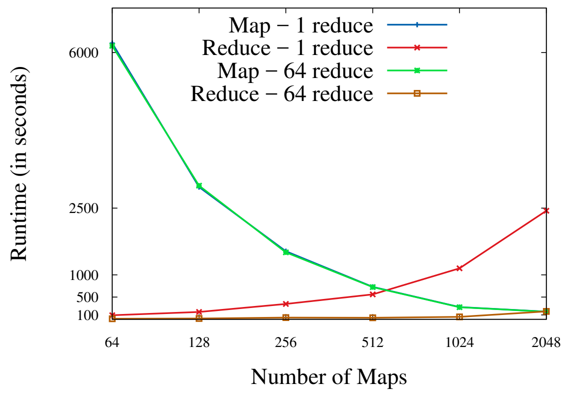
- The reduction in the required time is not linear in respect to the number of reducers. As pointed out in [43], an excessive number of reducers can also lead to a MapReduce overhead in the network. As we can see, for example in Figure 6, there are no great differences when using 32 or 64 reducers.

- The use of multiple reducers is devised to use with a high number of maps. Otherwise, its behavior may damage the efficiency. For example, for the Susy dataset, it is not convenient to use more than 32 reducers unless we have more than 512 maps.

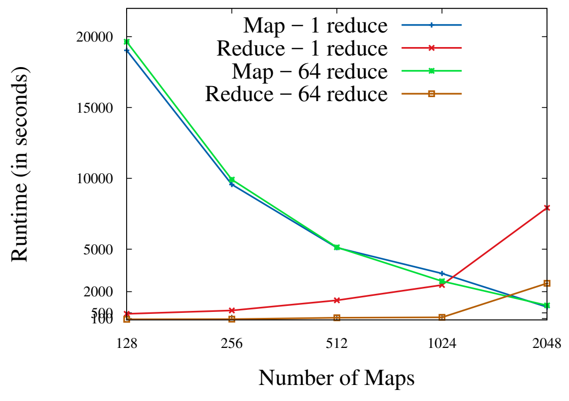
In conclusion, it is important to find a trade-off between the number of maps and reducers according to the cluster and the dataset that we dispose.



(a) ECBDL’14



(b) SUSY



(c) HIGGS

Figure 7: Map and Reduce runtimes required according to the number of maps

5.4 Dealing with large amounts of test data

To test the behavior of the full model presented here, we carry out a study in which both training and test sets are composed of the same number of instances. In this way, we ensure that kNN-IS is obliged to perform multiple iterations. To do so, we test training versus training datasets.

Table 8 presents the results of the three datasets with more than one iteration (#Iter), average reduce runtime (AvgRedTime) and the average total runtime (AvgTotalTime). To study the influence of test size, we focus on k=1, with 256 maps, 64 reduces for Susy and ECBDL'14 datasets and 128 reduce tasks for the Higgs dataset (#Red).

Table 8: Results obtained with more than one iteration.

Dataset	#Iter	AvgMapTime	AvgRedTime	AvgTotalTime
ECBDL'14	3	7309.7122	8.8911	28673.7015
#Red=64	5	4303.7106	5.2520	28918.2992
	10	2027.5685	2.8076	29121.0583
SUSY	2	2385.6183	34.2682	6723.7762
#Red=64	5	1156.9453	14.1350	9493.5098
	10	649.7218	5.9823	10278.2612
HIGGS	2	16835.6982	144.3371	44414.1423
#Red=128	5	7145.7838	59.3202	46806.8294
	10	3668.4418	29.7266	51836.9468

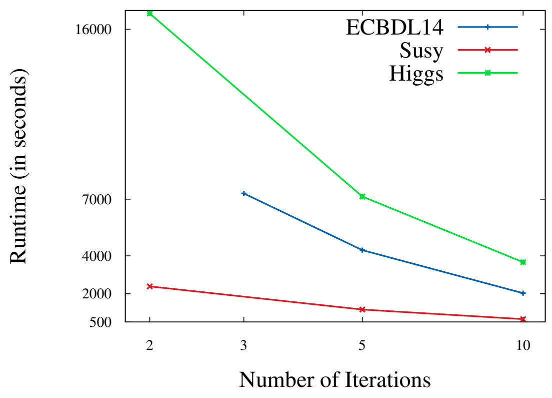
Figure 8 presents the influence of the number of iterations. The ECBDL'14 dataset needs 3 iterations to fit the main memory. The other datasets only need 2 iterations. Figure 8a shows the map time with a different number of iterations for the three datasets used. Figure 8b presents how the number of iterations influences the reduce runtimes and Figure 8c plots the total runtime versus the number of iterations.

Analyzing these tables and plots, we can observe that:

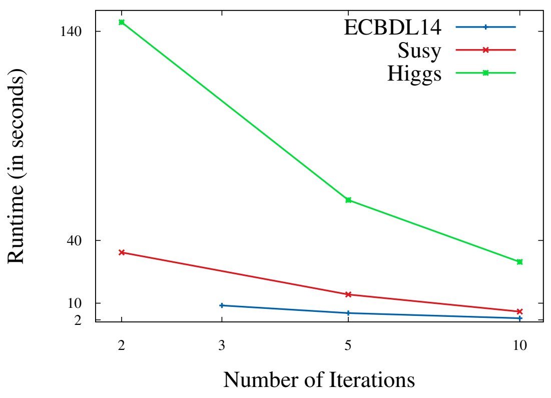
- As Figures 11 and 12 show and as we can expected, when more than one iteration is used, the map and reduce runtimes decrease. This occurs because the number of instances to be calculated on each core are less than a simple iteration.

- However, Figure 8c shows how it slightly increases the total runtime. This behavior could be caused by a network saturation of the cluster. For ECBDL'14 dataset, the total runtime increases less than other two datasets. This happens because it has fewer samples as shown in Table 1. Thus, it produces less network traffic in spite of having more features.

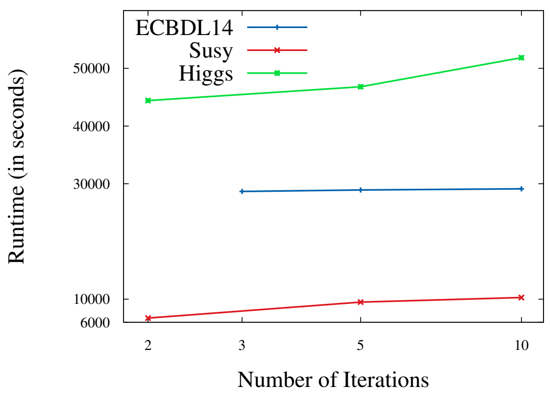
In conclusion, the iterative functionality of kNN-IS has to be used when the size of datasets exceeds the available memory of a core of the cluster because it becomes slower in total runtimes and network traffic is increased.



(a) Map Runtime



(b) Reduce Runtime



(c) Total Runtime

Figure 8: Runtimes vs Number of Iterations

6 Conclusions and further work

In this paper we have developed a Iterative MapReduce solution for the k-Nearest Neighbors algorithm based on Spark. It is denominated as kNN-IS. The proposed scheme is an exact model of the kNN algorithm that we have enabled to apply with large-datasets. Thus, kNN-IS obtains the same accuracy as kNN. However, the kNN algorithm has two main issues when dealing with large-scale data: Runtime and Memory consumption. The use of Apache Spark has provided us with a simple, transparent and efficient environment to parallelize the kNN algorithm as an iterative MapReduce process.

The experimental study carried out has shown that kNN-IS obtains a very competitive runtime. We have tested its behavior with datasets of different sizes (different number of features and different number of samples).

The main achievements obtained are the following:

- kNN-IS is an exact parallel approach and obtains the same accuracy and very good achievements on runtimes.

- kNN-IS (Spark) has allowed us to reduce the runtime needed by almost 10 times in comparison to MR-kNN (Hadoop).
- Despite producing more transfer from the map to reduce, the number of neighbors (k) does not drastically affect to the total runtime.
- We can optimize the runtime with a trade-off between the number of maps and reducers according to the cluster and the dataset used
- When datasets are enormous and it exceed the memory capacity of the cluster, kNN-IS calculates the solution with more than one iteration by splitting the test set. Therefore, it has allowed us to apply the kNN algorithm in large-scale problems.
- The software of this contribution can be found as a spark-package at `http://spark-packages.org/package/JMailloH/kNN_IS`. The source code of this technique can be found in the next repository `https://github.com/JMailloH/kNN_IS`

As future work, we aim to tackle big datasets that contain missing values [44] by using kNN-IS to impute them, and datasets with a very large number of features by using multi-view approaches. We are planning to extend the use of kNN-IS to instance selection techniques for big data [45], where it reports good results. Another direction for future work is to extend the application of the presented kNN-IS approach to a big data semi-supervised learning [46] context.

APPENDIX

As consequence of this work, we have developed a Spark package with the kNN-IS implementation. It has all the functionalities exposed in this study. In addition, we have developed kNN-IS for the machine learning library on Spark, as part of the MLlib library and the MLbase platform.

Prerequisites: You must have Spark 1.5.1, Scala 2.10 and Maven 3.3.3 or higher installed. Java Virtual Machine 1.7.0 is necessary because Scala runs over it.

The implementation allows us to determine the:

- Number of Cores to be used: Number of cores to compute the MapReduce approach.
- Number of neighbors: Number of neighbors. The value of k.
- Number of maps: Number of map tasks.
- Number of reduces: Number of reduce tasks.
- Number of iterations: Number of iterations. Setting to -1 to auto-setting the iterations. We give optional parameter (*Maximum memory per node*) limit on GB for each map task. This selects the minimum number of iterations within the limit provided.

The input data is expected to be in KEEL Dataset format [47]. The datasets are previously stored in HDFS.

The output will be stored in HDFS in the following format: *./outputPath*/Predictions.txt/part-00000 contains the predicted and right class in two column. *./outputPath*/Result.txt/part-00000 shows confusion matrix, accuracy and total runtime. *./outputPath*/Times.txt/part-00000 presents higher map time, higher reduce time, average iterative time and total runtime.

For more details, please refer to the README in the GitHub repository: `https://github.com/JMailloH/kNN_IS/blob/master/README.md`

The proposed kNN-IS is now available as a Spark Package at `http://spark-packages.org/package/JMailloH/kNN_IS`

Acknowledgments

This work has been supported by the Spanish National Research Project TIN2014-57251-P and the Andalusian Research Plan P11-TIC-7765. J. Maillo and S. Ramírez hold FPU scholarships from the Spanish Ministry of Education. I. Triguero held a BOF postdoctoral fellowship from Ghent University during part of the development of this work.

References

[1] Clifford Lynch. Big data: How do your data grow? *Nature*, 455(7209):28–29, 2008.

[2] Michael Minelli, Michele Chambers, and Ambiga Dhiraj. *Big Data, Big Analytics: Emerging Business Intelligence and Analytic Trends for Today's Businesses (Wiley CIO)*. Wiley Publishing, 1st edition, 2013.

[3] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[4] Xindong Wu and Vipin Kumar, editors. *The Top Ten Algorithms in Data Mining*. Chapman & Hall/CRC Data Mining and Knowledge Discovery, 2009.

[5] Yihua Chen, Eric K. Garcia, Maya R. Gupta, Ali Rahimi, and Luca Cazzanti. Similarity-based classification: Concepts and algorithms. *Journal of Machine Learning Research*, 10:747–776, 2009.

[6] Kilian Q. Weinberger and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, 2009.

[7] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, November 1998.

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

[9] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, 2003.

[10] 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.

[11] Zhenhua Guo, G. Fox, and Mo Zhou. Investigation of data locality in mapreduce. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 419–426, May 2012.

[12] Ashwin Srinivasan, TanveerA. Faruquie, and Sachindra Joshi. Data and task parallelism in ILP using mapreduce. *Machine Learning*, 86(1):141–168, 2012.

[13] 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. *Know.-Based Syst.*, 87(C):69–79, October 2015.

[14] X. Yan, J. Zhang, Y. Xun, and X. Qin. A parallel algorithm for mining constrained frequent patterns using mapreduce. *Soft Computing*, pages 1–13, 2015.

[15] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999.

[16] A. Fernández, S. Río, V. López, A. Bawakid, M.J. del Jesus, J.M. Benítez, and F. Herrera. Big data with cloud computing: An insight on the computing environment, mapreduce and programming frameworks. *WIREs Data Mining and Knowledge Discovery*, 4(5):380–409, 2014.

[17] K. Grolinger, M. Hayes, W.A. Higashino, A. L'Heureux, D.S. Allison, and M.A.M. Capretz. Challenges for mapreduce in big data. In *Services (SERVICES), 2014 IEEE World Congress on*, pages 182–189, June 2014.

[18] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Graphlab: A new parallel framework for machine learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, Catalina Island, California, July 2010.

[19] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. Haloop: Efficient iterative data processing on large clusters. *Proc. VLDB Endow.*, 3(1-2):285–296, September 2010.

[20] 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*, pages 1–14. USENIX Association, 2012.

[21] Mohammed Ghesmoune, Mustapha Lebbah, and Hanene Azzag. Micro-batching growing neural gas for clustering data streams using spark streaming. *Procedia Computer Science*, 53:158 – 166, 2015. INNS Conference on Big Data 2015 Program San Francisco, CA, USA 8-10 August 2015.

[22] Isaac Triguero, Daniel Peralta, Jaume Bacardit, Salvador García, and Francisco Herrera. MRPR: A mapreduce solution for prototype reduction in big data classification. *Neurocomputing*, 150, Part A(0):331 – 345, 2015.

[23] S. Ding M. Du and H. Jia. Study on density peaks clustering based on k-nearest neighbors and principal component analysis. *Knowledge-Based Systems*, 99:135 – 145, 2016.

[24] D. Cheng M. Zong Z. Deng, X. Zhu and S. Zhang. Efficient knn classification algorithm for big data. *Neurocomputing*, 195:143 – 148, 2016. Learning for Medical Imaging.

[25] Chi Zhang, Feifei Li, and Jeffrey Jestes. Efficient parallel knn joins for large data in mapreduce. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 38–49, New York, NY, USA, 2012. ACM.

[26] G. Chatzimilioudis, C. Costa, D. Zeinalipour-Yazti, W. C. Lee, and E. Pitoura. Distributed in-memory processing of all k nearest neighbor queries. *IEEE Transactions on Knowledge and Data Engineering*, 28(4):925–938, April 2016.

[27] Kai Sun, Hoon Kang, and Ho-Hyun Park. Tagging and classifying facial images in cloud environments based on kNN using mapreduce. *Optik - International Journal for Light and Electron Optics*, 126(21):3227 – 3233, 2015.

[28] J. Maillo, I. Triguero, and F. Herrera. A mapreduce-based k-nearest neighbor approach for big data classification. In *9th International Conference on Big Data Science and Engineering (IEEE BigDataSE-15)*, pages 167–172, 2015.

[29] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 3rd edition, 2012.

[30] Apache Hadoop Project. Apache hadoop, 2015.

[31] 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.

[32] Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia. *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, Incorporated, 2015.

[33] Apache Spark. Apache Spark: Lightning-fast cluster computing, 2015. [Online; accessed July 2015].

[34] Apache Spark. Machine Learning Library (MLlib) for Spark, 2016. [Online; accessed May 2016].

[35] Apache Spark. Machine Learning Library (ML) for Spark, 2016. [Online; accessed May 2016].

[36] Ge Song, J. Rochas, F. Huet, and F. Magoules. Solutions for processing k nearest neighbor joins for massive data on mapreduce. In *Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on*, pages 279–287, March 2015.

[37] Wei Lu, Yanyan Shen, Su Chen, and Beng Chin Ooi. Efficient processing of k nearest neighbor joins using mapreduce. *Proc. VLDB Endow.*, 5(10):1016–1027, June 2012.

[38] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[39] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[40] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM.

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

[42] ECBDL14 dataset: Protein structure prediction and contact map for the ECBDL2014 big data competition, 2014.

[43] C.-T. Chu, S.K. Kim, Y.-A. Lin, Y.Y. Yu, G. Bradski, A.Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems*, pages 281–288, 2007.

[44] Julián Luengo, Salvador García, and Francisco Herrera. On the choice of the best imputation methods for missing values considering three groups of classification methods. *Knowledge and Information Systems*, 32(1):77–108, 2011.

[45] J.J. Rodríguez Á. Arnaiz-González, J.F. Díez-Pastor and C. García-Osorio. Instance selection of linear complexity for big data. *Knowledge-Based Systems*, pages –, 2016.

[46] Isaac Triguero, Salvador García, and Francisco Herrera. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information Systems*, 42(2):245–284, 2013.

[47] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera. KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2011.

2 Fast and Scalable Approaches to Accelerate the Fuzzy k Nearest Neighbors Classifier for Big Data

- J. Maillo, J. Luengo, S. García, F. Herrera, I. Triguero. Fast and Scalable Approaches to Accelerate the Fuzzy k Nearest Neighbors Classifier for Big Data. IEEE Transactions on Fuzzy Systems. Accepted 2019 (In press) DOI:10.1109/TFUZZ.2019.2936356.
  - Status: **Accepted**.
  - Impact Factor (JCR 2018): **8.759**
  - Subject Category: **Computer Science, Artificial Intelligence**
  - Rank: **6/133**
  - Quartile: **Q1**

FAST AND SCALABLE APPROACHES TO ACCELERATE THE FUZZY k NEAREST NEIGHBORS CLASSIFIER FOR BIG DATA

Jesus Maillo
Department of Computer Science
and Artificial Intelligence
University of Granada, Granada, Spain 18071
jesusmh@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

Isaac Triguero
Automated Scheduling, Optimisation and Planning Research Group
School of Computer Science, University of Nottingham
Jubilee Campus, Nottingham NG8 1BB, United Kingdom
Isaac.Triguero@nottingham.ac.uk

ABSTRACT

One of the best-known and most effective methods in supervised classification is the k nearest neighbors algorithm (kNN). Several approaches have been proposed to improve its accuracy, where fuzzy approaches prove to be among the most successful, highlighting the classical Fuzzy k-nearest neighbors (FkNN). However, these traditional algorithms fail to tackle the large amounts of data that are available today. There are multiple alternatives to enable kNN classification in big datasets, spotlighting the approximate version of kNN called Hybrid Spill Tree. Nevertheless, the existing proposals of FkNN for big data problems are not fully scalable, because a high computational load is required to obtain the same behavior as the original FkNN algorithm. This work proposes Global Approximate Hybrid Spill Tree FkNN and Local Hybrid Spill Tree FkNN, two approximate approaches that speed up runtime without losing quality in the classification process. The experimentation compares various FkNN approaches for big data with datasets of up to 11 million instances. The results show an improvement in runtime and accuracy over literature algorithms.

***K*eywords** Fuzzy sets · K-nearest neighbors · Classification · MapReduce · Apache Spark · Big Data

1 Introduction

The Fuzzy k Nearest Neighbor algorithm (FkNN) [1] is developed with the aim of improving and alleviating the main weakness of the k Nearest Neighbor algorithm (kNN) [2]. This weakness resides in considering all neighbors as equally important in the classification, making the kNN algorithm more vulnerable to noise at the class boundaries, leading to a downgrading of the classification.

In the experimental analysis at [3], the classic algorithm FkNN stands out as one of the most effective approaches. FkNN is composed of two stages: class membership degree and classification. The first stage changes the label of the class by a vector of membership degree belonging to each class, according to the closest training instances. To calculate the nearest instances, it uses a similarity function, usually with a distance function (Euclidean or Manhattan). The second stage calculates the kNN with the information of the membership degree. Thus, it is possible to detect borders with greater precision, being less affected by noise and improving the kNN in most classification problems used in many applications such as medicine [4], spacecraft [5], and many other fields.

Nowadays, FkNN and kNN are used in many areas of data mining. They are used as data preprocessing techniques [6] to deal with imperfect data [7] and uncertainty in the classification process by means of aggregation operators [8]. Studies to improve the FkNN algorithm and its applications continue to develop in many areas such as convergence [9] and runtime improvement [10]. There are some recent proposals that enhance quality of the classic FkNN classifier, two proposals based on evolutionary algorithms [11] and [12] and one proposal based on parameter independent fuzzy weighted kNN [13]. Nevertheless, these solutions used to increase the computational complexity, making the algorithm less scalable for the application in big data problems. For this reason, we will focus on the classical FkNN algorithm.

In the big data environment [14], the kNN and FkNN algorithms have been key to solving different machine learning problems such as fuzzy-rough based NN classification [15], time-series forecasting [16] or data preprocessing to obtain quality data [17]. In this work, we are focused on standard classification. When handling large datasets the kNN and FkNN classifiers have problems regarding runtime and memory consumption. There is an exact proposal of the kNN algorithm to address big data problems and it is called *k Nearest Neighbor - Iterative Spark* (kNN-IS) [18]. In addition to this exact version, there are also approximate variations that drastically reduce execution times: Metric-Tree [19] and Spill-Tree. In [20], the authors studied the Metric-Tree and Spill-Tree models and proposed the Hybrid Spill-Tree model [21] (HS). HS is the hybridization of the two models with the aim of improving the runtime in big data.

Regarding the fuzzy approach, in [22], we investigate the feasibility of an exact approach to apply FkNN in big data called Global Exact Fuzzy k Nearest Neighbors (GE-FkNN) [22]. Even though it is able of scaling up to large datasets, the runtime of the first stage are substantially high, causing a bottleneck. Subsequently, the authors of contribution [23] present a preliminary study on the use of approximate kNN search to accelerate the execution time and alleviate the bottleneck.

The objective of this work is to design and develop a FkNN model capable of handling large datasets accurately and quickly. To do this, we use the Spark framework and use HS as the base algorithm due to its balance between scalability and accuracy that improves previous kNN proposals in the literature. The proposed algorithm is composed of the same two stages of classical FkNN: membership degree and classification. The main difference of the proposed algorithm can be noted in the first stage, focusing on handling the bottleneck with two different approaches:

- Local Hybrid Spill Tree FkNN (LHS-FkNN): The local approach divides the dataset into different parts and calculates the class membership degree internally in each partition, without considering other partitions.

- Global Approximate Hybrid Spill Tree (GAHS-FkNN): The global approach is based on the HS model. It generates a tree with the instances of the training set and distributes it among all the computation nodes, considering all the instances for the calculation of the class membership degree.

The second stage classifies the unseen samples from the test set using the class membership degree knowledge calculated in the first stage. The classification stage is the same for both models, following a HS based approach and with a workflow similar to the first stage of GAHS-FkNN. The novelty of the proposal is the use of approximate kNN searches, presenting local and global approaches, achieving quality accuracy and scalability that allows execution with large datasets through the use of the MapReduce [24] paradigm and the Spark framework [25].

In order to study the performance of this model, experiments have been carried out on 8 datasets with up to 11 million instances and 631 features. The experimental study analyzes the accuracy and runtime making a comparison with existing algorithms of the literature.

In addition, we have developed a software package with FkNN algorithms for big data, making use of in-memory native operations and distributed computing from Apache Spark. The developed algorithms can be found in the repository `https://spark-packages.org/package/JMailloH/HS_FkNN`.

The paper is structured in the following five sections. Section 2 introduces the state of the art in the FkNN and Hybrid Spill-Tree algorithms. Next, Section 3 details the proposals of the FkNN algorithm. Section 4 describes the experimental study and Section 5 includes multiple analyses of results. The Section 6 concludes the document and outlines future work.

2 Preliminaries

This section provides background knowledge of the FkNN algorithm (Section 2.1), the Hybrid Spill-Tree (Section 2.2) and the big data technologies used (Section 2.3).

2.1 Fuzzy k nearest neighbors and its computational complexity

FkNN needs a pre-computation stage in the training set, which calculates the class membership degree. Afterwards, FkNN calculates the nearest neighbors for each unseen instance and decides on the predicted class with the highest membership degree. A formal notation for the FkNN algorithm is as follows:

Let TR be a training set and TS a test set, composed of **n** and **t** instances respectively. Each instance $\mathbf{x}_i$ is a vector $(\mathbf{x}_{i1}, \mathbf{x}_{i2}, \mathbf{x}_{i3}, \ldots, \mathbf{x}_{ij})$, where $\mathbf{x}_{ij}$ is the value of the i-th instance and j-th feature. For each instance of TR its class ω is known. However, for TS instances the class is unknown.

FkNN has two stages: class membership degree computation and classification. The first stage calculates the kNN for each instance of TR, keeping a scheme leave-one-out selecting the k instances with a shorter distance. Finally, it calculates the class membership degree according to the Equation 1. The result of the first stage is the TR modifying the class label ω, for a membership vector to each class $(\omega_1, \omega_2, \ldots, \omega_l)$ where l is the number of classes. This new set will be called Fuzzy Training Set, FTR.

$$u_j(x) = \begin{cases} 0.51 + (n_j/k_{memb}) \cdot 0.49 & if \quad j = i \\ (n_j/k_{memb}) \cdot 0.49 & if \quad j \neq i \end{cases} \tag{1}$$

For each instance of the TS, the classification stage calculates its kNN in FTR. Thus, it gets the membership vector of each neighbor and aggregates this vector by applying the Equation 2. Finally, the class with a higher membership will be predicted.

$$u_i(x) = \frac{\sum_{j=1}^{K} u_{ij}(1/|x - x_j|^{2/(m-1)})}{\sum_{j=1}^{K}(1/|x - x_j|^{2/(m-1)})} \tag{2}$$

The first stage of FkNN, which is an extra stage compared to kNN, causes increased computational complexity and generates two issues to deal with big data problems:

- Runtime: The complexity of computing kNN for an instance is $\mathcal{O}(n \cdot c)$, where n is the number of instances of TR and c is the number of features. For more than one neighbor, it increases to $\mathcal{O}(n \cdot log(N))$. In addition, FkNN has an extra stage of computation for calculating the class membership degree.
- Memory consumption: To speed up the calculation, the TR and TS sets stored in the main memory are required. However, when both sets are large, the available main memory is easily exceeded.

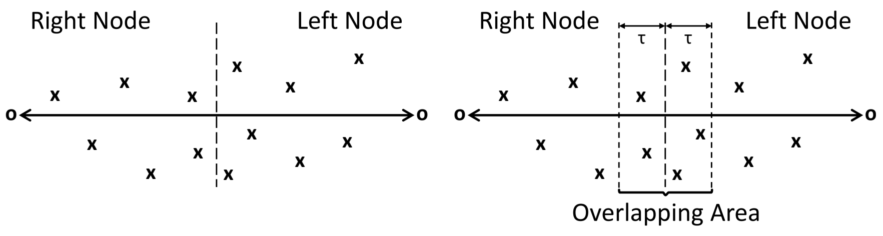
To alleviate these difficulties, we worked on the design of two approximate models based on Hybrid Spill-Tree developed under the big data technologies of MapReduce and Apache Spark.

2.2 Hybrid Spill-Tree: Approximate kNN search

In the search for the nearest neighbor two approaches can be followed: Exact and Approximate. The exact approach aims to ensure that the instance identified as closest is actually the closest. To do this, it needs to calculate the distance to all the samples in TR and select the one with the lowest distance. In the big data environment, reducing runtime and increasing scalability is a very important factor, so the approximate approach is more relevant. The approximate approach can be tackled from different perspectives. Due to its high number of features, the dimensionality reduction [26] is a way to speed up the calculation of distance. The Locality-sensitive hashing algorithm [27] is a well-recognized algorithm for reducing dimensionality through hash functions, generating collisions between similar instances. This requires a previous stage of computation for the calculation of hash functions, reducing the scalability of the algorithm. When dealing with not so many features, but with a large number of instances, tree-based proposals get the best performance. In [20], the authors study tree-based approaches, and propose the Hybrid Spill-Tree algorithm (HS) [21] as the most promising algorithm to accelerate the search for the kNN.

The HS algorithm is formed of Metric-Tree (MT) with its precise search and Spill-Tree (SP) with its fast search. The MT data structure organizes the dataset in a spatial hierarchy, performing a search that ensures the exact nearest instance is found. MT is a binary tree whose root includes the entirety of the samples, and where each child represents a subset of elements. Figure 1a illustrates how to divide the elements between the two children, selecting each child as the furthest possible instance (represented by ◯). The mean distance between the children will be the separation of these nodes. The tree will have a depth of $\mathcal{O}(log(N))$. In order to search for the nearest instance, it keeps the candidate with the shortest distance C and its own distance d. If the distance to a branch is more than d, prune it and continue the search. Once there is no branch in the tree with a distance less than d, the search is finished and C and d are returned. Note that a backtracking operation is made in the structure to ensure that C is the nearest, returning exactly the nearest instance.

The SP data structure is a variation of MT, performing an approximate search to speed up its execution. The main difference compared to MT consists in sharing instances between child nodes. Figure 1b shows how data is divided with the same procedure as MT, allowing a set of duplicate instances in the child nodes. The overlapping area is dependent on the τ parameter. When τ is 0, it would be a MT with no instances shared. If τ is too high, is too high, the depth of the tree rises to infinity because the overlap is high. SP does not backtrack to ensure that the nearest instance has been found, reducing execution times. Moreover, due to the overlapping area, it obtains representative instances of the problem. A common characteristic of MT and SP is that they perform a depth-first search, computationally dependent on the number of features. Thus, when the number of features increases, the runtime is higher.



(a) Metric-Tree (b) Spill-Tree

Figure 1: Partition methodology

HS is proposed with the objective of achieving a balance between accuracy and runtime. Thus, it merges the MT and SP models. To build a HS, it starts by building a SP, and if the number of instances in the overlapping area is less than the Balance Threshold (BT), it will continue to be a SP. If repeated instances exceed BT, it is reconstructed as MT. Figure 2 shows an example of HS, differentiating the MT nodes from the SP nodes. It is important to highlight the starting point for the development of this contribution, which is available in the library developed by the spark-packages community[1].

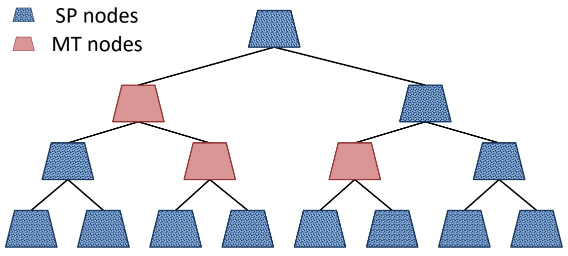


Figure 2: Example of Hybrid-spill tree

2.3 Apache Spark and MapReduce paradigm

The programming paradigm MapReduce [24] will be used in the development of the algorithm proposed in this paper. MapReduce aims to process large datasets through the distribution of data storage and execution through a cluster of computers.

The MapReduce implementation selected is Apache Spark [25] [28]. Spark parallelizes the calculation transparently through a distributed data structure called *Resilient Distributed Datasets* (RDD). RDDs allow data structures stored in main memory to persist and be reused. Additionally, Spark was developed to cooperate with the distributed file system of Hadoop [29] [30] (Hadoop Distributed File System - HDFS).

[1]Hybrid Spill-Tree. https://spark-packages.org/package/saurfang/spark-knn

With this configuration, you gain the benefits provided by Spark: fault tolerance, data splitting and job communication.

MLlib [31] is the official library of machine learning in spark. It incorporates a large number of stadistic techniques and algorithms in areas such as regression, classification, or clustering.

3 Fast and scalable FkNN classifiers for Big Data

This section presents two approximate and distributed proposals for the FkNN algorithm based on the HS method to address big data problems implemented in Spark. Two different approaches are proposed in the class membership degree stage: local and global. The local approach applies a divide-and-conquer approach, where each partition does not know the instances of the other partitions. The global approach has knowledge of all the instances of the TR and develops the use of the HS algorithm. Section 3.1 describes the local approach, performing the computation on each partition independently, without knowing information about the other partitions in the dataset. Section 3.2 presents the global approach based on HS, considering the totality of the data for the calculation of the class membership degree. Section 3.3 defines the classification stage, which is the same for both models and is based on the HS algorithm.

3.1 LHS-FkNN: Local Hybrid Spill Tree FkNN

The proposed local stage together with the classification stage is called Local Hybrid Spill Tree FkNN (LHS-FkNN). Figure 3 shows the class membership stage workflow. To alleviate the bottleneck, data is partitioned and distributed among the computation nodes. Subsequently, the membership to each partition is calculated independently. Finally, the results of each partition are joined, obtaining as output the FTR.

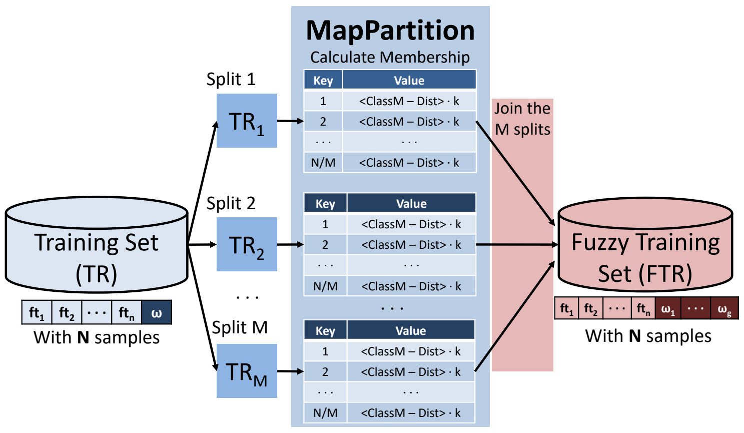


Figure 3: Class membership stage: LHS-FkNN

Algorithm 1 shows the steps and operations in Spark for calculating the class membership degree. It begins by reading the TR from HDFS and divides it into $\#Maps$ parts. Subsequently, a Spark mapPartition operation is used to calculate the class membership degree for each Training set Split (TRS_i) partition in a distributed manner. The membership calculation is represented in lines 6-12. For each y instance of each TRS_i partition, kNN is calculated and finally, the class membership degree is obtained by applying the Equation 1. Once the membership for each partition is obtained, the results are joined and form the FTR (Line 3), which will be the input of the classification stage.

Algorithm 1 Class membership degree stage - Local

```
Require: $TR$, $k$, $\#Maps$
 1: $TRS \leftarrow$ repartition($TR$, #Maps)
 2: $FTRS \leftarrow$ mapPartition(computeMembership($TRP$, $k$))
 3: $FTR \leftarrow$ join($TRPD$)
 4: return $FTR$
 5:
 6: BEGIN computeMembership
 7: for $y$: $TRP_i$ do
 8:     $Neigh_y \leftarrow$ computekNNLocal ($models$, $k$, $y$)
 9:     $membership_y \leftarrow$ computeMembership ($Neigh_y$)
10:     $FTRS \leftarrow$ join($y$, $membership_y$)
11: end for
12: return $FTRS$
13: END computeMembership
```

3.2 GAHS-FkNN: Global Approximate Hybrid Spill Tree FkNN

The global stage together with the classification stage is called the Global Approximate Hybrid Spill Tree FkNN (GAHS-FkNN). Figure 4 specifies the workflow of the membership stage, which follows an approximate scheme based on HS. This approach aims to alleviate the bottleneck computation, with consideration of the data globally to obtain quality in the membership degree. Thus, this approach prioritizes quality over scalability. As in the local approach, the output from this stage is the FTR.

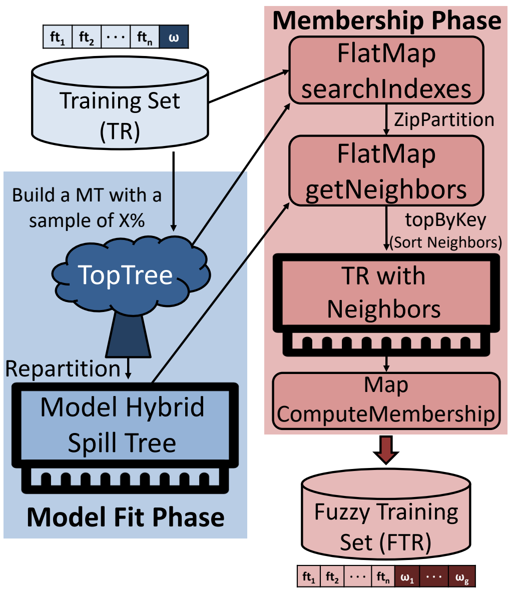


Figure 4: Class membership phase: GAHS-FkNN

Algorithm 2 shows Spark's instructions for the membership degree stage with the global approach. Lines 1-5 correspond to the model creation stage based on HS, and the remaining lines correspond to the kNN and membership computation.

The model fit phase begins by reading the TR from HDFS. First, it takes a random subsample to construct a MT as described in Section 2.2 (the authors recommend 0.2%). This MT receives the name of top tree

Algorithm 2 Class membership degree stage - Global

```
Require: $TR, TS, k$
1: $samples \leftarrow$ sample($TR$,0, 2%)
2: $TopTree \leftarrow$ buildMT($samples$)
3: $\tau \leftarrow$ estimate$\tau$($TopTree$)
4: $tree \leftarrow$ repartition($TR$, $TopTree$, $\tau$, $UE = 70\%$)
5: $model \leftarrow$ (broadcast($TopTree$),tree)
6: for $y$: $TR$ do
7:     $Neigh_y \leftarrow$ computekNN ($model$, $k$, $y$)
8:     $membership_y \leftarrow$ computeMembership ($Neigh_y$)
9:     $result_y \leftarrow$ join($y$, $membership_y$)
10: end for
11: return $prediction_y$
```

(TT) and is used to estimate the value of the τ parameter and partition the entire TR. The estimate of τ is the average distance between all the instances. To speed up this calculation, it is done with the TT instances.

The next step is to split the TR. To do this, the instances are distributed in the space taking as reference the TT. The value of τ defines the overlapping area. It starts building a SP, and checks if the number of instances in the overlapping area is less than 70%. Otherwise, a MT is reconstructed. When performing the search, the SP branches perform a faster search by not backtracking in the tree. However, those built as MT perform backtracking to ensure the nearest is found. The construction stage of the model ends up distributing the TT and the tree associated with the TR.

The membership phase is shown in lines 6-10. For each TR instance, kNN is calculated following the model generated. Algorithm 3 describes how to perform the kNN with native Spark operations. Using a flatMap operation, the indexes of the nearest instances of TR are computed and obtained. Thus, the distance to the right and the left nodes is calculated, and it continues the search of the nearest instance through the node with a shorter distance. When it reaches a leaf node, it returns the index of the selected instance.

Algorithm 3 Compute kNN

```
Require: $model$, $k$, $x$
1: $Indexes \leftarrow$ x.flatMap (searchIndexes($model.tree$) )
2: $Neighs \leftarrow$ query($model.tree$, $Indexes$, $k$)
3: return $Neighs$
4:
5: BEGIN searchIndexes
6: $distLeft \leftarrow$ nodeLeft.dist($x$)
7: $distRight \leftarrow$ nodeRight.dist($x$)
8: if $node! = LEAF$ then
9:     if $distLeft < distRight$ then
10:         searchIndexes($nodeLeft$, $ID$)
11:     else
12:         searchIndexes($nodeRight$, $ID + childLeft$)
13:     end if
14: else
15:     return $Indexes$
16: end if
17: END searchIndexes
```

With the neighbors, the class membership degree vector is calculated by Equation 1 (Line 8). The result of this phase is the FTR, and becomes the input of the classification stage.

3.3 Classification stage

The proposed classification stage receives as input the FTR calculated in the previous stage, the TS and the value of k. The TS is usually significantly smaller than FTR, for this reason, the classification stage has a lower computational cost than the membership stage. In order to obtain better results in the classification, a global approach is followed, which considers all the instances for the decision making. However, it is approximate in nature in order to speed up the runtime and obtain a higher scalability. Figure 5 shows the HS-based classification stage workflow. It has two distinct phases: model fit and classification. In the first the tree is built and the instances are divided between the computation nodes. In the second the kNN of FTR is searched for and the predicted class is returned as output according to the membership degree vector.

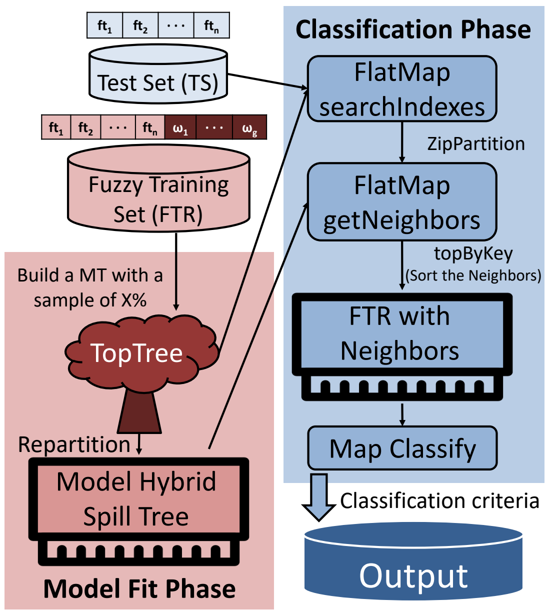


Figure 5: Flowchart of the classification phase

Algorithm 4 shows the native instructions from Spark for the classification stage. Lines 1-5 correspond to the model fit phase, and the remaining lines correspond to the classification stage. Due to the similarity in the data flow with the calculation stage of class membership degree based on HS, only the differences are detailed.

Algorithm 4 Classification Stage

```
Require: $FTR, TS, k$
1: $samples \leftarrow$ sample($FTR$,0.2%)
2: $TopTree \leftarrow$ buildMT($samples$)
3: $\tau \leftarrow$ estimate$\tau$($TopTree$)
4: $tree \leftarrow$ repartition($FTR, TopTree, \tau, UE = 70\%$)
5: $model \leftarrow$ (broadcast($TopTree$),tree)
6: for $x$: $TS$ do
7:     $NeighMemb_x \leftarrow$ computekNN ($model, k, x$)
8:     $prediction_x \leftarrow$ computeMembership ($NeighMemb_x$)
9:     $result_x \leftarrow$ join($x, prediction_x$)
10: end for
11: return $prediction_y$
```

The first difference is in the input datasets. In this case, FTR and TS will be used. The model fit phase is not affected, since the input variables are not modified, and the distances of the instances are maintained. Thus, the model is built with the same methodology, modifying only the class label of the TR, by the membership degree vector of the FTR.

The FkNN calculation is the same as that applied in the HS-based membership calculation stage, which was described in Algorithm 3. In contrast to the membership calculation, the kNN calculation returns the membership degree vector instead of the class label (Line 7). In Line 8, it calculates the predicted class applying Equation 2, obtaining the predicted class for each TR instance as the final result.

4 Experimental set-up

This section presents the issues involved in the experimental framework. It presents the performance measures used (Section 4.1), the details of the datasets (Section 4.2) and the algorithms used with their respective parameters (Section 4.3). Finally, the hardware and software used for the experimentation phase are specified (Section 4.4).

4.1 Performance measures

In this work, the efficiency and scalability of the models will be evaluated using the following metrics:

- *Accuracy:* The most widely used metric in the literature [32] [33] will be applied to evaluate the quality of the classifiers. This metric counts the number of correct classifications in relation to the total number of instances. Experimentation will be performed on classification problems with an appropriate class balance, where accuracy is a representative measure.
- *Runtime:* Time consumed in computation, also considering the readings and network communications by Spark. In addition, the runtimes will be taken for each one of the two stages that compose the fuzzy algorithms studied in order to analyze the time each of them require.

To validate the results of the experiments, we have used the pairwise non-parametric statistical tests based on Dirichlet Process called Bayesian Sign test [34]. Bayesian Sign test calculates a distribution with the differences of the results obtained from the confrontation of the two algorithms. Thus, a triangle is constructed that will determine depending on the position of the majority of the distribution, if there is a draw (rope position), victory of the first algorithm (right position) or victory of the second algorithm (left position). The statistical test and the graph shown in the experiments have been generated by the package in R called rNPBST [35].

4.2 Dataset

For the experimental study, we have selected eight datasets in a large number of instances. The ECBDL14 dataset is extracted from the competition [36]. Although it has an imbalance ratio greater than 45, we incorporate this dataset to study the effect of a large number of features. However, in this paper we do not address the problem of imbalance classification, so it has been sub-sampled by obtaining an imbalance ratio of two. The Epsilon dataset has been taken from the LIBSVM repository [37] and it was artificially created for the Pascal Large Scale Learning competition [38]. This dataset was selected to analyze how a high number of features affects the proposed algorithms. The other six datasets have been extracted from the UCI repository [39]. The Table 1 presents the number of instances, characteristics and classes ($\#\omega$).

The cross-validation scheme will be followed in 5 partitions, composed of 80% training instances and the remaining 20% test instances.

In the MapReduce schema, the number of instances processed in each worker depends on the number of instances of the dataset and the number of map tasks used in the execution. The Table 2 shows the number of instances for TR and TS according to the number of map tasks.

Table 1: Description of the datasets

Dataset	#Examples	#Features	$\#\omega$
Covtype	581,012	54	2
ECBDL14-S	2,063,187	631	2
Epsilon	500,000	2,000	2
Higgs	11,000,000	28	2
Poker	1,025,010	10	10
Susy	5,000,000	18	2
Watch-acc	3,540,962	20	7
Watch-gyr	3,205,431	20	7

Table 2: Number of instances per map

Dataset	TR - **#Instances**			TS - **#Instances**		
	64	128	256	64	128	256
Covtype	7,262	3,631	1,816	1,816	908	454
ECBDL14-S	25,790	12,895	6,447	6,448	3,224	1,612
Epsilon	6,250	3,125	1,562	1,562	781	390
Higgs	137,500	68,750	34,375	34,376	17,188	8,594
Poker	12,812	6,406	3,203	3,204	1,602	801
Susy	62,500	31,250	15,625	15,626	7,813	3,906
Watch-acc	44,262	22,131	11,066	11,066	5,533	2,766
Watch-gyr	40,068	20,034	10,017	10,016	5,008	2,504

4.3 Algorithms and parameters

The experimentation carried out has been compared with other proposals of FkNN and its crisp analogs. The algorithms used and their acronyms are presented below:

- *Global Exact FkNN (GE-FkNN)* [22]: exact model of the FkNN algorithm to tackle big data problems, obtaining the same results as the original FkNN. Its two stages are global and exact.
- *Local FkNN (L-FkNN)*: developed proposal of the FkNN algorithm for this contribution. The first stage, which is responsible for calculating the class membership degree, is described in Section 3.1. The second stage is global and exact, identical to the used by that GE-FkNN algorithm.
- *k Nearest Neighbor - Iterative Spark (kNN-IS)* [18]: crisp kNN's exact proposal to tackle big data problems, getting the same results as the original kNN.
- *Hybrid Spill-Tree kNN (HS-kNN)* [21]: Approximate proposal of crisp kNN for big data. Although approximate, consider all instances in the search.

The best-known FkNN parameter is the number of neighbors (k) considered in the classification. k may be different in the membership and classification stages. However, for the sake of simplicity, it is kept the same in both stages. For all the algorithms used, values 3, 5 and 7 have been used. In addition, the experiments on the GAHS-FkNN model and the LHS-kNN proposal are extended using values of k from 3 to 51. The distributed component adds an extra parameter, the number of partitions or map operations. In the experiments, they take values of 64, 128 and 256.

Models based on the HS algorithm need two parameters to build the model and speed up the search for the nearest instances. The first is the percentage of instances that will be taken into account to form the TT, which is then used to divide and distribute the data. The second is the BT, the admission percentage of repetition of instances between nodes of the tree to decide if a ST or a MT is constructed. The study has taken the optimal values recommended by the authors of HS: TT equal to 0.2% and BT equal to 70%.

4.4 Hardware and software used

All experiments have been performed on a cluster composed of 15 nodes: a master node and 14 computation nodes. All the nodes have the same configuration:

- Processor: 2x Intel Xeon CPU E5-2620 (2 GHz).
- Cores: 6 cores (12 threads).
- RAM: 64 GB.
- Cache: 15 MB
- Network: Infiniband 40 Gb/s.

All nodes have the same software set-up:

- Operative System: CentOS 6.5.
- Apache Spark: Version 2.2.1.
- Scala: Version 2.11.6.
- Hadoop Distributed File System: Version 2.6.0-cdh5.8.0.

With this software and hardware configuration, there is a maximum of 256 concurrent map tasks available. Thus, there are approximately 2GB of main memory for each of these map tasks.

5 Analysis of results

In this section, we study the results compiled from different experimental studies. Specifically, we analyze the following points:

- First, we establish a comparison between the proposals and the state-of-the-art FkNN algorithms in terms of accuracy and runtime. (Section 5.1)
- Second, we performed a scalability study on the successful proposals. To do this we will focus on the runtimes. (Section 5.2)

- Third, we extend the two most promising models to higher values of k by focusing on accuracy. (Section 5.3)
- Fourth, we compared the results obtained for each algorithm and dataset against the crisp kNN proposals similar to the fuzzy models studied. (Section 5.4)

5.1 Accuracy study

The accuracy study starts by showing the results from Table 3, which compares the accuracy between the algorithms in relation to the number of neighbors (k) and the number of map operations equal to 128 for all datasets. The best result for each dataset and the best average result are highlighted in bold. Those values that could not be executed due to scalability problems are represented with the symbol “-", for the calculation of the mean, they are considered as a zero in accuracy. Figure 6 presents the probability distribution of the differences between the GAHS-FkNN and LHS-FkNN algorithms obtained with the Bayesian Sign Test.

Table 3: Accuracy comparison between algorithms

Algorithm	k	Covtype	Epsilon	ECBDL14-S	Higgs	Poker	Susy	Watch-acc	Watch-gyr	Average
GE-FkNN	3	0.9299	0.5545	-	-	0.5264	0.7338	0.9330	0.9597	0.5797
	5	0.9151	0.5452	-	-	0.5329	0.7354	0.9069	0.9398	0.5719
	7	0.9014	0.5394	-	-	0.5371	0.7320	0.8880	0.9254	0.5654
L-FkNN	3	0.9360	0.5727	0.7659	0.5954	0.5240	0.7227	0.9216	0.9506	0.7486
	5	0.9268	0.5696	0.7526	0.5984	0.5274	0.7260	0.8935	0.9284	0.7403
	7	0.9167	0.5729	0.7455	0.6000	0.5286	0.7253	0.8768	0.9148	0.7351
GAHS-FkNN	3	**0.9375**	0.5808	**0.8054**	0.5969	0.5237	0.7298	**0.9601**	**0.9808**	0.7644
	5	0.9347	0.5897	0.8046	0.6084	0.5368	0.7461	0.9576	0.9790	0.7696
	7	0.9313	0.5946	0.8013	0.6163	0.5451	0.7514	0.9558	0.9776	0.7717
LHS-FkNN	3	0.9372	0.5838	0.8034	0.6047	0.5333	0.7298	0.9566	0.9790	0.7660
	5	0.9362	0.5957	0.8025	0.6145	0.5446	0.7461	0.9544	0.9779	0.7715
	7	0.9338	**0.6066**	0.7968	**0.6214**	**0.5502**	**0.7514**	0.9530	0.9765	**0.7737**

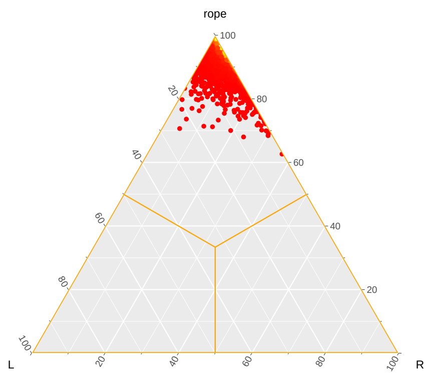
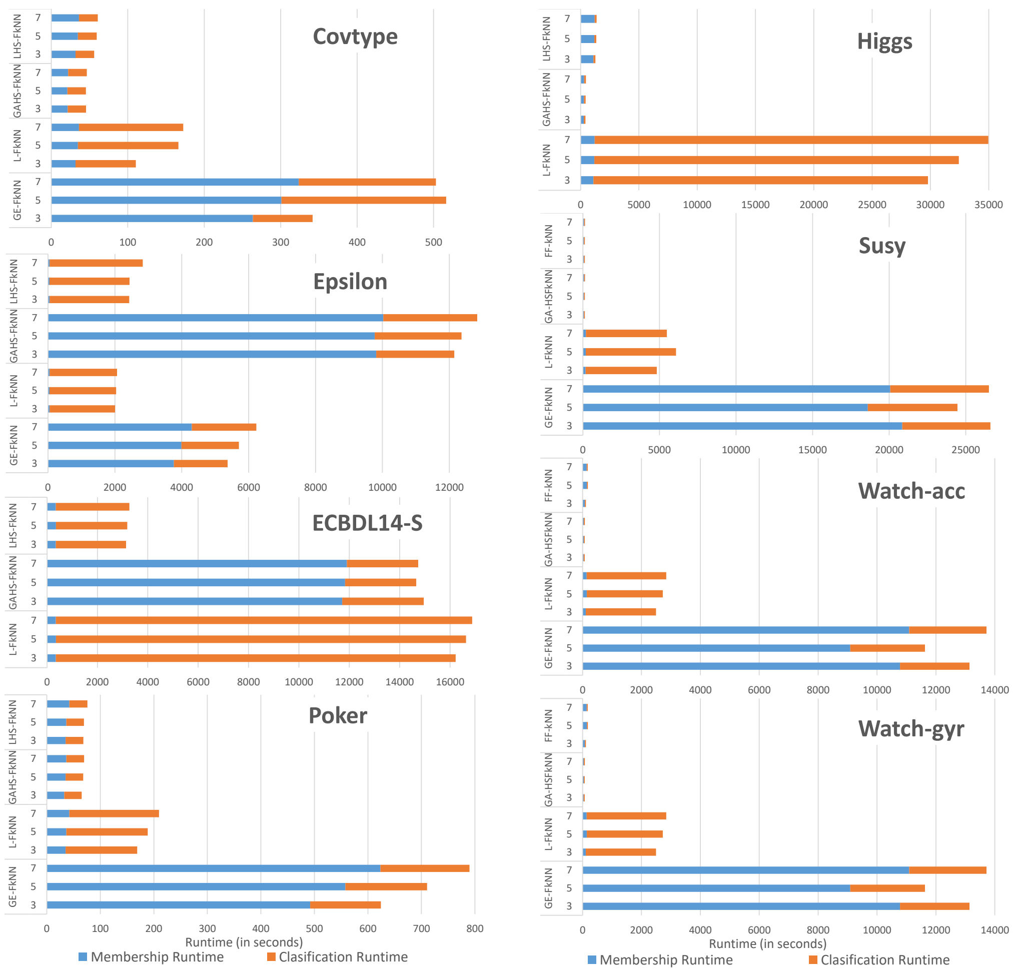


Figure 6: Heatmap of the Bayesian Sign Test: GAHS-FkNN vs LHS-FkNN

Figure 7 shows the membership stage and the classification stage runtimes in seconds, for each dataset, algorithms and values of k equal to 3, 5 and 7.

Analyzing the table and figures presented, we can observe:

- The GE-FkNN algorithm finds its scalability limit in its first stage, not being able to run for the ECBDL14-S and Higgs datasets.



(a) Datasets: Covtype, Epsion, ECBDL14-S and Poker (b) Datasets: Higgs , Susy, Watch-acc and Watch-gyr

Figure 7: Runtime comparison between algorithms: GAHS-FkNN, LHS-FkNN, GE-FkNN and L-FkNN

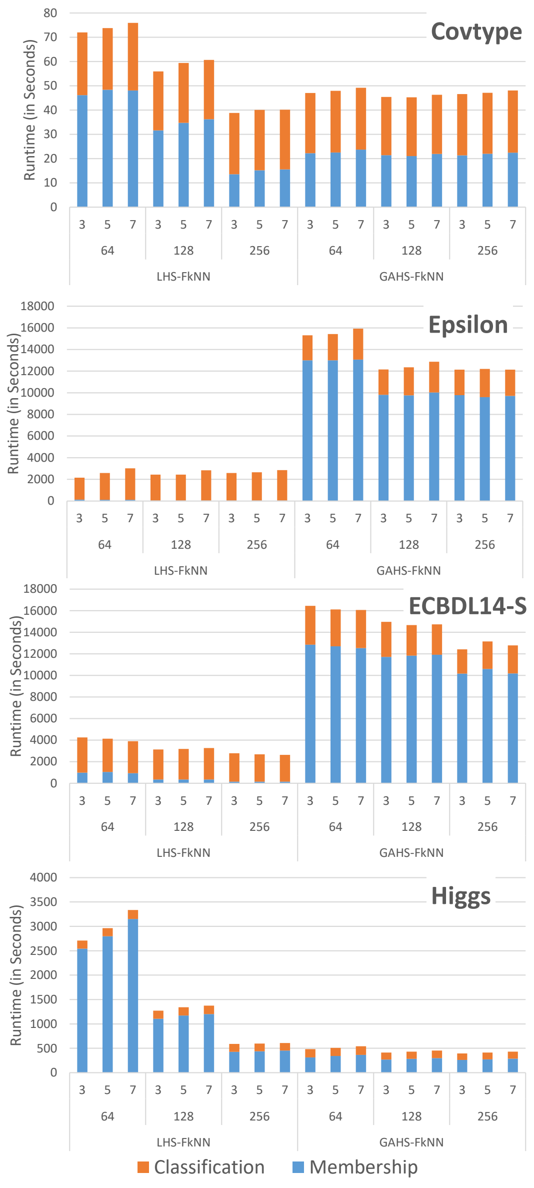
- Models based on HS (GAHS-FkNN and LHS-FkNN) achieve better results than models with the exact classification stage (GE-FkNN and L-FkNN). This may be due to the approximate component of the subjascent model HS, which obtains a noise tolerance of the datasets in the GAHS-FkNN and LHS-FkNN algorithms, as the exact component of kNN can lead to a high overfit to the training set.

- Regarding the influence of the k, the GE-FkNN and L-FkNN algorithms do not obtain an appreciable improvement, with a stagnation of the results in accuracy. However, the GAHS-FkNN and LHS-FkNN algorithms display an increase in accuracy. For this reason, the two most promising algorithms will be studied for higher k values. In relation to the runtime, it should be noted that the value of k affects the GE-FkNN and L-FkNN models to a certain extent, whereas it does not drastically affect the proposed GAHS-FkNN and LHS-FkNN algorithms.

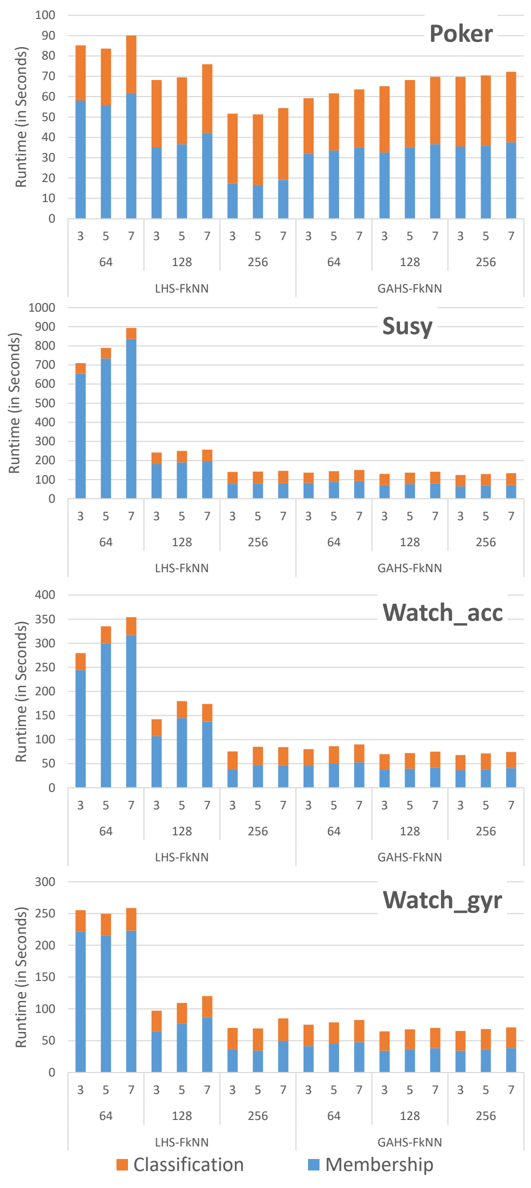
- Figure 6 shows how the GAHS-FkNN and LHS-FkNN algorithms are statistically equal in accuracy, although on average, the LHS-FkNN algorithm is slightly better than GAHS-FkNN. For this reason, it is important to analyze the scalability of the models in relation to the number of map operations used.

5.2 Scalability study

The scalability study starts by presenting the results from Figure 8, which compares the runtime of the membership stage and the runtime of the classification stage in seconds, for each datasets, with values of k equal to 3, 5 and 7 and number of maps equal to 64, 128 and 256.



(a) Datasets: Covtype, Epsilon, ECBDL14-S and Higgs



(b) Datasets: Poker, Susy, Watch-acc and Watch-gyr

Figure 8: Scalability comparison between GAHS-FkNN and LHS-FkNN

According to the figure shown:

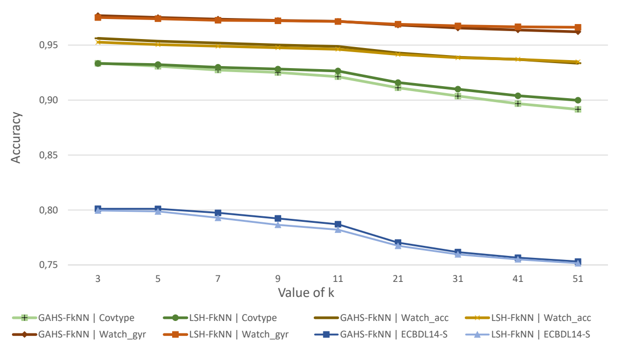
- The GAHS-FkNN algorithm is affected when dealing with a large number of features. This is due to the HS structure generating trees with a high depth in their branches as it has a high number of features, resulting in higher runtimes. This can be observed in the Epsilon and ECBDL14-S datasets, where the runtime obtained by LHS-FkNN are much faster than the runtimes for the GAHS-FkNN algorithm.

- LHS-FkNN scale depending on the number of maps thanks to its first local stage, obtaining a performance associated with the hardware used.
- GAHS-FkNN gets good runtimes without significantly affecting the hardware used, showing interesting behavior but limiting the scalability of the model.
- If we focus on the runtime of the classification stage, it is shared by GAHS-FkNN and LHS-FkNN as both models follow the same scheme on this stage.

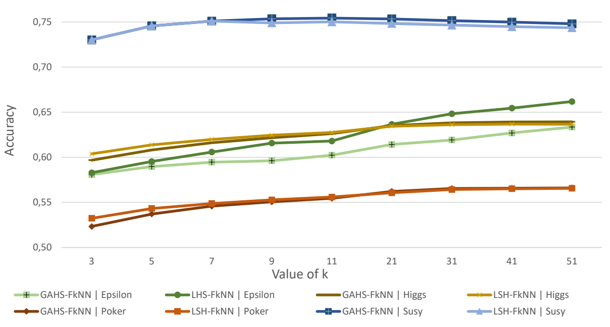
5.3 Study for higher values of k

According to the results shown, the GE-FkNN and L-FkNN algorithms show a stagnation in accuracy with the values of k set to 3, 5 and 7. In addition, the runtime is not drastically affected by the k. However, the GAHS-FkNN and LHS-FkNN algorithms keep improving the results. For this reason, this section extends the values of k up to 51, studying the accuracy obtained by both algorithms and the 8 datasets, setting the number of maps to 128.

Figure 9 presents the accuracy obtained for each dataset with the GAHS-FkNN and LHS-FkNN algorithms, with values of k between 3 and 51. In order to facilitate the visualization of the results, two figures are shown due to the differences in accuracy between the datasets.



(a) Datasets: Covtype, ECBDL14-S, Watch-acc and Watch-gyr



(b) Datasets: Epsilon, Higgs, Poker and Susy

Figure 9: Accuracy comparison with higher values of k: GAHS-FkNN vs LHS-kNN

According to the figures presented we can observe:

- The results obtained (accuracy) according to the value of k follow a similar behavior pattern for both algorithms. Focusing on the datasets, high values of k improve the accuracy in the Higgs, Poker, Epsilon and Susy datasets. However, low values of k improve accuracy for Covtype, Watch-acc, Watch-gyr and ECBDL14-S datasets. This is a natural behavior for the FkNN algorithm, which occurs with classical datasets from the literature. Therefore, the proposed algorithms show the same behavior in large datasets.
- Comparing GAHS-FkNN and LHS-FkNN in terms of accuracy, we see that the differences are very low, and when this difference is accentuated somewhat more in the Covtype and Epsilon datasets, LHS-FkNN is the clear winner.

5.4 Comparison with crisp kNN algorithms

As the influence of the number of maps has already been analyzed and not considered significant, this experiment has been set to 128 maps in order to focus on the comparative study of crisp vs fuzzy.

Figure 10 shows the total runtime for the algorithms kNN-IS, HS-kNN, GAHS-FkNN and LHS-kNN. To facilitate the study of the runtime, it is presented only with the value of $k = 5$. The results are shown for two figures due to the differences in the scales of the total runtime of each dataset.

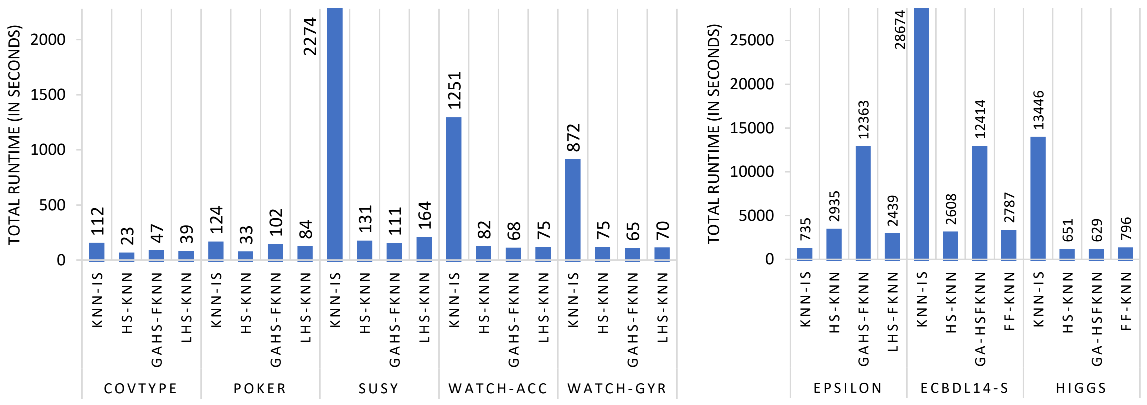


Figure 10: Total Runtime comparison between Crisp and Fuzzy models

Table 4 shows a comparison between the result obtained by the two proposed algorithms and the two crisp-alternatives, exploring the values of k 3, 5 and 7.

Table 4: Crisp vs Fuzzy models: accuracy

Algoritm	k	Covtype	Epsilon	ECBDL14-S	Higgs	Poker	Susy	Watch-acc	Watch-gyr	Average
kNN-IS	3	0.9371	0.5864	0.7833	0.5454	0.4758	0.6675	**0.9647**	**0.9845**	0.7431
	5	0.9367	0.6007	0.7797	0.5458	0.4952	0.6784	0.9613	0.9811	0.747
	7	0.9326	**0.6110**	0.7683	0.5559	0.4937	0.6861	0.9582	0.9788	0.7481
HS-kNN	3	0.9308	0.5847	0.8020	0.5885	0.5201	0.7223	0.9542	0.9755	0.7598
	5	0.9232	0.5981	0.8017	0.5936	0.5305	0.7360	0.9478	0.9698	0.7626
	7	0.9161	0.6086	0.7986	0.5981	0.5369	0.7431	0.9423	0.9651	0.7636
GAHS-FkNN	3	**0.9375**	0.5808	**0.8054**	0.5969	0.5237	0.7298	0.9601	0.9808	0.7644
	5	0.9347	0.5897	0.8046	0.6084	0.5368	0.7461	0.9576	0.9790	0.7696
	7	0.9313	0.5946	0.8013	0.6163	0.5451	**0.7514**	0.9558	0.9776	0.7717
LHS-FkNN	3	0.9372	0.5838	0.8034	0.6047	0.5333	0.7331	0.9566	0.9790	0.7664
	5	0.9362	0.5957	0.8025	0.6145	0.5446	0.7446	0.9544	0.9779	0.7713
	7	0.9338	0.6066	0.7968	**0.6214**	**0.5502**	0.7480	0.9530	0.9765	**0.7733**

According to the table and figure presented, it can be seen that the best results are obtained by the proposed algorithms. Although HS-kNN improves with respect to the kNN-IS algorithm, it is always less accurate than the FkNN models, without the runtime being excessively increased due to the optimization carried out in the classification stage of the GAHS-FkNN algorithm. kNN-IS wins in the Epsilon, Watch-acc and Watch-gyr datasets, possibly because it is a dataset with clearly differentiated boundaries and low noise, where the classification problem is simpler than in the other datasets. Despite this, on average the proposed fuzzy models are clearly better.

6 Conclusions and further work

In this paper, two MapReduce approaches have been proposed to speed up the FkNN algorithm in Big Data problems. Because of the design and use of big data technologies, it is possible to execute with very large datasets. In order to study any possible improvements, the proposed model has also been compared with Fuzzy and Crisp versions of the literature. The GAHS-FkNN and LHS-FkNN algorithms achieve statistically equal results in terms of accuracy. On the one hand, the LHS-FkNN algorithm demonstrates very high scalability depending on the hardware facilities available, as well as high accuracy results. On the other hand, the GAHS-FkNN algorithm is less dependent on hardware resources but is more affected by a high number of features.

Thus, the use of LHS-FkNN is recommended when we have powerful hardware according to the problem we want to address, and if the number of features is high. The use of GAHS-FkNN is recommended when the number of features is not too high and we have hardware limitations.

A library has been generated with the algorithms used in this study and is available in the spark-packages platform at `https://spark-packages.org/package/JMailloH/HS_FkNN`.

As future work, we aim to tackle the class imbalanced problem through evolutionary undersampling techniques [40], capable of handling large datasets.

Acknowledgments

This contribution has been supported by the Spanish National Research Project TIN2017-89517-P. J. Maillo hold a FPU scholarship from the Spanish Ministry of Education.

References

[1] J. M. Keller, M. R. Gray, and J. A. Givens. A fuzzy k-nearest neighbor algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(4):580–585, July 1985.

[2] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[3] Joaquin Derrac, Salvador García, and Francisco Herrera. Fuzzy nearest neighbor algorithms: Taxonomy, experimental analysis and prospects. *Information Sciences*, 260:98 – 119, 2014.

[4] Zhennao Cai, Jianhua Gu, Caiyun Wen, Dong Zhao, Chunyu Huang, Hui Huang, Changfei Tong, Jun Li, and Huiling Chen. An intelligent parkinson's disease diagnostic system based on a chaotic bacterial foraging optimization enhanced fuzzy knn approach. *Computational and Mathematical Methods in Medicine*, 2018:24, 2018.

[5] J. Qin, L. Wang, and R. Huang. Research on fault diagnosis method of spacecraft solar array based on f-knn algorithm. In *2017 Prognostics and System Health Management Conference (PHM-Harbin)*, pages 1–4, July 2017.

[6] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Springer, 2015.

[7] Jose M. Cadenas, M. Carmen Garrido, Raquel Martínez, Enrique Muñoz, and Piero P. Bonissone. A fuzzy k-nearest neighbor classifier to deal with imperfect data. *Soft Computing*, 22(10):3313–3330, May 2018.

[8] Soufiane Ezghari, Azeddine Zahi, and Khalid Zenkouar. A new nearest neighbor classification method based on fuzzy set theory and aggregation operators. *Expert Systems with Applications*, 80:58 – 74, 2017.

[9] I. Banerjee, S. S. Mullick, and S. Das. On convergence of the class membership estimator in fuzzy k-nearest neighbor classifier. *IEEE Transactions on Fuzzy Systems*, pages 1–1, 2018.

[10] Hamed Nikdel, Yahya Forghani, and S. Mohammad Hosein Moattar. Increasing the speed of fuzzy k-nearest neighbours algorithm. *Expert Systems*, 35(3):e12254, 2018.

[11] Joaquín Derrac, Francisco Chiclana, Salvador García, and Francisco Herrera. Evolutionary fuzzy k-nearest neighbors algorithm using interval-valued fuzzy sets. *Information Sciences*, 329:144 – 163, 2016.

[12] Peyman Hosseinzadeh Kassani, Andrew Beng Jin Teoh, and Euntai Kim. Evolutionary-modified fuzzy nearest-neighbor rule for pattern classification. *Expert Systems with Applications*, 88:258 – 269, 2017.

[13] Nimagna Biswas, Saurajit Chakraborty, Sankha Subhra Mullick, and Swagatam Das. A parameter independent fuzzy weighted k-nearest neighbor classifier. *Pattern Recognition Letters*, 101:80 – 87, 2018.

[14] Saint John Walker. *Big data: A revolution that will transform how we live, work, and think*. Taylor & Francis, 2014.

[15] Oliver Urs Lenz, Daniel Peralta, and Chris Cornelis. A scalable approach to fuzzy rough nearest neighbour classification with ordered weighted averaging operators. In *International Joint Conference on Rough Sets, June*, June 2019.

[16] R Talavera-Llames, R Pérez-Chacón, A Troncoso, and F Martínez-Álvarez. Big data time series forecasting based on nearest neighbours distributed computing with spark. *Knowledge-Based Systems*, 161:12–25, 2018.

[17] 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.

[18] Jesus Maillo, 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(Supplement C):3 – 15, 2017. Volume, Variety and Velocity in Data Science.

[19] Jeffrey K. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175 – 179, 1991.

[20] Ting Liu, Andrew W Moore, Ke Yang, and Alexander G Gray. An investigation of practical approximate nearest neighbor algorithms. In *Advances in neural information processing systems*, pages 825–832, 2005.

[21] Ting Liu, Charles J Rosenberg, and Henry A Rowley. Performing a parallel nearest-neighbor matching operation using a parallel hybrid spill tree, January 6 2009. US Patent 7,475,071.

[22] J. Maillo, J. Luengo, S. García, F. Herrera, and I. Triguero. Exact fuzzy k-nearest neighbor classification for big datasets. In *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6, July 2017.

[23] J. Maillo, J. Luengo, S. García, F. Herrera, and I. Triguero. A preliminary study on hybrid spill-tree fuzzy k-nearest neighbors for big data classification. In *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–8, July 2018.

[24] J. Dean and S. Ghemawat. Mapreduce: A flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[25] 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*, pages 1–14. USENIX Association, 2012.

[26] Subhajit Dutta and Anil K. Ghosh. On some transformations of high dimension, low sample size data for nearest neighbor classification. *Machine Learning*, 102(1):57–83, Jan 2016.

[27] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 459–468, Oct 2006.

[28] Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia. *Learning spark: lightning-fast big data analysis*. " O'Reilly Media, Inc.", 2015.

[29] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 3rd edition, 2012.

[30] Apache Hadoop Project. Accessed on Dec. 2018. [online].

[31] 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.

[32] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[33] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 3th edition, 2014.

[34] 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.

[35] Jacinto Carrasco, Salvador García, María del Mar Rueda, and Francisco Herrera. rNPBST: an r package covering non-parametric and bayesian statistical tests. In *Hybrid Artificial Intelligent Systems*, pages 281–292. Springer, 2017.

[36] ECBDL14 dataset: Protein structure prediction and contact map for the ECBDL2014 big data competition, 2014.

[37] Epsilon in the LIBSVM website, 2019. Accessed on May. 2019. [online].

[38] Pascal large scale learning challenge. 2008.

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

[40] Salvador García and Francisco Herrera. Evolutionary undersampling for classification with imbalanced datasets: Proposals and taxonomy. *Evolutionary computation*, 17(3):275–306, 2009.

3 Transforming big data into smart data: An insight on the use of the k nearest neighbors algorithm to obtain quality data

- I. Triguero, D. García-Gil, J. Maillo, J. Luengo, S. García, F. 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. 9:3 (2018). e1289.

  – Status: **Published**.

  – Impact Factor (JCR 2018): **2.541**

  – Subject Category: **Computer Science, Artificial Intelligence**

  – Rank: **61/133**

  – Quartile: **Q2**

TRANSFORMING BIG DATA INTO SMART DATA: AN INSIGHT ON THE USE OF THE K-NEAREST NEIGHBORS ALGORITHM TO OBTAIN QUALITY DATA

Isaac Triguero
School of Computer Science, University of Nottingham
Jubilee Campus, Nottingham NG8 1BB, United Kingdom
Isaac.Triguero@nottingham.ac.uk

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

Jesus Maillo
Department of Computer Science
and Artificial Intelligence
University of Granada, Granada, Spain 18071
jesusmh@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

The k-nearest neighbours algorithm is characterised as a simple yet effective data mining technique. The main drawback of this technique appears when massive amounts of data - likely to contain noise and imperfections - are involved, turning this algorithm into an imprecise and especially inefficient technique. These disadvantages have been subject of research for many years, and among others approaches, data preprocessing techniques such as instance reduction or missing values imputation have targeted these weaknesses. As a result, these issues have turned out as strengths and the k-nearest neighbours rule has become a core algorithm to identify and correct imperfect data, removing noisy and redundant samples, or imputing missing values, transforming Big Data into Smart Data - which is data of sufficient quality to expect a good outcome from any data mining algorithm. The role of this smart data gleaning algorithm in a supervised learning context will be investigated. This will include a brief overview of Smart Data, current and future trends for the k-nearest neighbour algorithm in the Big Data context, and the existing data preprocessing techniques based on this algorithm. We present the emerging big data-ready versions of these algorithms and develop some new methods to cope with Big Data. We carry out a thorough experimental analysis in a series of big datasets that provide guidelines as to how to use the k-nearest neighbour algorithm to obtain Smart/Quality Data for a high quality data mining process. Moreover, multiple Spark Packages have been developed including all the Smart Data algorithms analysed.

Keywords Big data · Data preprocessing · Instance reduction · K-nearest neighbors · Imperfect data · Smart data · Instance reduction · Spark

GRAPHICAL TABLE OF CONTENTS

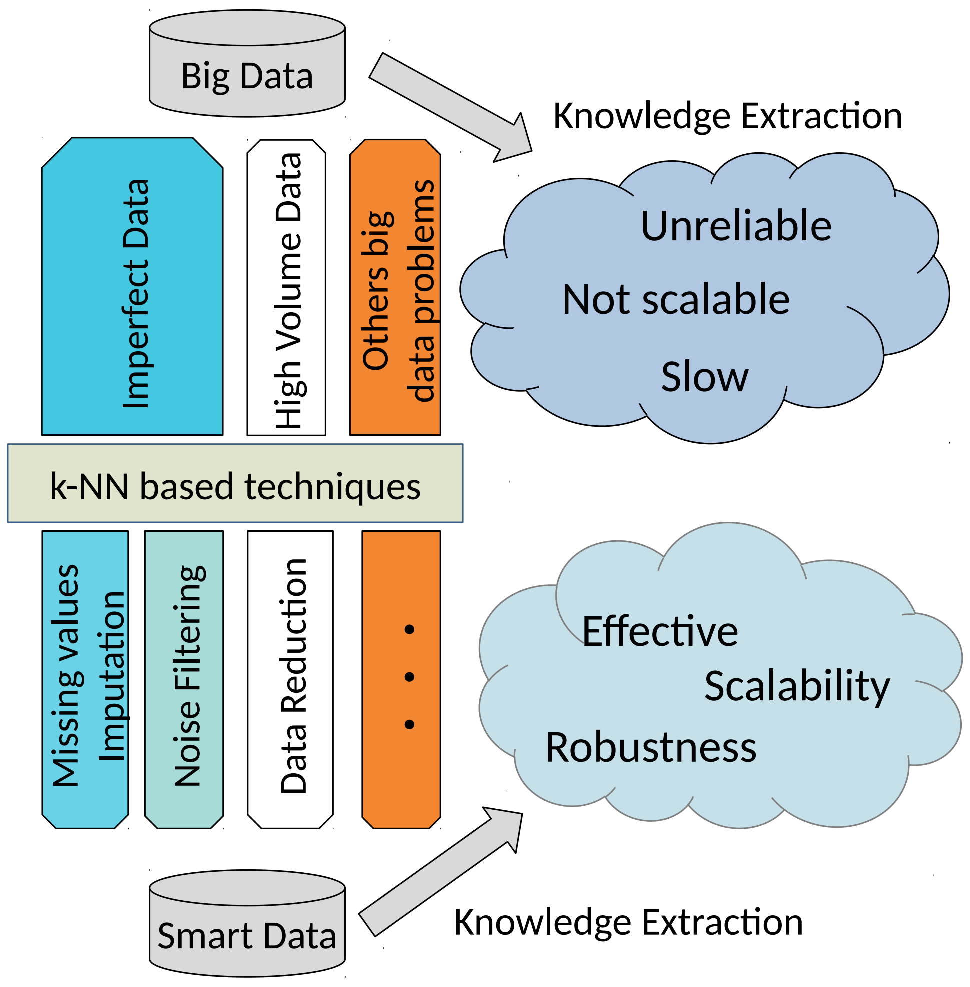


Figure 1: The k-Nearest Neighbours algorithm plays a key role to cope with Big Data by transforming it into Smart data that is free of redundant information, noise and/or missing values. Gleaning quality data is essential for a correct data mining process that will uncover valuable insights.

1 INTRODUCTION

Big data analytics is nowadays sitting at the forefront of many disciplines that are not directly related to computer science, statistics or maths. The advent of the Internet of Things, the Web 2.0, and the great advances in technology are transforming many areas such as medicine, business, transportation or energy by collecting massive amounts of information [1, 2, 3, 4]. However, the real benefit of Big Data is not on the data itself, but in the ability to uncover (unexpected) patterns and glean knowledge from it with appropriate Data Science techniques. The impact of exploiting this data may reflect on competitive advantages for companies or unprecedented discoveries in multiple science fields [5]. Nevertheless, both companies and researchers are facing major challenges to cope with the Volume, Velocity, Veracity, and Variety (among others V's) that characterise this flood of data. These V's define the main issues of the Big Data problem [6].

The premise of Big Data is that having a world rich in data may enable machine learning and data mining techniques [7] to obtain more accurate models than ever before, but classical methods fail to handle the new data space requirements. With the leverage of distributed technologies such as the MapReduce programming paradigm and the Apache Spark platform [8, 9], some classical data mining algorithms are being adapted to this new data-intensive scenario [10, 11]. However, Big Data mining techniques are not only confronted with scalability or speed issues (volume/velocity) and they will also have to handle inaccurate data (noisy or incomplete) and massive amounts of redundancy. In addition, a key question for many companies and research institutions remains unanswered: Do we really need to keep stored big amounts of raw data that may be inaccurate just for the sake of it? Storing data does not come for free and a way of finding sustainable storage is becoming imperative.

The term of Smart Data [12] refers to the challenge of transforming raw data into quality data that can be appropriately exploited to obtain valuable insights [13]. Gartner, Inc in 2015[1] defined Smart Data discovery as "a next-generation data discovery capability that provides business users or citizen data scientists with insights from advanced analytics". Therefore, Smart Data discovery is tasked to extract useful information from data, in the form of a subset (big or not), which poses enough quality for a successful data mining process. The impact of Smart Data discovery in industry and academia is two-fold: higher quality data mining and reduction of data storage costs.

Data preprocessing [14] clearly resembles the concept of Smart Data as one of the most important stages of a data mining process. Its goal is to clean and correct input data, so that, a machine learning process may be later applied faster and with a greater accuracy. With this definition, data preprocessing techniques should enable data mining algorithms to cope with Big Data problems more easily. Unfortunately, these methods are also heavily affected by the increase in size and complexity of datasets and they may be unable to provide a preprocessed/smart dataset in a timely manner, and therefore, need to be redesigned with Big Data technologies.

A simple yet powerful data mining technique is the k-Nearest Neighbour algorithm (k-NN) [15]. This is based on the concept of similarity between samples, which in classification problems, for example, this implies that patterns that are similar have to be assigned to the same class. As a lazy learning algorithm [16], it does not carry out a training phase per se, and new unseen cases are classified looking at the class labels of the closest samples to them according to a given similarity metric. The k-NN algorithm experiences a series of difficulties to deal with big datasets, such as high computational cost, high storage requirements, sensitivity to noise and inability to work with incomplete information. Based on MapReduce, different distributed alternatives have recently emerged to enable k-NN to handle Big Data [17, 18, 19], alleviating memory and

[1]Smart Data Discovery Will Enable a New Class of Citizen Data Scientist. `https://www.gartner.com/doc/3084217/smart-data-discovery-enable-new`

computational cost limitations, but these do not reduce the storage requirements or look at the quality of the data.

Another way to simultaneously approach several k-NN weaknesses in Big Data is based on data preprocessing strategies such as data reduction or missing values imputation. The goal of data reduction techniques is to shrink the size of original data in terms of the number of samples (instance reduction [20, 21, 22]) or attributes (feature selection [23, 24]) to mitigate the computational complexity, storage requirements and noise tolerance by eliminating redundant, irrelevant and noisy information. The idea of the k-NN itself takes on an important role within those data reduction algorithms (e.g. by finding discrepancies between nearest neighbours). Dealing with incomplete information such as missing values is a big challenge for most data mining techniques [25], and the k-NN is not an exception as it may not be able to compute distances between examples containing missing values. However, the underlying idea of the k-NN has been used to impute missing values (kNN-I, [26]) based on the k nearest neighbours. To the best of our knowledge, data reduction approaches have been already proposed in the Big Data scenario, but the imputation of missing values with k-NN in Big Data has not been explored so far.

Although most of these data preprocessing techniques were motivated by k-NN drawbacks, it turns out that the resulting "smart" dataset provided by the above approaches can also be of use in many other learning algorithms [27, 25]. This work reviews the current specialised literature that revolves around the idea of the k-NN to come up with Smart Data, greatly extending our preliminary contribution in [28] around this topic. First, we will deepen into the concepts of big and Smart Data and how to extract value from Big Data with existing technologies and Big Data preprocessing techniques (Section 2). Then, we will formally introduce the k-NN algorithm and its main drawbacks to deal with Big Data (Section 3). Next, we will dig into how the k-NN algorithm can be used as a core model for data preprocessing (Section 4), distinguishing between smart reduction of data, smart noise filtering and smart imputation. To characterise the behaviour of these reviewed techniques, we will carry out an extensive experimental evaluation on a number of big datasets (Section 5). To do this, we have used existing Big Data designs of some data preprocessing techniques and we have implemented these on Apache Spark. In addition, in this paper, we design Big Data solutions for those data preprocessing algorithms that were not big data-ready to date (e.g. for missing values imputation). As a result, we have developed a number of new Spark packages that contain big data preprocessing algorithms to perform Smart Reduction[2], Filtering[3] and Imputation[4]. To conclude this review, we discuss current and future trends for the k-NN algorithm (Section 6) in the Big Data context and summarise the main conclusions (Section 7).

2 SMART DATA: FOCUSING ON VALUE IN BIG DATA

This section is first devoted to introducing the main concepts of Big Data technologies as have been established nowadays (Section 2.1). As such technologies are evolving how data is processed, the vast piles of data are being transformed in an accessible form known as Smart Data, which is described in Section 2.2. Thus, thanks to Big Data preprocessing, which includes a large selection of techniques, we are able to clean and transform raw Big Data into Smart Data.

2.1 Big Data Technologies

As stated before, Big Data is typically characterised by a Volume, Velocity, Variety and Veracity (among other V's) that poses a challenge for current technologies and algorithms. The problem of Big Data has

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

[3]`https://spark-packages.org/package/djgarcia/SmartFiltering`

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

many different faces such as data privacy/security, storage infrastructure, visualisation or analytics/mining. In this work, we are interested in Big Data analytics/mining to extract hidden knowledge from Big Data by means of distributed analyses and algorithms. Tackling big datasets with data mining and machine learning algorithms means moving from sequential to distributed systems that can make use of a network of computers to operate faster. However, parallel computation has been around for many years, what is then new with Big Data? "*The principle of data locality*". Traditional High Performance Clusters (HPCs) have provided a way to accelerate computation by means of parallel programming models such as MPI (Message Passing Interface) [29]. Classical HPCs fail to scale out when data-intensive applications are involved, as data will be moved across the network causing significant delays. In a Big Data scenario, minimising the movement of data across the network by keeping data locally in each computer node is key to provide an efficient response. Thus, ideally, each computer node will operate only on data that is locally available.

The MapReduce functional programming paradigm [8] and its open-source implementation in Hadoop [30] were the precursor parallel processing tools to tackle data intensive applications, by implementing the data locality principle. A MapReduce operation is defined in terms of two functions: *map* and *reduce*. These functions work on key/value pairs, which are defined based on the data to be processed and the algorithm to be applied on that data. The map phase applies a user-specified function to each input pair, the result of which is then emitted to the reduce function (also user-specified), grouping those values with the same key. The reduce function merges the values assigned to a particular key together, usually returning a single value per key. Hadoop implements this MapReduce programming model together with a distributed file systems that provides data locality across a network of computing nodes. As a result, the end-user is able to design scalable/parallel algorithms in a transparent way, so that data partitioning, job communication and fault-tolerance are automatically handled. Despite the great success of Hadoop, researchers in the field of data mining found serious limitations when consecutive operations needed to be applied on the same (big) data, reporting a significant slow-down. Many other frameworks have been made available to address these limitations of Hadoop, and one of the most popular platform nowadays is Apache Spark [9]. As a data processing engine, Spark operates with MapReduce-like functions on a distributed dataset, known as Resilient Distributed Datasets (RDDs), which can be cached in main memory to allow for multiple iterations. Spark is evolving very quickly and more efficient APIs such as DataFrames and Datasets are being developed.

Multiple MapReduce-like solutions have been designed to accommodate classical machine learning and data mining techniques to the new Big Data scenario [4]. Broadly speaking, we can find two main approaches: local or global methods. Local approaches are approximations of the original algorithms in which the data is split into a number of smaller subsets and the original algorithm is applied locally. Then, the results from each individual partition are (smartly) combined. Global models, or sometimes known as exact approaches, aim to replicate the behaviour of the sequential version by looking at the data as a whole (and not as a combination of smaller parts). Local approaches typically require a simpler design than global models, but they lose the full picture of the data. Global models could become more robust and precise (depending on the data), but they will also tend to be slower.

2.2 Smart Data through Big Data Preprocessing

Data is only as valuable as the knowledge and insights we can extract from it. Referring to the well-known "garbage in, garbage out" principle, accumulating vast amounts of raw data will not guarantee quality results, but poor knowledge. *Smart data* refers to the development of tools capable of dealing with massive and unstructured data to reveal its value [13]. Once Smart Data is obtained, real time interactions with other business intelligence or transactional applications are affordable, evolving from data-centered to learning organisations, where knowledge is the core instead of data management [12].

In the traditional knowledge discovery process in databases, extracting the *value* in the data was achieved by means of data preprocessing [14]. Big Data preprocessing has now become an open, emergent topic that draws much attention nowadays [31]. Recent efforts are focusing on adapting data preprocessing tasks to Big Data environments, enabling techniques such as feature selection, discretization or sampling algorithms to deal with a high dimensionality and huge sample size. The application of this sort of techniques is the key to move from “Big” to “Smart” Data [13] (Figure 2).

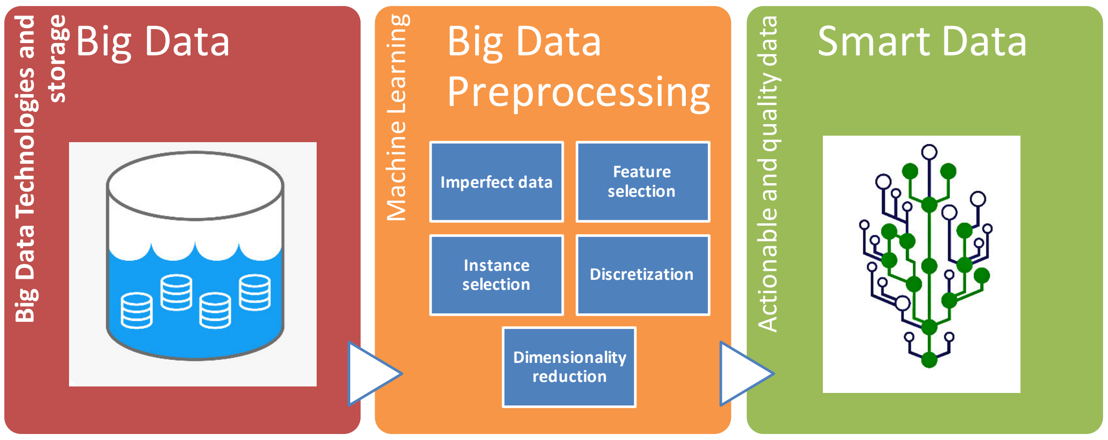


Figure 2: Big data preprocessing is the key to transform raw big data into quality and smart data.

Among all data preprocessing approaches, data integration is the first step when transforming Big Data to Smart Data. It aims to unify the semantics and domains from heterogeneous sources under a common structure. In order to support this process, the usage of ontologies has recently emerged as a popular option [32, 33]. Graph databases are also a common choice, storing the data in a relational schema, especially in health care domains [34].

Even when the integration phase ends, data may still be far from being “smart”. As data grows (especially in dimensionality), noise accumulates [35] and algorithmic instability appears [36]. Thus, in order to be “smart”, the data still needs to be cleaned even after its integration. Currently, there is a lack of proposals for noise cleaning in Big Data environments as finding efficient solutions in this scenario is challenging [37].

To focus on the valuable data, the application of data reduction techniques aims to remove redundant or contradictory examples. Fortunately, the most relevant reduction techniques already have Big Data solutions: feature selection algorithms [24, 38, 39], instance selection techniques [22] and discretization procedures [40].

We also acknowledge that class imbalance acquires a new dimension in Big Data, where the overwhelming amount of majority examples mislead learning algorithms. While data resampling may work on Big Data frameworks [41, 42], the introduced artificial minority examples increment the data size. For this reason, novel preprocessing approaches are being explored by researchers [43].

The implementation of the aforementioned approaches is gathered in specialised packages of the main Big Data programming frameworks (such as Spark’s MLlib [44]). Despite all these progresses, challenges are still present to fully operate a transition between Big Data to Smart Data. The lack of a universal tool that can broadly be applied with robustness and ease in different domains and problem typologies motivate the current paper. We postulate the usage of the k-NN algorithm, a simple yet powerful technique, sits at the core of many preprocessing tasks that will help practitioners to achieve Smart Data.

3 THE K-NN ALGORITHM

A useful and well-known method for supervised learning is based on the computation of the k nearest neighbours to predict a target output (i.e. a class label) that a queried object or sample should have by the principle of similarity [45]. The k-NN algorithm infers the target output of new objects according to the result of the nearest samples or the outcome of several nearest objects in the feature space of a training set. The k-NN algorithm is a non-parametric method that can manage both classification and regression problems as one of the simplest of all machine learning algorithms [15]: a sample is classified by estimating the majority vote of its neighbours, with the new object assigned to the class that is most common among its nearest neighbours (k being a positive integer, and typically small). A formal notation for k-NN in classification is as follows:

Let TR be a training dataset and TS a test set, they are formed by a determined number **n** and **t** of samples, respectively. Each sample $\mathbf{x}_p$ is a vector $(\mathbf{x}_{p1}, \mathbf{x}_{p2}, ..., \mathbf{x}_{pD}, \omega)$, where, $\mathbf{x}_{pf}$ is the value of the f-th feature of the p-th sample. Every sample of TR belongs to a known class ω, while it is unknown for TS. For every sample included in the TS, the k-NN algorithm calculates the distance between this and all the samples of TR. The Euclidean distance is the most used distance function. Thus, k-NN takes the k closest samples in TR by ranking in ascending order according to the distance. Then, the simplest approach computes a majority voting with the class label of the k nearest neighbours.

The k-NN algorithm belongs to the family of lazy learning [16], which means that it does not carry out an explicit training phase (i.e. it does not need to build a model) and new unseen cases are classified on-the-fly by comparing them against the entire training set. In spite of its simplicity, the k-NN is known because it usually offers a good performance in a wide variety of problems. However, this method becomes very sensitive to the local structure of the training data (that needs to be kept stored on a drive). Thus, the classical k-NN algorithm suffers from a number of weaknesses that affect its accuracy and efficiency.

Computing similarity between samples correctly is key for the k-NN to perform well. Its accuracy may be heavily affected by the presence of noisy or irrelevant features. The nature and number of the input variables (i.e. numerical vs. categorical) and their variety of ranges highly complicate distance computation. Therefore, as many other classifiers, the k-NN algorithm is influenced by the so-called curse of dimensionality and plenty of research has been devoted to overcoming this [46]. More advanced versions of the k-NN algorithm that are capable of improving the performance could be found by varying the voting weights, neighbourhood sizes, similarity metrics, etc [47, 48, 49].

In terms of efficiency, the k-NN algorithm also presents severed issues to handle large-scale datasets. The two main problems found are:

- Memory consumption: In addition to the data storage requirement (on secondary memory), it needs to have the training raw dataset allocated in main memory for fast distance computations. Although preliminary distance computations can be conducted and stored, when TR and TS sets are really big, they can easily exceed the available memory in the computer. Furthermore, the preliminary distance computations do not work in dynamic environments when the training data is continually changing over time.

- Computational cost: The complexity to obtain the nearest neighbour samples of a single test instance in the training set is $\mathcal{O}((n \cdot D))$, where n is the number of training instances and D the number of features. In order to find the k closest neighbours, we typically need to maintain a priority queue with the top nearest neighbours, adding a complexity of $\mathcal{O}(n \cdot log(k))$ where the binary search needed for the queue update adds the log complexity while comparing against the n examples. This computational cost is for every test sample we want to classify, so the classification time is linear with respect to the size of the test dataset.ç

Multiple approaches have been proposed in the literature to accelerate the k-NN algorithm ranging from data reduction (See Section 4.1) to approximate versions (See Section 6.2). In Big Data environments, [19] proposed a technological solution based on Apache Spark [9] for the standard k-NN algorithm to partly alleviate some of problems stated above (memory consumption and computation cost) by means of a distributed computation of nearest neighbours. This exact (global) Big Data version of the k-NN algorithm does not tackle the sensitivity to noisy data, and data storage requirements.

However, as stated before, a proper and aimed use of the k-NN algorithm can help us to achieve the so-called Smart Data. This is because the k-NN algorithm can easily be integrated into more complex processes as simple local operations to make decisions able to enhance and adapt the data to the actual requirements. Next, we will describe the k-NN algorithm as a useful instrument to procure Smart Data.

4 THE K-NN ALGORITHM AS A TOOL TO TRANSFORM BIG DATA INTO SMART DATA

The idea of computing k nearest neighbours has been extensively used to carry out data preprocessing. In most of the cases, existing techniques were originally designed to tackle the weaknesses of the k-NN algorithm mentioned in the previous section. However, these methods may act as general data preprocessing techniques that help us to get rid of unnecessary data and refine imperfect raw data to obtain useful (smart) data. In what follows, we discuss two different scenarios in which the k-NN algorithm has been applied to reduce data size (Subsection 4.1) and correct data imperfections (Subsection 4.2).

4.1 Data reduction with the k-NN algorithm

Data reduction encompasses a set of techniques devoted to reducing the size of the original data whilst retaining as much information as possible. These techniques are used to both obtain a representative sample of the original data, as well as to alleviate data storage requirements. This process does not only obtain a relevant sample of the original data, but also aims at eliminating noisy instances, and redundant or irrelevant data, improving the later data mining process.

In the literature, there are two main approaches to perform data reduction consisting of reducing the number of input attributes or the instances. Focusing on reducing attributes, the most popular data reduction techniques are Feature Selection (FS) and feature extraction [23], which are designed to either select the most representative features or construct a new whole set of them. Similarly, from the instances point of view, we can differentiate between Instance Selection (IS) methods [20], and Instance Generation (IG) methods [21]. The objective of an IS method is to obtain a subset $SS \subset TR$ such that SS does not contain redundant or noisy examples and $Acc(SS) \simeq Acc(TR)$, where $Acc(SS)$ is the classification accuracy when using SS as the training set. Likewise, IG methods may generate artificial data points if needed for a better representation of the training set. The purpose of an IG method is to obtain a generated set IGS, which consists of p, $p < n$, instances, which can be either selected or generated from the examples of TR.

In this subsection, we focus on those data reduction methods that are inspired by the weaknesses of the k-NN algorithm. Most existing instance reduction methods were actually conceived to address those shortcomings. Prototype Selection (PS) methods are IS methods that use an instance-based classifier with a distance measure, commonly k-NN, for finding a representing subset of the training set. One of the classic and most widely used algorithms for PS is the Fast Condensed Nearest Neighbour (FCNN), which is an order-independent algorithm to find a consistent subset of the training dataset using the NN rule [50]. Another simple yet powerful example is the Random Mutation Hill Climbing (RMHC) [51], it randomly selects a subset of the training data and performs RMHC iteratively to select the best subset using k-NN as a classifier. The IS problem can be seen as a binary optimisation problem which consists of whether or not to select a training example [52]. For this reason, evolutionary algorithms have been used for PS, with very promising results. In these algorithms, the fitness function usually consists of classifying the whole training set using the k-NN algorithm [27]. To date, one of the best performing algorithms for evolutionary PS is [53], which is a steady-state memetic algorithm (SSMA) that achieves a good reduction rate and accuracy with respect to classical PS schemes.

Another approach to perform instance reduction is IG, also called Prototype Generation (PG) in the case of instance-based classifiers. In contradistinction to PS, these methods aim to overcome an addition limitation of the k-NN algorithm: it makes predictions over existing training data assuming they perfectly delimit the decision boundaries between classes (in classification problems). To overcome that limitation, these methods are not restricted to selecting examples of the training data, but they can also modify the values of the instances based on nearest neighbours. The most popular strategy is to use merging of nearest examples to set the new artificial samples [54]. We can also find clustering based approaches [55] or evolutionary-based schemes [56], but the vast majority of them are based on the idea of computing nearest neighbours to reduce the training set. A complete survey on this topic can be found in [21].

In terms of FS, a variety of strategies such as wrappers, filters and embedded methods have been proposed in the literature [57]. Nevertheless, we can still find that the k-NN algorithm has also played an important role in many existing FS proposals [58]. One of the classic and most relevant methods is ReliefF [59] that ranks features according to how well an attribute allows us to distinguish the nearest neighbours within the same class label from the nearest neighbors from each of the different class labels. Similarly to the instance reduction scenario, evolutionary algorithms have also been employed to perform FS with good results. In [60] we can find a complete survey on FS using evolutionary computation.

Hybrid approaches for data reduction have also been proposed in the literature. Instead of using IS and FS methods separately, some research has been devoted to the combination of both IS and FS. In [61], for instance, a hybrid of IS and FS algorithm is presented, using an evolutionary model to perform FS and IS for k-NN classification. Hybrid approaches of PS and PG have also been studied in the literature. In these methods, PS is used for selecting the most representative subset of the training data, and PG is tasked to improve this subset by modifying the values of the instances. In [62] a hybrid combination of SSMA with a scale factor local search in differential evolution (SSMA-SFLSDE) is introduced.

As stated previously, data reduction methods are focused on reducing the size of the original data, facilitating the later data mining processes or actually making them possible in the case of Big Data problems. However, these methods are not prepared to work on Big Data environments, as they were not initially conceived for it. Several approaches have recently emerged to tackle big datasets by means of distributed frameworks such as Hadoop MapReduce [8]. In particular, we can find approaches based on k-NN for Big Data such as [24] where FS is performed on huge datasets using the k-NN algorithm within an evolutionary approach, or a distributed Spark-based version of the ReliefF algorithm [63]. In [64] a parallel implementation of the Democratic IS algorithm (DIS) is presented, called MR-DIS. The idea of DIS algorithm is to apply a classic IS algorithm over a number of equally sized partitions of the training data. Selected instances receive a vote. This process is repeated a number of rounds, and at the end of it, the instances with most votes are removed. Additionally, in [22] a distributed framework named MRPR is proposed to enable the practitioner to perform instance reduction methods on big datasets. This method also splits the big training data into a number of chunks, using a MapReduce process, and IS or IG approaches are locally applied to each chunk. Then, the resulting reduced sets from each split are merged together following different strategies.

In the experimental section of this paper, we will analyse the behaviour of some of the most representative instance reduction approaches based on k-NN when tackling big datasets. MR-DIS and SSMA-SFLSDE were already proposed for Big Data as local models (apply IS and IG algorithms in different chunks of data). For our experiments, the FCNN has been adapted to Big Data using the MRPR framework [22] (same framework used for SSMA-SFLSDE). MRPR and MR-DIS follow a local approach, which means that these methods will operate on separated chunks of data. Due to its simplicity, the RMHC algorithm has been implemented in a global manner based on the kNN-IS [19], so that, it looks at the training data as a whole (although it looks at the data taking iteratively subsets of the whole dataset).

4.2 Handling Imperfect Data

Albeit most techniques and algorithms assume that the data is accurate, measurements in our analogic world are far from being perfect [37]. The alterations of the measured values can be caused by noise, an external process that generates corruption in the stored data, either by faults in data acquisition, transmission, storage, integration and categorisation. The impact of noise in data has drawn the attention of researchers in the specialised literature [65]. The presence of noise has a severe impact in learning problems: to cope with the noise bias, the generated models are more complex, showing less generalisation abilities, lower precision and higher computational cost [66, 67].

Alleviating or removing the effects of noise implies that we need to identify the components in the data that are prone to be affected. The specialised literature often distinguishes between noise in the input variables (namely *attribute noise*) and the noise that affects the supervised features. Attribute noise may be caused by erroneous attribute values, missing attribute values (MVs) and "do not care" values. Note that only in the case of supervised problems the noise in the output variables can exist. In classification, this kind of noise is often known either as *class* or *label noise*. The latter refers to instances belonging to the incorrect class either by contradictory examples [68] or misclassifications [67], due to labelling process subjectivity, data entry errors, or inadequacy of the information used to label each instance. In regression problems, noise in the output will appear as a bias added to the actual output value, resulting in a superposition of two different functions that it is difficult to separate.

MVs, among all the corruptions in input attribute values, deserve special attention. In spite of being easily identifiable, MVs pose a more severe impact in learning models, as most of the techniques assume that the training data provided is complete [69]. Until recently, practitioners opted to discard the examples containing MVs, but this praxis often leads to severe bias in the inference process [70]. In fact, inappropriate MVs handling will lead to model bias due to the distribution difference among complete and incomplete data unless the MVs are appropriately treated. Statistical procedures have been developed to impute (fill-in) the MVs to generate a complete dataset, obeying the underlying distributions in the data. The usage of machine learning approaches to perform imputation, as regressors or classifiers, quickly followed in the specialised literature, resulting in a large set of techniques than can be applied to cope with MVs in the data [25].

The applicability of noise filters or MVs imputations cannot be blindly carried out. The statistical dependencies among the corrupted and clean data will dictate how the imperfect data can be handled. Originally, Little and Rubin [70] described the three main mechanisms of MVs introduction. When the MV distribution is independent of any other variable, we face Missing Completely at Random (MCAR) mechanism. A more general case is when the MV appearance is influenced by other observed variables, constituting the Missing at Random (MAR) case. These two scenarios enable the practitioner to utilise imputators to deal with MVs. Inspired by this classification, Frénay and Verleysen [37] extended this classification to noise data, analogously defining Noisy Completely at Random and Noisy at Random. Thus, methods that correct noise, as noise filters, can only be safely applied with these two scenarios as well.

Alternatively, the value of the attribute itself can influence the probability of having a MV or a noisy value. These cases were named as Missing Not at Random (MNAR) and Noisy Not at Random for MVs and noisy data, respectively. Blindly applying imputators or noise correctors in this case will result in a data bias. In these scenarios, we need to model the probability distribution of the noisy or missigness mechanism by using expert knowledge and introduce it in statistical techniques as Multiple Imputation [71]. To avoid improperly application of correcting techniques, some test have been developed to evaluate the underlying mechanisms [72] but still careful data exploration must be carried out first.

The underlying idea of the k-NN algorithm has served of inspiration to tackle data imperfection. Here, we will distinguish between two main kinds of data imperfection that need to be addressed: noisy data and incomplete data.

4.2.1 Noisy data treatment with the k-NN algorithm

As we have mentioned, the presence of noise involves a negative impact in the model obtained. This effect is aggravated if the learning technique is noise sensitive. In particular, the k-NN algorithm is very sensitive to noise, especially when the value of k is low. The negative effects of noise will also increase as the data size does, since noise accumulates when the dimensionality and number of instances becomes lager [36]. Thus, models obtained from sensitive algorithms will become even weaker in Big Data environments. The solution goes by transforming Big to Smart Data prior to the application of such weak learners.

As we have indicated in Section 4.2, noise filtering is a popular option in these cases, which becomes even more helpful in Big Data environments as noise filters reduce the size of the datasets. However, designing Big Data noise filters is a challenge and only some prior designs and methods can be found in the literature [73, 74]. On the other hand, k-NN has been the seminal method to remove redundant and noisy instances in learning problems. The key idea of k-NN, distance-based similarity, has been recurrently used to detect and remove class noise. Therefore, k-NN seems as a promising starting point to transform Big Data to Smart Data.

The literature in the usage of k-NN to clean datasets is very prolific and span over several categories or topics. For instance, in [20] and [21], the authors categorised noise filtering techniques based on k-NN as sub-families of PS and PG methods: edition-based methods and class-relabelling methods, respectively. The objective of edition-based methods is to only eliminate noisy instances (in contradistinction to more general PS methods that also remove redundant samples), and class-relabelling methods do not always remove the noisy instances, but they may amend those labels that the method found mistakenly assigned [75].

Among all the previous categories, one of the most popular methods is the Edited Nearest Neighbour (ENN) [76], which removes all incorrectly labelled instances that do not agree with their k nearest neighbours. If the labels are different, the instance is considered as noisy and removed. Other relevant examples of this family of methods are: All-kNN [77], NCN-Edit [75] or RNG [78]. A distributed version of the ENN algorithm based on Apache Spark is proposed in [74] for very large datasets. This distributed version of ENN performs a global filtering of the instances, considering the whole dataset at once. The time complexity of this method is reduced to the same time complexity of the k-NN.

In the experimental section of this work, we make the All-KNN global as ENN, as this algorithm basically consists of applying multiple times ENN. However, for NCN-Edit and RNG, further investigation would be required to design them as global approaches. Thus, these two methods will be considered within the MRPR framework proposed in [22] to make them scalable to Big Data.

4.2.2 Missing values imputation with the k-NN algorithm

There are different ways to approach the problem of MVs. For the sake of simplicity, we will focus on the MCAR and MAR cases by using imputation techniques, as MNAR will imply a particular solution and modelling for each problem. When facing MAR or MCAR scenarios, the simplest strategy is to discard those instances that contain MVs. However, these instances may contain relevant information or the number of affected instances may also be extremely high, and therefore, the elimination of these samples may not be practical or even bias the data.

Instead of eliminating the corrupted instances, the imputation of MVs is a popular option. The simplest and most popular estimate used to impute is the average value of the whole dataset, or the mode in case of categorical variables. Mean imputation would constitute a perfect candidate to be applied in Big Data environments as the mean of each variable remains unaltered and can be performed in $\mathcal{O}(n)$. However, this procedure presents drawbacks that discourage its usage: the relationship among the variables is not preserved and that is the property that learning algorithms want to exploit. Additionally, the standard error of any procedure applied to the data is severely underestimated [70] leading to incorrect conclusions.

Further developments in imputation are to solve the limitations of the two previous strategies. Statistical techniques such as Expectation-Maximisation [79] or Local Least Squares Imputation [80] were applied in bioinformatics or climatic fields. Note that imputing MVs can be described as a regression or classification problem, depending on the nature of the missing attribute. Shortly after, computer scientists propose the usage of machine learning algorithms to impute MVs [25].

One of the most popular imputation approaches is based on k-NN (denoted as kNN-I) [26]. In this algorithm, for each instance that contains one or more MVs, it calculates the k nearest neighbours and the gaps are imputed based on the existing values of the selected neighbours. If the value is nominal or categorical, it is imputed by the statistical mode. If the value is numeric, it will be imputed with the average of the nearest neighbours. A similarity function is used to obtain the k nearest neighbours. The most commonly used similarity function for missing values imputation is a variation of the Euclidean distance that accounts for those samples that contain MVs. The advantage of kNN-I is that is both simple and flexible, requiring few parameters to operate and being able to use incomplete instances as neighbours. Most imputation algorithms only utilise complete instances to generate the imputation model, resulting in an approximate or biased estimation when the number of instances affected by Mvs is high.

The proposal of imputation techniques in Big Data is still an open challenge, due to the difficulties associated to adapt complex algorithms to deal with partial folds of the data without losing predictive power. At this point, MVs pose an important pitfall in the transition from Big to Smart Data. To the best of our knowledge, there has not been proposed a way of applying kNN-I on big datasets. Although further investigation is required, we propose a simple yet powerful approach to handle MVs with the kNN-I algorithm on Big Data problems, which will be called k Nearest Neighbours Local Imputation (kNN-LI). Figure 3 shows the workflow of the algorithm. Due to the scalability problems to tackle the Euclidean distance with MVs, the proposed kNN-LI algorithm follows a divide and conquer scheme under the MapReduce paradigm and it is implemented under the Apache Spark platform. It begins by splitting and distributing the dataset between the worker nodes. For each chunk of data, we compute the kNN-I method locally with the existing instances. Once all MVs have been imputed for each chunk of the data, the results are simply grouped together to obtain a whole dataset free of MVs. This local design is similar to the one followed in [22] for instance reduction approaches, and allows us to impute MVs in very large datasets. Nevertheless, as a local model we are aware that the quality of the imputation may vary depending on the number of partitions considered. A preliminary global version of this imputator is also available on the provided Spark Package, but we have not included this in our experiments because it does not cope well with the used datasets.

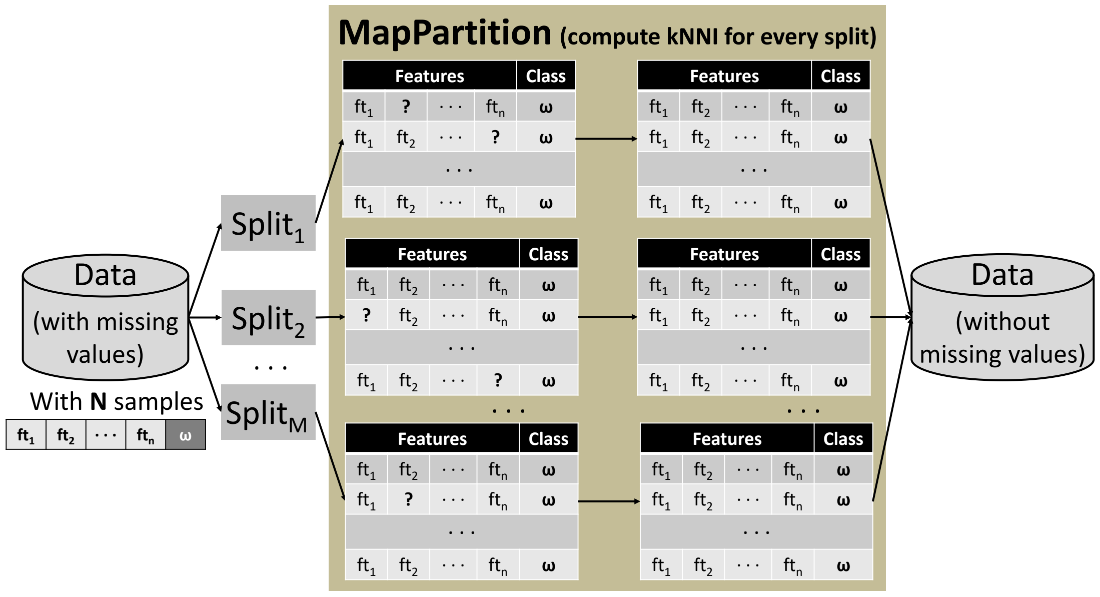

MapPartition (compute kNNI for every split)
Data (with missing values)
With N samples
Split1
Split2
SplitM
Features
Class
Data (without missing values)

Figure 3: Flowchart of the kNN-LI algorithm. The dataset is split into M chunks (Map function) that are processed locally by a standard kNN-I algorithm. The resulting amended partitions are then gathered together.

5 EXPERIMENTAL STUDY AND ANALYSIS OF RESULTS

The effectiveness of k-NN-based algorithms to obtain smart data has been widely analysed in small and medium datasets. This section presents different case studies that show the potential of the k-NN algorithm as a unique and simple idea to obtain Smart Data from big amounts of potentially imperfect data. After presenting the details associated to the experimental study in Subsection 5.1, we conduct a series of experiments with relevant methods for smart instance reduction (IS and IG) in Subsection 5.2, noise filtering in Subsection 5.3 and missing values imputation in Subsection 5.4. All the implementations of the techniques analysed in this section are available as Spark Packages for public use. Note that most of the used algorithms were already designed under MapReduce, and we implemented them on Spark to improve their efficiency. For those algorithms for which there was not a MapReduce-like design available, we proposed in the previous section a Big Data solution for them.

5.1 Experimental set-up

In this section, we show all the details related to the experimental set-up, introducing the problems selected for the experimentation, the performance measures and the parameters of the methods used. In addition, we detail the hardware and software resources used to carry out the experiments.

We have selected 7 Big Data classification problems. The ECBDL'14 dataset is extracted from a Big Data competition [81]. This is a binary classification problem with a high imbalance ratio (>45). As we are not focused on the imbalanced problem in this experimental study, we have randomly sampled the dataset to obtain a more balanced ratio of 2 (meaning that there will be the double of examples from one class w.r.t the other). We selected this dataset because of its relatively high number of features and to analyse how this characteristic affects to our experiments. The remaining 6 datasets are extracted from the UCI machine learning and KEEL datasets repositories [82, 83]. Table 1 presents the number of examples, number of features, and the number of classes ($\#\omega$) for each dataset. All datasets have been partitioned using a 5 fold cross-validation scheme. This means that each partition includes 80% of samples for training and 20% of them are left out for test.

Table 1: Summary description of the datasets

Dataset	#Examples	#Features	$\#\omega$
ECBDL'14	2,063,187	631	2
Higgs	11,000,000	28	2
Ht-sensor	928,991	11	3
Skin	245,057	3	2
Susy	5,000,000	18	2
Watch-acc	3,540,962	20	7
Watch-gyr	3,205,431	20	7

To assess the scalability and performance of the experimental study, we use the following measures:

- *Accuracy:* It represents the percentage of correctly classified instances with respect to the total of test samples.
- *Reduction Rate:* This shows the percentage of data w.r.t to the original training data that has been removed as it is considered redundant or noisy. It has a strong influence on the efficiency and efficacy of the latter classification step.
- *Runtime:* This measure collects the execution time invested by the algorithms. All analysed algorithms have been run in parallel on Spark, and the runtime includes reading and distributing the dataset across a cluster of computing nodes. The runtime highly depends on the number of partitions established. In this work, we are not focused on performing a study of the scalability of the methods, so that, we have set up a fixed number of partitions depending on the type of method and datasets (see details in the following subsections).

As stated before, we test the performance of data reduction techniques, noise filters and missing value imputation methods on a Big Data scenario following a variety of designs (i.e. local or global model). As a summary of the Spark packages provided in this paper, Table 2 briefly describe all the analysed methods. The parameters used by these algorithms are summarised in Table 3. As recommended by different authors, we have focused on a k = 1 for data reduction methods, k = 3 for noise filtering, and we explore a number of k values for missing values imputation.

After the preprocessing stage, we need to apply a classifier to analyse the impact of removing noise, redundant data or imputing missing values. In this work, we have focused most of the experiments on a distributed version of a Decision Tree [85], which is available on the Machine Learning library of Apache Spark [9]. We have selected this algorithm because it is able to learn a Decision Tree globally, meaning that all the data is used at once to build a Decision Tree. In addition, its learning and classification runtimes are typically lower than other machine learning techniques. Whenever possible we have also included the results of applying the k-NN algorithm using the resulting preprocessed datasets. Table 4 shows the parameters used for these classifiers.

The cluster used for all the experiments performed in this work 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 Intel Xeon CPU E5-2620 processors, 6 cores (12 threads) per processor, 2 GHz and 64 GB of RAM. The

Table 2: Spark Packages for Smart Data Gleaning: Brief description of the MapReduce-based methods included in these packages

Smart-Reduction:	`https://.org/package/djgarcia/SmartReduction`
FCNN_MR [50]	This method applies FCNN locally in separate chunks of the data, using the MRPR framework [22]. FCNN begins with the centroids of the different classes as initial subset. Then, each iteration, for every instance in the subset, it adds the nearest enemy inside its Voronoi region. This process is repeated until no more instances are added to the subset. The resulting reduced sets from each chunk is joined together.
MR-DIS [64]	MR-DIS applies a Condensed Nearest Neighbour algorithm [84] repeatedly to each partition of the data (locally). After each round, selected instances receive a vote. The instances with the most votes are removed.
SSMA-SFLSDE [62]	Following a local MRPR approach, this performs an IS phase to select the most representative instances per class. Then the particular of positioning of prototypes is optimised with a differential evolution algorithm. The resulting partial reduced sets are joined together.
RMHC_MR [51]	This is implemented as a global model. It starts from a random subset, and at each iteration, a random instance from the sample is replaced by another from the rest of the data. If the classification accuracy is improved (using the global k-NN), the new sample is maintained for the next iteration.
Smart-Filtering:	`https://spark-packages.org/package/djgarcia/SmartFiltering`
ENN_MR [76]	ENN_MR performs a global 1NN (based on [19] to the input data and removes examples whose nearest neighbour does not agree with its class.
All-kNN_MR [77]	All-KNN performs ENN_MRrepeatedly with different values of *k*, removing instances whose class differ from its nearest neighbours. Therefore, this is also a global model.
NCNEdit_MR [75]	It follows a local MRPR scheme, using the original NCNEdit algorithm to discard misclassified instances using the *k* nearest centroid neighborhood classification rule with a leave-one-out error estimate for separate chunks of the data.
RNG_MR [78]	RNG_MR also follows a local MRPR approach, so that, the RNG_MR computes a proximity graph of the input data. Then, all the graph neighbours of each sample give a vote for its class. Finally, mislabeled examples are removed.
Smart-Imputation:	`https://spark-packages.org/package/JMailloH/Smart_Imputation`
kNN-LI [26]	This approach splits the data into different chunks, and applies the original kNN-I on each partition. This means that for each instance containing a MV, the imputation is based on the values of its *k* nearest neighbours.

network used is Infiniband 40Gb/s. The operating system is Cent OS 6.5, with Apache Spark 2.2.0. and the maximum number of concurrent operations is equal to 256 and 2 GB for each task.

Table 3: Parameter settings for the data preprocessing algorithms utilised

Algorithm	Parameters
FCNN_MR	k = 1, ReducerType = Join
MR-DIS	k = 1, numRep = 10, alpha = 0.75, dataPerc = 1.0
SSMA-SFLSDE	PopulationSFLSDE = 40, IterationsSFLSDE = 500, iterSFGSS = 8
	iterSFHC = 20, Fl = 0.1, Fu = 0.9, ReducerType = Join
RMHC_MR	k = 1, iterations = 300, p = 0.1, ReducerType = Join
ENN_MR	k = 3
All-kNN_MR	k = 3
NCNEdit_MR	k = 3, ReducerType = Join
RNG_MR	graphOrder = 1st order, selType = edition, ReducerType = Join
kNN-LI	k = 3, 5 and 7

Table 4: Parameter settings for the base classifiers

Classifier	Parameters
Decision Tree	impurity = “gini”, maxDepth = 20 and maxBins = 32
k-NN	k = 1

5.2 Smart Instance Reduction

The aim of this section is to analyse the behaviour of relevant instance reduction methods and characterise their capabilities in Big Data classification in terms of accuracy performance, reduction rate and computing times obtained. We compare the four data reduction algorithms described in Table 2, using both Decision Trees and k-NN as base algorithms. It is important to recall at this point a few characteristics about the Big Data implementations used in this paper. MR-DIS, SSMA-SFLSDE and FCNN_MR will act as local models, handling the data in a divide-and-conquer fashion. The RHMC_MR implementation is, however, a global approach capable of looking at the whole training data as a single piece. The number of partitions has been set to a number that results in no less than 1000 examples per partition. We acknowledge that local models may be affected by the number of partitions considered as discussed in [22] and further investigation may be required. Nevertheless, this has established a fair comparison framework for all the considered data reduction techniques.

Table 5 summarises the results obtained with all the data reduction algorithms using a Decision Tree as a classifier. It shows the accuracy and reduction rate obtained on test. For each dataset, we also include the result of applying the Decision Tree algorithm without applying any preprocessing (denoted as Baseline). Decision trees are known to be very sensitive to instance reduction techniques, as they have less instances to consider when splitting. Therefore, we may expect some accuracy drops when instance reduction techniques are applied. As a way of quantifying the reduction rate impact, Figure 4 plots the data storage reduction (in Gigabytes) for all tested instance reduction methods on the ECBDL’14 dataset. Looking in detail at these results, we can draw the following conclusions from the results:

- The main goal of this type of techniques is to widely reduce the amount of data samples that we keep as training data. However, the analysed algorithms work quite differently and they lead to very different accuracy and reduction rates. As we can see, for the same dataset, depending on the technique used, the reduction rate may vary from 22% to 96% of reduction. This shows the

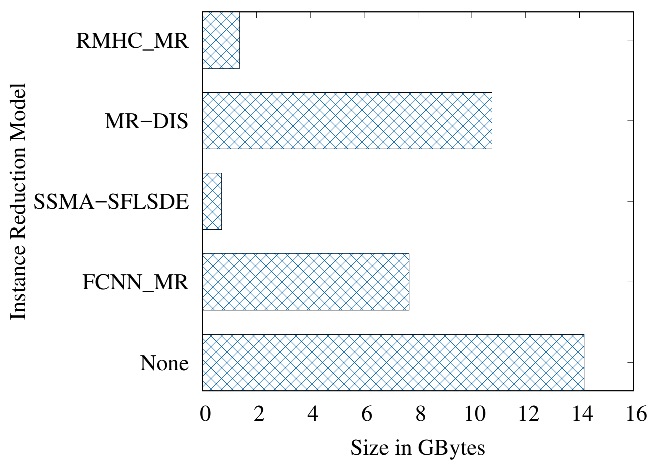


Figure 4: Storage Requirements Reduction on ECBDL'14 dataset

importance of choosing the right technique depending on whether our objective is to reduce data size or our focus is on obtaining a high accuracy.

- On average, the SSMA-SFLSDE algorithm provides the highest reduction rates, achieving up to 98.6% reduction without a significant loss in accuracy. For example, on the skin dataset, SSMA-SFLSDE is able to find a subset of roughly 2700 instances that represents almost perfectly the 196,000 training instances. Figure 4 shows the great impact of SSMA-SFLSDE on ECBLD'14 dataset, in which the training data is reduced from approximately 14GBs to 725MBs.

- MR-DIS is the most conservative algorithm as far as the number of removed instances is concerned, followed by FCNN_MR. These methods are also achieving less accuracy than RMHC_MR and SSMA-SFLSDE. The relatively high accuracy of the RMHC_MR algorithm w.r.t more advanced methods such as MR-DIS and SSMA-SFLSDE may be explained by the fact that the implementation is global, so that, it is able to find redundant data in a more global manner. Nevertheless, we have to recall that RMHC_MR basically subsamples the entire dataset and apply k-NN. Regarding MR-DIS and FCNN_MR, we can see that they have a close performance in accuracy and reduction rates, achieving MR-DIS slightly better performance in some datasets. These similarities are given because MR-DIS uses a condensed nearest neighbour algorithm as instance selection algorithm in the internal process of DIS, which is the cornerstone of FCNN.

- As expected, the application of a Decision Tree on the entire training dataset (raw data) is normally providing a higher accuracy (of course, needing a higher learning time). However, in many of the cases, the drop in accuracy is so reduced and the reduction provided is so high that we conclude that it is worth obtaining Smart Data before applying learning. In particular, on those datasets in which the Baseline is able to obtain a very high accuracy, we can obtain high reduction rates without losing much accuracy.

Since all the tested data reduction algorithms are based on similarity between instances (rather than a comparison at a feature level as the Decision Tree does), they are expected to have a better performance when using a distance-based classifier. In Table 6, we can find the test accuracy results and reduction rate using the k-NN algorithm as a classifier. Baseline now represents the results of the k-NN algorithm without any preprocessing. As we can see, data reduction techniques are performing better in this scenario compared to

Table 5: Impact of Instance Reduction on Decision Trees (Test Accuracy and Reduction Rate)

Dataset	Method	Accuracy (%)	Reduction (%)
ECBDL'14	Baseline	75.85	-
	FCNN_MR	72.59	45.81
	SSMA-SFLSDE	69.63	96.76
	MR-DIS	74.02	24.12
	RMHC_MR	70.08	90.28
Higgs	Baseline	69.94	-
	FCNN_MR	69.37	34.95
	SSMA-SFLSDE	63.89	95.50
	MR-DIS	69.36	24.74
	RMHC_MR	66.89	89.99
Ht_sensor	Baseline	99.98	-
	FCNN_MR	64.96	99.90
	SSMA-SFLSDE	98.74	98.04
	MR-DIS	87.96	99.86
	RMHC_MR	99.76	89.99
Skin	Baseline	99.86	-
	FCNN_MR	99.78	93.23
	SSMA-SFLSDE	99.24	98.62
	MR-DIS	99.77	96.38
	RMHC_MR	99.79	89.91
Susy	Baseline	77.66	-
	FCNN_MR	76.18	41.77
	SSMA-SFLSDE	75.97	95.95
	MR-DIS	76.70	22.65
	RMHC_MR	74.64	89.98
Watch_acc	Baseline	91.22	-
	FCNN_MR	77.07	95.75
	SSMA-SFLSDE	88.16	93.45
	MR-DIS	79.44	97.20
	RMHC_MR	89.51	89.98
Watch_gyr	Baseline	90.35	-
	FCNN_MR	70.92	96.87
	SSMA-SFLSDE	86.54	93.52
	MR-DIS	75.18	97.71
	RMHC_MR	87.18	89.97

the previous study with Decision Trees. None of the data reduction algorithms methods is losing that much accuracy with respect to the baseline accuracy. In fact, in some cases they are able to improve the baseline performance, as they remove redundant and also noisy examples. Analysing further these results, we can conclude that:

Table 6: Impact of Instance Reduction on k-NN (Test Accuracy)

Dataset	Method				
	Baseline	FCNN_MR	SSMA-SFLSDE	MR-DIS	RMHC_MR
ECBDL'14	80.06	74.98	72.03	76.73	69.19
Higgs	58.36	57.64	57.03	57.41	56.57
Ht_sensor	99.99	68.97	99.71	95.85	99.97
Skin	99.95	99.87	99.76	99.86	99.90
Susy	69.35	66.70	70.80	67.24	67.59
Watch_acc	96.40	80.17	89.49	78.31	92.05
Watch_gyr	98.57	88.35	91.76	85.54	95.18

- As happened before with Decision Trees, there is not a clear outperforming method overall. The choice of the right technique crucially depends on the particular problem, and the needs to reduce data storage requirements and precision.
- In Susy dataset, SSMA-SFLSDE is improving the baseline accuracy by 1.5% with close to 96% of reduction. This exemplifies the importance of using data reduction techniques, not only for reducing the size of the data, but also for removing noisy and redundant instances. For datasets with high accuracy such as Skin and Ht_sensor, we can achieve up to 98.6% of reduction without losing accuracy. This allows techniques that could not be applied due to the size of the data, to be used in subsequent processes.

Finally, we analyse the runtime of the four data reduction algorithms to complete the smart reduction of the data size. In Figure 5, we show a graphic representation of these runtimes. Due to the variances in computing times, we have used a logarithmic scale to represent the results. The different working schemes of the algorithms are clearly reflected on these differences.

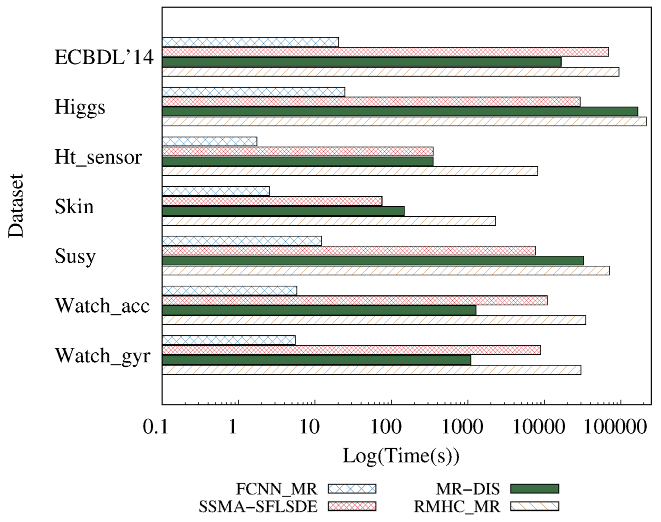


Figure 5: Runtime Chart in Logarithmic Scale to perform Smart Reduction

Overall, the RMHC_MR method is the most time consuming algorithm as it performs k-NN repeatedly in a global fashion. SSMA-SFLSDE is expected to be computationally expensive as it performs PS and PG with evolutionary computation to select the best subset. The fastest method is FCNN_MR, however, the balance between performance and computational cost does not make it the best choice. It is also worthwhile noticing that despite the very high runtime shown by most of these smart reduction algorithms, they will also be applied once on the raw data, and multiple classifiers could later be applied and studied on the resulting datasets (for example optimising parameters, which is typically a very time consuming operation and almost impossible on a Big Data scenario).

In this study, we have analysed four relevant methods to perform data reduction. These algorithms have very different properties between themselves. FCNN_MR is the fastest method with a decent performance for some datasets. The method that achieves the highest reduction rates without a significant drop in accuracy is SSMA-SFLSDE. It can even improve the baseline performance in some cases with up to 95% of reduction. Data reduction techniques have shown to be a powerful solution when facing storage limits or dimensionality restrictions for subsequent data mining.

5.3 Smart Noise Filtering

In this section, the goal is to analyse the performance of four significant smart noise filtering methods. We carry out experiments modifying the original datasets to include five levels of label noise. For each noise level, a percentage of the training instances are altered by randomly replacing their actual label by another label from the pool of available classes. The selected noise levels are 0%, 5%, 10%, 15% and 20%, where a 0% noise level indicates that the dataset was unaltered. Apart from the test accuracy, we also analyse the reduction rate of the datasets after the filtering process, and the runtime of the different methods.

For this study, we have compared four well-known smart noise filtering methods implemented on MapReduce: ENN_MR, All-kNN_MR, NCN-Edit_MR and RNG_MR. It is important to recall here that ENN_MR and All-kNN_MR methods have been implemented following a global approach, which means that they compute nearest neighbours against the entire datasets.

For the NCN-Edit_MR and the RNG_MR algorithms, we have used the MRPR framework [22], so that, these methods are applied locally in different chunks of the data. The results from each partition are simply joined together (following the join reducer offered in [22]). As before, the number of partitions has been established as a number that results in no less than 1000 examples per partition for a fairer comparison between noise filters.

Table 7 shows the test accuracy and reduction rate values obtained by the four noise filtering methods over the seven tested datasets using a decision tree. We also include an extra column, named Original, in which no filtering has been performed. This will help us characterise the influence of noise on these datasets and understand the effect of filtering methods. The best results in each row are highlighted in bold face.

Looking at these results, we can make the following conclusions:

- The usage of a noise treatment techniques improves in most cases the accuracy obtained (w.r.t. the *Original* column) at the same level of noise. This shows that avoiding noise treatment is not usually a good option, since using the appropriate noise filtering method will provide an important improvement in accuracy. However, we can also see that the behaviour of all of the analysed filters on the ECBDL'14 does not provide any improvement w.r.t to the *Original*. This may be due to the high-dimensionality of this dataset (with more than 600 features) in which a k-NN-based filter may not be the most suitable option.

- The Decision Tree has shown some intrinsic robustness against noise (looking at the *Original* column), and filters that are too aggressive remove both noisy and clean instances and reduce its

Table 7: Smart Filtering: Impact of Noise Filters on Decision Trees with different ratios of added noise ((%) Test Accuracy and (%) Reduction Rate)

Dataset	Noise(%)	Original	ENN_MR		All-kNN_MR		NCN-Edit_MR		RNG_MR	
			Accuracy	Reduction	Accuracy	Reduction	Accuracy	Reduction	Accuracy	Reduction
ECBDL'14	0	**75.85**	74.12	46.25	72.33	**69.37**	74.85	34.65	74.16	33.37
	5	**69.26**	68.48	46.52	67.66	**70.12**	67.82	37.06	67.87	35.03
	10	**68.51**	67.56	47.14	67.65	**71.05**	67.14	39.66	67.18	37.45
	15	**67.66**	67.42	47.69	66.70	**71.96**	66.67	41.95	66.87	39.84
	20	**66.43**	66.13	48.20	66.11	**72.68**	65.17	43.95	66.30	41.98
Higgs	0	**69.94**	69.12	49.68	67.51	**74.48**	69.27	46.25	69.13	46.97
	5	**69.66**	68.59	49.69	66.80	**74.56**	68.91	46.96	68.75	47.56
	10	**69.26**	68.13	49.77	66.21	**74.65**	68.32	47.63	68.22	48.09
	15	**68.83**	67.49	49.80	65.59	**74.71**	67.81	48.18	67.49	48.62
	20	**68.28**	66.69	49.84	64.72	**74.78**	67.11	48.65	66.72	49.00
Ht_sensor	0	99.98	**99.90**	1.36	99.06	**66.85**	**99.99**	0.02	**99.99**	0.02
	5	99.85	99.84	17.50	94.62	**73.34**	**99.95**	9.47	**99.95**	9.07
	10	99.72	99.75	24.92	86.70	**76.71**	99.90	18.25	**99.91**	17.49
	15	99.57	99.56	29.34	85.90	**78.46**	99.76	26.43	**99.80**	25.34
	20	99.38	99.24	35.57	83.08	**80.54**	**99.65**	33.69	99.64	32.57
Skin	0	**99.86**	99.81	32.73	99.33	**49.28**	99.83	1.54	99.60	5.61
	5	99.71	99.65	34.69	97.10	**53.94**	**99.81**	10.01	99.44	14.45
	10	99.49	99.28	37.02	90.45	**58.07**	**99.60**	17.90	99.28	22.23
	15	99.27	98.93	39.60	84.05	**61.87**	**99.64**	24.93	99.03	29.50
	20	98.96	97.20	41.82	82.72	**64.86**	**99.39**	31.10	98.36	35.24
Susy	0	77.66	76.83	49.16	75.55	**73.94**	**77.91**	33.92	77.40	36.69
	5	77.19	76.28	49.31	74.34	**74.14**	**77.53**	36.82	77.01	39.13
	10	76.77	75.42	49.43	73.75	**74.32**	**77.11**	39.47	76.51	41.34
	15	76.23	75.81	49.56	73.05	**74.47**	**76.51**	41.81	75.82	43.37
	20	75.61	74.22	49.63	71.86	**74.58**	**75.93**	43.95	75.06	45.12
Watch_acc	0	91.22	90.70	10.37	84.47	**87.05**	90.53	0.01	**91.42**	0.02
	5	90.97	90.29	24.30	81.25	**89.44**	91.06	9.62	**91.10**	9.43
	10	90.49	90.68	32.01	76.85	**90.91**	**91.22**	18.64	90.83	18.34
	15	90.30	90.60	36.67	71.80	**91.73**	**90.78**	27.17	90.35	26.71
	20	89.90	90.17	41.53	69.29	**92.65**	90.41	35.04	**90.63**	34.52
Watch_gyr	0	**90.35**	89.45	10.60	84.97	**87.15**	90.34	0.02	90.21	0.02
	5	89.62	89.10	28.60	81.45	**90.13**	89.94	9.63	**90.34**	9.53
	10	89.37	88.97	32.72	77.98	**90.96**	**90.21**	18.68	90.13	18.50
	15	89.46	88.54	36.61	75.39	**91.79**	**89.81**	27.18	**89.81**	26.98
	20	88.39	88.30	42.13	72.38	**92.62**	89.72	35.16	**88.93**	34.85

performance, since it is able to endure some noise while exploring clean instances. The choice of the noise filtering technique is crucial not to penalise the performance of a Decision Tree.

- The effect of noise is quite variable depending on the dataset, and as the noise level increases, the reduction rate is increased. This means that the noise filtering methods are performing well and detecting the noisy instances. Removing instances at 0% level of noise could mean that the dataset had some noise per se or the filtering algorithm is erroneously removing good instances.

- There is no noise filtering technique that clearly stands out from the rest. NCN-Edit_MR shows a good accuracy performance in five datasets while RNG_MR performs well in four of them. RNG_MR is achieving up to 3% more accuracy than a no noise filtering strategy. ENN_MR, and All-kNN_MR have highlighted as very aggressive noise filters which does not work well with a Decision Tree. Actually, the All-kNN_MR is the filter that removes more instances from the datasets at any noise level. It filters out around 80% of the instances of the datasets. It is probably removing not only noisy instances, but a lot of clean ones, affecting the posterior classification process.

- Looking at the Big Data implementation side of these noise filters. ENN_MR and All-kNN_MR are looking at the data as a whole (global approach), while NCN-Edit_MR and RNG_MR are being applied independently in a number of partitions of the data. It is remarkable that local implementations seem to perform well in this Big Data experiments, which means that without a global view of the data they are able to effectively identify noise data. This could be due to some data redundancy in these big datasets.

These results stress the importance of the use of noise filtering techniques, and how important is to choose the right noise filter. NCN-Edit_MR and RNG_MR have shown to have good performance in accuracy and a more moderate reduction rate, while ENN_MR and All-kNN_MR cannot match the performance of the previous ones.

Looking at the computational cost of these filters, Figure 6 presents a comparison across methods. As the percentage of noise is not a factor that affects the computing times, we show the average result for the five executions per dataset and level of noise. Due to the big differences in computation time between the noise filtering methods, we represent the times using a logarithmic scale.

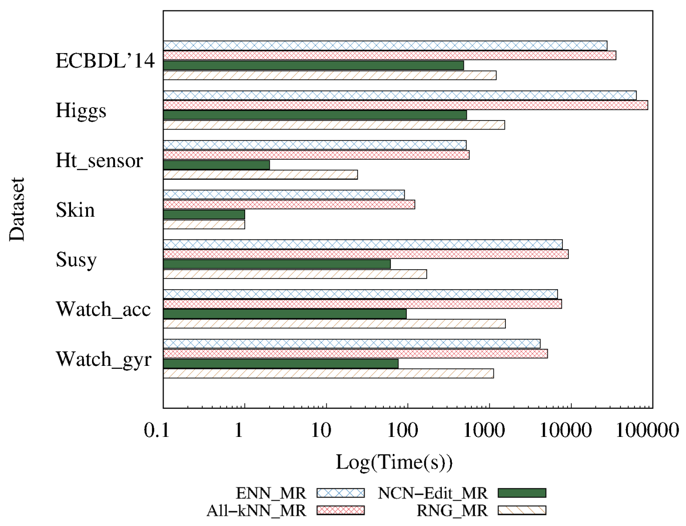


Figure 6: Runtime Chart in Logarithmic Scale to perform Smart Filtering

We can highlight that the NCN-Edit_MR is the most efficient method in terms of computing times. It is closely followed by RNG_MR. Both of them are local Big Data solutions, which approximate the original filtering method. All-kNN_MR and ENN_MR are the most time consuming methods as they have been implemented in a global manner.

In summary, we can conclude that applying a noise filtering technique is crucial in the presence of noise. NCN-Edit_MR and RNG_MR have shown to be the most competitive noise filtering methods, not only in test accuracy and reduction rates, but also in computing times. As big datasets tend to accumulate noise, these methods can be a solution to remove those noisy instances in a reasonable amount of time.

5.4 Smart Imputation of Missing Values

This study is focused on the proposed kNN-LI algorithm to impute missing values in the Big Data context. As detailed previously in Section 4.2.2, we have followed a simple local approach to enable the original kNN-LI algorithm to be run on very big datasets. This subsection is aimed to compare the results of kNN-LI against eliminating affected instances and imputing missing values based on the average/mode value. To do this, we will study the scalability and accuracy with different values of k equal to 3, 5 and 7 on the described datasets. The original training partitions are modified, introducing a 15% and a 30% of instances affected with missing values, using a MCAR mechanism. We will compare the quality of the used techniques to handle MVs using the Decision Tree algorithm as a classifier.

In this experiment, the number of partitions used has been set as follows: for very big datasets such as Susy, Higgs and ECBDL'14 we use 1024 maps; for the rest of datasets we use 256 maps. We set those values after some preliminary experiments in which we determined that a lower number of maps did not really provide any better results in comparison with the runtime needed to execute the algorithm.

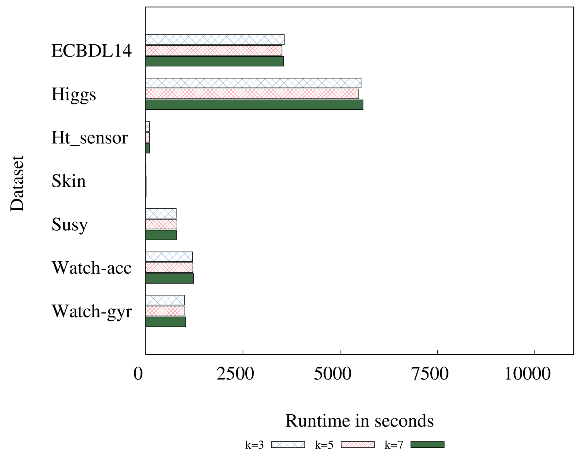
We characterise the proposed local kNN-LI imputator in terms of runtime and precision. Table 8 focuses on studying the quality of the imputation. It presents the accuracy obtained by the Decision Tree classifier in comparison with different techniques to deal with the missing values (ImpTech). First, the column "Original" denotes the results obtained if the dataset were not to have any missing values. Note that in a real situation, this comparison could not be made, but it serves as a reference of (possibly) the maximum accuracy that could be achieved. "Clear" presents the result of eliminating those instances that contain any missing value. "ImputedMean" deals with the missing values by imputing with the average value of a feature (if the feature is continuous) or the mode (if the feature is categorical). Finally, for the proposed kNN-LI we indicate the value of k. The best result of each column is highlighted in bold-face, without taking the Original column into account, because it does not contain MVs, so it should report the best result. To complement this table, Figure 7 shows the imputation runtime in seconds for each dataset. Figure 7a presents the runtime depending on the number of neighbours with 15% of MVs. Figure 7b presents the runtime with $k = 3$ for MVs = 15 and 30 %.

Table 8: Smart Imputation: Imputation quality for Decision Trees (Test Accuracy)

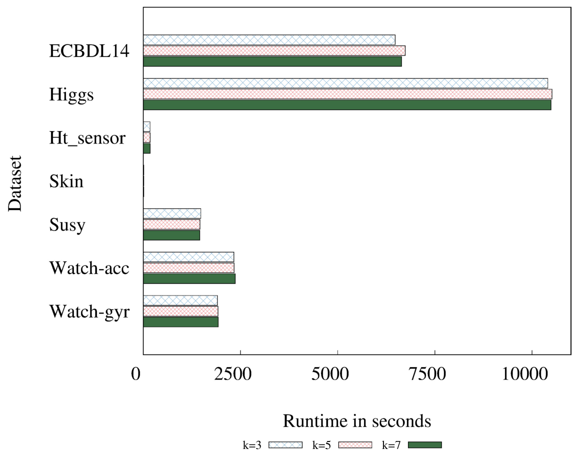
Dataset	MVs%	Original	Clear	ImputedMean	KNN-LI. K=3	KNN-LI. K=5	KNN-LI. K=7
ECBDL'14	15	75.85	75.50	**75.92**	75.89	75.88	75.87
	30	75.85	74.83	75.87	75.94	**75.96**	75.95
Higgs	15	69.94	69.79	69.93	69.97	**69.99**	69.95
	30	69.94	69.62	69.94	69.95	69.97	**70.00**
Ht_sensor	15	99.98	99.98	99.94	99.98	**99.98**	99.96
	30	99.98	99.98	9994	**99.97**	99.96	99.95
Skin	15	99.86	**99.86**	99.80	99.67	99.64	99.70
	30	99.86	**99.85**	99.78	99.54	99.50	99.54
Susy	15	77.66	77.55	77.79	78.03	78.03	**78.03**
	30	77.66	77.32	77.89	78.16	78.16	**78.18**
Watch_acc	15	91.22	91.20	90.81	91.12	91.13	**91.22**
	30	91.22	91.12	90.70	91.03	91.06	**91.30**
Watch_gyr	15	90.35	90.01	89.84	**90.25**	90.08	89.98
	30	90.35	90.02	89.94	**90.10**	89.93	89.92

According to these tables and the figure, we can make the following analysis:

- Focusing on runtime, Figure 7a shows how the value of k does not have a drastic effect on the runtime of the kNN-LI algorithm in any of the datasets. We can also observe in Figure 7b that the imputation



(a) 15% MVs. Different values of k



(b) 30% MVs. Different values of k

Figure 7: Runtime Chart to perform kNN-LI

time of 30% is, in most of the cases, approximately double the runtime to impute 15%. This shows a linear scalability of the kNN-LI model with respect to the number of samples to be imputed. In terms of precision, the imputation performed with different values of k does not provide very significant changes in the behaviour of the Decision Tree classifier in these datasets.

- Analysing the Table 8 we can appreciate that ignoring those instances that contain missing values in most cases reports the worst results. This fact highlights the importance of addressing the problem of missing values in big datasets. Comparing the imputation with the mean and the imputation with the kNN-LI algorithm, it can be seen that the kNN-LI method is the majority of the times reporting the best solution. However, in datasets with very high accuracy results (e.g. Skin and Ht_sensor), accuracy is not recovered with imputation, which is probably due to the noise that may be introduced while imputing. It is also important to note that the imputation performed with kNN-LI allows the Decision Tree to consistently obtain very similar results to the ones provided without any missing value ("Original"). It sometimes happens that the imputation carried out with kNN-LI is even able to outperform that upper-threshold. This might be related to the intrinsic noise of some datasets, which may be somehow alleviated by the imputation.

Table 9: Analysis of the performance of Decision Trees eliminating instances with MVs ("Clear")

Dataset	**Decision Tree Accuracy**					
	0% MVs	15% MVs	30% MVs	50% MVs	60% MVs	70% MVs
ECBDL'14	75.85	75.50	74.83	73.92	73.17	72.49
Higgs	69.94	69.79	69.62	69.27	68.94	68.60
Ht_sensor	99.98	99.98	99.98	99.96	99.95	99.94
Skin	99.86	99.86	99.85	99.84	99.86	99.83
Susy	77.66	77.55	77.32	76.91	76.60	76.31
Watch_acc	91.22	91.20	91.12	90.98	90.66	90.52
Watch_gyr	90.35	90.01	90.02	89.85	89.47	89.15

Although the results presented in the previous tables show that kNN-LI is typically the most appropriate way of handling MVs, the differences in accuracy are however not very significant. To further investigate as to why the imputation is not providing greater advantages, Table 9 presents the accuracy obtained with a greater range of MV percentages, and simply eliminating those instances that are affected ("Clear"). As can be expected, the accuracy obtained with the Decision Tree decreases as the number of instances with MVs is increased. However, this table reveals that the differences between not having any instance affected (0%) and having 70% of the instances affected is quite limited in most of the datasets. Especially, the results on Skin or Ht_sensor datasets do not vary much, indicating that these datasets contain a great number of redundant instances. In other datasets, more significant differences may be found, but still, we cannot expect that the imputation of values will provide a very drastic change in performance. In summary, we could conclude that the imputation of values in big datasets may not be always necessary if the datasets contain too much redundancy. In these scenarios, a Smart Data reduction may be a more suitable option.

6 THE K-NN ALGORITHM IN BIG DATA: CURRENT AND FUTURE TRENDS

This section briefly presents novel trends that are being used to improve the k-NN algorithm in the Big Data context and may serve as an inspiration to develop new Smart Data techniques. Only a few classical approaches to improve the effectiveness of the k-NN algorithm have been explored so far for big amounts of data, and they typically end up adding some additional computation that makes them even more computationally expensive. Similarly, classical approximate k-NN algorithms are under-explored, but recently a few approaches have shown to massively improve the efficiency of this technique in Big Data. Here we postulate that the integration of both trends - more effective and faster k-NN - within the analysed data preprocessing techniques may result in faster and more reliable models in Big Data. Section 6.1 is focused on different alternatives that have already been proposed to boost the accuracy of the k-NN algorithm in the Big Data scenario, and Section 6.2 looks at the acceleration of the search of neighbours.

6.1 Enhancing the correctness of the k-NN

Many different approaches have been proposed to improve the effectiveness of the standard k-NN algorithm. One of the key ideas to do this lies in the fact that the standard k-NN algorithm considers all neighbours equally important when making a final classification. In the literature we can find a variety of strategies to tackle that issue including different similarity measures [86, 87], neighborhood sizes [49] or weighting approaches [88, 47].

A very successful way to jointly handle similarity, neighborhoud sizes and weighting in a single idea is the use of Fuzzy sets. Fuzzy-based k-NN approaches have been widely studied in the literature [89] to account for this issue, and in its easiest form - the Fuzzy k-NN algorithm [90] - it computes a class membership degree for each single training sample, using that information to weigh the importance of the nearest neighbours. This simple idea has empirically highlighted as one of the most powerful fuzzy-based approaches to improve the k-NN algorithm [89].

To the best of our knowledge, the Fuzzy k-NN algorithm has been the first enhanced k-NN-based algorithm that has been made available in the literature to handle the Big Data scenario. The approach presented in [91] is focused on designing a Big Data version of the Fuzzy k-NN that resolves memory restrictions and allows us to apply the original algorithm in a timely manner by using Spark-based parallelisation. This algorithm improves upon the exact parallel k-NN algorithm [19] in terms of accuracy, but it significantly increases its computational costs, as Fuzzy k-NN adds a preliminary stage to compute class memberships.

In summary, if the classic k-NN algorithm has served as a tool for obtaining Smart Data, the improvements in terms of accuracy, such as the Fuzzy k-NN and derivatives, will be very useful to delve into this purpose and to obtain higher quality data alternatives. However, these methods do not reduce the storage requirements, and they actually slow down the original k-NN algorithm by adding some extra computations, which may be a handicap for their successful application in Big Data, and accelerating these will be key for an effective approach.

6.2 Accelerating the k-NN algorithm

Apart from reducing the size of the training data, the acceleration of the k-NN algorithm has also been approached by means of approximate algorithms [92]. Typically focused on domains with a large dimensionality [93], multiple methods have been proposed to perform a search that is approximate in nature, and therefore, assume that the actual nearest neighbours may not be found but they should be sufficiently close. Relaxing the goal of finding exactly the nearest neighbours allows to significantly run faster a search of nearest neighbours. Many of those methods are based on indexing [94], constructing a multi-dimensional index structure that provides a mapping between a query sample and the ordering on the clustered index values, speeding up the search of neighbours.

When a Big Data problem is presented as a domain with a large number of characteristics, dimensionality reduction approaches may be needed [95] to accelerate distance compensations in nearest neighbours classification. The Locality-sensitive hashing (LSH) [93] algorithm is a well-known example that reduces the dimensionality of the data using hash functions with the particularity of looking for a collision between instances that are similar. This adds an additional precomputing stage to the training set, transforming it before applying the hash functions to reduce the dimension of the problem. This results in a reduced scalability of the LSH algorithm whenever a (high-dimensional) dataset contains a high number of instances. An implementation of the LSH algorithm for Big Data is available within the MLlib [44], and another implementation can be found at `https://github.com/marufaytekin/lsh-spark`.

When the data is characterised by a high number of instances, tree indexing approaches may be more suitable than LSH. Many tree-based variants have been designed to accelerate the k-NN algorithm ranging from k-dimensional trees [96] that perform axis parallel partitions of the data, metric trees [97] which split the data with random hyperplanes, to spill-trees [98] - a variant of metric trees where the children nodes can share objects. In [99], a hybrid spill tree is proposed to compute parallel k-NN, hybridising metric trees and spill trees to speed up the classification and maintain a good performance. This approximate approach dramatically reduces the computational costs of the k-NN algorithm in a Big Data context with a high number of instances. An open-source implementation of this hybrid spill tree is available at `https://spark-packages.org/package/saurfang/spark-knn`.

As we have seen, the efficiency of k-NN as analytic technique is low and it will suffer from drawbacks when it is embedded into data preprocessing tasks. The approaches based on approximations will be appealing solutions to address Big Data scenarios, as computing exact nearest neighbours may not be that necessary in a domain composed of massive amount of data, being approximations faster and performing at a similar level in terms of accuracy. In this sense, much work still needs to be done in this field.

7 CONCLUSIONS

In this work, we have discussed the role of one of the simplest data mining techniques – the k nearest neighbour algorithm – as a powerful tool to obtain "Smart Data", which is data with a high quality to be mined. Initially focused on the own k-NN issues, researchers have developed numerous data preprocessing algorithms to reduce the influence of noise, impute missing values or eliminate redundant information to speed up the execution of this algorithm. Many of these data preprocessing techniques have been based on the underlying working of the k-NN algorithm allowing for a simple but effective preprocessing process.

These processes have turned out to be useful not only for the k-NN algorithm for what many of them were initially designed, but also for many other data mining techniques. We have reviewed the existing literature with a focus on the use of these techniques on the Big Data scene, in which extracting Smart Data is essential for a sustainable storage and a fast mining process. We have selected and implemented a number of relevant k-NN-based data preprocessing techniques under Apache Spark (publicly available as Spark Packages), and we have conducted an empirical analysis of the behaviour of these techniques in a series of big datasets, which will allow practitioners and non-experts in the field to determine what kind of Smart Data preprocessing techniques they should be using when dealing with big datasets. In addition to specific conclusions achieved in the previous section, several remarks and guidelines can be suggested:

- Data redundancy seems to be a key issue in most of the investigated datasets. Transforming these big amounts of information into smaller datasets heavily reduce the data storage requirements and the time needed to perform high quality data mining.
- The appearance of noisy data damages the performance of most data mining methods, and its cleaning in a Big Data scale is possible by means of simple k-NN-based filters.
- Having missing values in a Big Data context may deteriorate the performance of any data mining process. However, in the case of severe redundancy of data, our experiments have shown that, although the imputation will typically improve the final accuracy, the absolute gain would not be extremely significant.

Finally, we have briefly covered some of the latest trends for the k-NN algorithm in Big Data, and discussed some of the potential improvements in terms of accuracy and acceleration that may be useful to develop new Smart Data preprocessing techniques.

As future work, we foresee that ad-hoc instance reduction algorithms may be needed for specific data mining algorithms to do a more tailored smart data reduction. In terms of noise filtering and correction, fusion and ensemble-like techniques [100] may be key to better handle noise in a Big Data scale. Also, we would like to investigate the effect of missing values in more complex problems such imbalanced classification, in which data scarcity may still happen for a particular class, and imputation methods may be even more needed.

Acknowledgments

This work is supported by the Spanish National Research Project TIN2017-89517-P and the Foundation BBVA project 75/2016 BigDaP-TOOLS - "Ayudas Fundación BBVA a Equipos de Investigación Científica 2016". J. Maillo holds a FPU scholarship from the Spanish Ministry of Education.

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

[3] G. P. Figueredo, I. Triguero, M. Mesgarpour, A. M. Guerra, J. M. Garibaldi, and R. I. John. An immune-inspired technique to identify heavy goods vehicles incident hot spots. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(4):248–258, 2017.

[4] 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.

[5] Vivien Marx. Biology: The big challenges of big data. *Nature*, 498(7453):255–260, June 2013.

[6] Alberto Fernández, Sara del Río, Victoria López, Abdullah Bawakid, María José del Jesús, José Manuel 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.

[7] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[8] J. Dean and S. Ghemawat. Map Reduce: A flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[9] 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*, pages 1–14, 2012.

[10] 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.

[11] Preeti Gupta, Arun Sharma, and Rajni Jindal. Scalable machine learning algorithms for big data analytics: a comprehensive review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 6(6):194–214, 2016.

[12] Fernando Iafrate. *A Journey from Big Data to Smart Data*, pages 25–33. Springer International Publishing, 2014.

[13] Alexander Lenk, Leif Bonorden, Astrid Hellmanns, Nico Rödder, and Stefan Jähnichen. Towards a taxonomy of standards in smart data. In *Proceedings of the 2015 IEEE International Conference on Big Data (Big Data)*, pages 1749–1754, 2015.

[14] Salvador García, Julián Luengo, and Francisco Herrera. *Data Preprocessing in Data Mining*. Springer Publishing Company, Incorporated, 2015.

[15] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[16] E. K. Garcia, S. Feldman, M. R. Gupta, and S. Srivastava. Completely lazy learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(9):1274–1285, 2010.

[17] Chi Zhang, Feifei Li, and Jeffrey Jestes. Efficient parallel knn joins for large data in mapreduce. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 38–49, New York, NY, USA, 2012. ACM.

[18] Kai Sun, Hoon Kang, and Ho-Hyun Park. Tagging and classifying facial images in cloud environments based on kNN using mapreduce. *Optik - International Journal for Light and Electron Optics*, 126(21):3227 – 3233, 2015.

[19] J. Maillo, S. Ramírez, I. Triguero, and F. Herrera. kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data. *Knowledge-Based Systems*, 117:3–15, 2017.

[20] S. García, J. Derrac, J.R. Cano, and F. Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):417–435, 2012.

[21] I. Triguero, J. Derrac, S. García, and F. Herrera. A taxonomy and experimental study on prototype generation for nearest neighbor classification. *IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews*, 42(1):86–100, 2012.

[22] Isaac Triguero, Daniel Peralta, Jaume Bacardit, Salvador García, and Francisco Herrera. MRPR: A mapreduce solution for prototype reduction in big data classification. *Neurocomputing*, 150:331–345, 2015.

[23] Huan Liu and Hiroshi Motoda. *Computational methods of feature selection*. CRC Press, 2007.

[24] Daniel Peralta, Sara del Río, Sergio Ramírez-Gallego, Isaac Triguero, Jose M Benitez, and Francisco Herrera. Evolutionary feature selection for big data classification: A mapreduce approach. *Mathematical Problems in Engineering*, 2015, 2016.

[25] Julián Luengo, Salvador García, and Francisco Herrera. On the choice of the best imputation methods for missing values considering three groups of classification methods. *Knowledge and Information Systems*, 32(1):77–108, 2012.

[26] Gustavo E. A. P. A. Batista and Maria Carolina Monard. An analysis of four missing data treatment methods for supervised learning. *Applied Artificial Intelligence*, 17(5-6):519–533, 2003.

[27] José Ramón Cano, Francisco Herrera, and Manuel Lozano. Using evolutionary algorithms as instance selection for data reduction in KDD: an experimental study. *IEEE transactions on evolutionary computation*, 7(6):561–575, 2003.

[28] I. Triguero, J. Maillo, J. Luengo, S. García, and F. Herrera. From big data to smart data with the k-nearest neighbours algorithm. In *2016 IEEE International Conference on Smart Data*, pages 859–864, Dec 2016.

[29] M. Snir and S. Otto. *MPI-The Complete Reference: The MPI Core*. MIT Press, 1998.

[30] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 3rd edition, 2012.

[31] 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, Nov 2016.

[32] H. Fadili and C. Jouis. Towards an automatic analyze and standardization of unstructured data in the context of big and linked data. In *8th International Conference on Management of Digital EcoSystems, MEDES 2016*, pages 223–230, 2016.

[33] 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.

[34] 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.

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

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

[37] Benoît Frénay and Michel Verleysen. Classification in the presence of label noise: A survey. *IEEE Trans. Neural Netw. Learning Syst.*, 25(5):845–869, 2014.

[38] Sergio Ramírez-Gallego, Iago Lastra, David Martínez-Rego, Verónica Bolón-Canedo, José Manuel Benítez, Francisco Herrera, and Amparo Alonso-Betanzos. Fast-mRMR: Fast Minimum Redundancy Maximum Relevance algorithm for High-Dimensional big data. *International Journal of Intelligent Systems*, 32(2):134–152, 2017.

[39] 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.

[40] 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.

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

[42] Avnish Kumar Rastogi, Nitin Narang, and Zamir Ahmad Siddiqui. Imbalanced big data classification: A distributed implementation of smote. In *Proceedings of the Workshop Program of the 19th International Conference on Distributed Computing and Networking*, Workshops ICDCN ’18, pages 14:1–14:6, 2018.

[43] Alberto Fernández, Sara del Río, Nitesh V Chawla, and Francisco Herrera. An insight into imbalanced big data classification: outcomes and challenges. *Complex & Intelligent Systems*, 3(2):105–120, 2017.

[44] 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.

[45] Grard Biau and Luc Devroye. *Lectures on the Nearest Neighbor Method.* Springer, 2015.

[46] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 604–613, 1998.

[47] Shounak Datta, Debaleena Misra, and Swagatam Das. A feature weighted penalty based dissimilarity measure for k-nearest neighbor classification with missing features. *Pattern Recognition Letters*, 80:231–237, 2016.

[48] Peng-Cheng Zou, Jiandong Wang, Songcan Chen, and Haiyan Chen. Margin distribution explanation on metric learning for nearest neighbor classification. *Neurocomputing*, 177:168–178, February 2016.

[49] Zhibin Pan, Yidi Wang, and Weiping Ku. A new general nearest neighbor classification based on the mutual neighborhood information. *Knowledge-Based Systems*, 121:142–152, 2017.

[50] F. Angiulli. Fast nearest neighbor condensation for large data sets classification. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1450–1464, 2007.

[51] D.B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *11th International Conference on Machine Learning (ML'94)*, pages 293–301, 1994.

[52] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[53] S. García, J.R. Cano, and F. Herrera. A memetic algorithm for evolutionary prototype selection: A scaling up approach. *Pattern Recognition*, 41(8):2693–2709, 2008.

[54] Chin Liang Chang. Finding prototypes for nearest neighbor classifiers. *IEEE Transactions on Computers*, 100(11):1179–1184, 1974.

[55] James C Bezdek and Ludmila I Kuncheva. Nearest prototype classifier designs: An experimental study. *International journal of Intelligent systems*, 16(12):1445–1473, 2001.

[56] Isaac Triguero, Salvador García, and Francisco Herrera. IPADE: iterative prototype adjustment for nearest neighbor classification. *IEEE Transactions on Neural Networks*, 21(12):1984–1990, 2010.

[57] I. Iguyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[58] Amir Navot, Lavi Shpigelman, Naftali Tishby, and Eilon Vaadia. Nearest neighbor based feature selection for regression and its application to neural activity. In *Advances in Neural Information Processing Systems*, pages 996–1002, 2006.

[59] Igor Kononenko. Estimating attributes: Analysis and extensions of relief. In *Machine Learning: ECML-94*, pages 171–182, Berlin, Heidelberg, 1994.

[60] B. Xue, M. Zhang, W. N. Browne, and X. Yao. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation*, 20(4):606–626, Aug 2016.

[61] Joaquín Derrac, Salvador García, and Francisco Herrera. IFS-CoCo: Instance and feature selection based on cooperative coevolution with nearest neighbor rule. *Pattern Recognition*, 43(6):2082 – 2105, 2010.

[62] I. Triguero, S. García, and F. Herrera. Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification. *Pattern Recognition*, 44(4):901–916, 2011.

[63] Raul Jose Palma-Mendoza, Daniel Rodriguez, and Luis de-Marcos. Distributed reliefF-based feature selection in spark. *Knowledge and Information Systems*, Jan 2018.

[64] Álvar Arnaiz-González, Alejandro González-Rogel, José-Francisco Díez-Pastor, and Carlos López-Nozal. MR-DIS: democratic instance selection for big data by MapReduce. *Progress in Artificial Intelligence*, 6(3):211–219, Sep 2017.

[65] 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.

[66] Shi Zhong, Taghi M. Khoshgoftaar, and Naeem Seliya. Analyzing Software Measurement Data with Clustering Techniques. *IEEE Intelligent Systems*, 19(2):20–27, 2004.

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

[68] Mauricio A. Hernández and Salvatore J. Stolfo. Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2:9–37, 1998.

[69] Pedro J. García-Laencina, José-Luis Sancho-Gómez, and Aníbal R. Figueiras-Vidal. Pattern classification with missing data: A review. *Neural Computing and Applications*, 19(2):263–282, 2010.

[70] Roderick J A Little and Donald B Rubin. *Statistical analysis with missing data*, volume 333. John Wiley & Sons, 2014.

[71] Patrick Royston et al. Multiple imputation of missing values. *Stata journal*, 4(3):227–41, 2004.

[72] Roderick J. A. Little. A test of missing completely at random for multivariate data with missing values. *Journal of the American Statistical Association*, 83(404):1198–1202, 1988.

[73] 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, 2016.

[74] Diego García-Gil, Julián Luengo, Salvador García, and Francisco Herrera. Enabling smart data: Noise filtering in big data classification. *CoRR*, abs/1704.01770, 2017.

[75] J. S. Sánchez, R. Barandela, A. I. Marqués, R. Alejo, and J. Badenas. Analysis of new techniques to obtain quality training sets. *Pattern Recognition Letters*, 24(7):1015–1022, 2003.

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

[77] I. Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems and Man and Cybernetics*, 6(6):448–452, 1976.

[78] J.S. Sánchez, F. Pla, and F.J. Ferri. Prototype selection for the nearest neighbor rule through proximity graphs. *Pattern Recognition Letters*, 18:507–513, 1997.

[79] Tapio Schneider. Analysis of incomplete climate data: Estimation of mean values and covariance matrices and imputation of missing values. *Journal of climate*, 14(5):853–871, 2001.

[80] Hyunsoo Kim, Gene H Golub, and Haesun Park. Missing value estimation for DNA microarray gene expression data: local least squares imputation. *Bioinformatics*, 21(2):187–198, 2004.

[81] ECBDL14 dataset: Protein structure prediction and contact map for the ECBDL2014 big data competition, 2014.

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

[83] I. Triguero, S. Gonzalez, J.M. Moyano, S. García, J. Alcala-Fdez, J. Luengo, A. Fernández, M.J. del Jesus, L. Sánchez, and F. Herrera. Keel 3.0: An open source software for multi-stage analysis in data mining. *International Journal of Computational Intelligence Systems*, 10:1238–1249, 2017.

[84] P. E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 18:515–516, 1968.

[85] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1993.

[86] KQ Weinberger and LK Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, 2009.

[87] Bac Nguyen, Carlos Morell, and Bernard De Baets. Supervised distance metric learning through maximization of the jeffrey divergence. *Pattern Recognition*, 64:215 – 225, 2017.

[88] Dietrich Wettschereck, David W. Aha, and Takao Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, 11(1):273–314, Feb 1997.

[89] Joaquin Derrac, Salvador García, and Francisco Herrera. Fuzzy nearest neighbor algorithms: Taxonomy, experimental analysis and prospects. *Information Sciences*, 260:98 – 119, 2014.

[90] J. M. Keller, M. R. Gray, and J. A. Givens. A fuzzy k-nearest neighbor algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(4):580–585, July 1985.

[91] J. Maillo, J. Luengo, S. García, F. Herrera, and I. Triguero. Exact fuzzy k-nearest neighbor classification for big datasets. In *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6, July 2017.

[92] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, November 1998.

[93] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 459–468, Oct 2006.

[94] Elisa Bertino, C, Kian-Lee Tan, Beng Chin Ooi, Ron Sacks-Davis, Justin Zobel, and Boris Shidlovsky. *Indexing Techniques for Advanced Database Systems*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[95] Subhajit Dutta and Anil K. Ghosh. On some transformations of high dimension, low sample size data for nearest neighbor classification. *Machine Learning*, 102(1):57–83, Jan 2016.

[96] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, September 1977.

[97] Jeffrey K. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175 – 179, 1991.

[98] Ting Liu, Andrew W Moore, Ke Yang, and Alexander G Gray. An investigation of practical approximate nearest neighbor algorithms. In *Advances in neural information processing systems*, pages 825–832, 2005.

[99] Ting Liu, Charles J Rosenberg, and Henry A Rowley. Performing a parallel nearest-neighbor matching operation using a parallel hybrid spill tree, January 6 2009. US Patent 7,475,071.

[100] Julián Luengo, Seong-O Shim, Saleh Alshomrani, Abdulrahman Altalhi, and Francisco Herrera. Cncnos: Class noise cleaning by ensemble filtering and noise scoring. *Knowledge-Based Systems*, 140:27 – 49, 2018.

4 Redundancy and Complexity Metrics for Big Data Classification: Towards Smart Data

- J. Maillo, I. Triguero, F. Herrera. Redundancy and Complexity Metrics for Big Data Classification: Towards Smart Data.
  - Status: **Submitted**.
  - Impact Factor (JCR 2018): **2.541**
  - Subject Category: **Computer Science, Artificial Intelligence**
  - Rank: **0/133**
  - Quartile: **Q1**

Redundancy and Complexity Metrics for Big Data Classification: Towards Smart Data

Jesus Maillo
Department of Computer Science
and Artificial Intelligence
University of Granada, Granada, Spain 18071
`jesusmh@decsai.ugr.es`

Isaac Triguero
Computational Optimisation and Learning Lab (COL)
School of Computer Science, University of Nottingham
Jubilee Campus, Nottingham NG8 1BB, United Kingdom
`Isaac.Triguero@nottingham.ac.uk`

Francisco Herrera
Department of Computer Science and Artificial Intelligence
University of Granada, Granada, Spain 18071
Faculty of Computing and Information Technology
King Abdulaziz University (KAU) Jeddah, Saudi Arabia
`herrera@decsai.ugr.es`

Abstract

It is recognized the importance of knowing the descriptive properties of a dataset when tackling a data science problem. Having information about the redundancy, complexity and density of a problem allows us to make decisions as to which data preprocessing and machine learning techniques are most suitable. In classification problems, there are multiple metrics to describe the overlapping of the features between classes, class imbalances or separability, among others. However, these metrics may not scale up well when dealing with big datasets, or may not simply be sufficiently informative in this context. In this paper, we provide a package of metrics for big data classification problems. In particular, we propose two new big data metrics: Neighborhood Density and Decision Tree Progression, which study density and accuracy progression by discarding half of the samples. In addition, we enable a number of basic metrics to handle big data. The experimental study carried out in standard big data classification problems shows that our metrics can quickly characterize big datasets. We identified a clear redundancy of information in most datasets, so that, discarding randomly 75% of the samples does not drastically affect the accuracy of the classifiers used. Thus, the proposed big data metrics, which are available as a Spark-Package, provide a fast assessment of the shape of a classification dataset prior to applying big data preprocessing, toward smart data.

Keywords Big Data · Smart Data · Classification · Redundancy · Complexity · Apache Spark

1 Introduction

In many different applications, we are collecting large amounts of data with the purpose of obtaining useful insights through a Knowledge Discovery in Databases process [1]. Their nature is very diverse, with implications for society in all its fields, such as theoretical physics in studies carried out at CERN [2], implications for politics [3], new challenges posed in social media [4] or advances in medical applications [5], among others.

Despite the ease of finding/gathering large amounts of data in a multitude of fields, this data needs to be preprocessed to discard those samples that are disruptive, and select the data that provides quality information for machine learning. This process, included in the denominated Smart Data technologies [6], aims to obtain quality data [7] through the application of data preprocessing algorithms [8]. In [9], we discussed the use of the k Nearest Neighbors (kNN) algorithm [10] as a key technique capable of imputing missing values [11] and reducing redundant [12] and noisy data [13] to obtain quality data from big datasets.

The main assumption of most current research in big data is that having more data would enable better insights. However, having more data does not necessarily imply that we can obtain more relevant information, and may result in unnecessary computational cost. Smart data technologies alleviate this issue [9]. However, the application of very sophisticated big data preprocessing algorithms may also not be needed if, for example, we identify high levels of redundancy. With this hypothesis and the problem highlighted, we ask the following question:

- When is Big Data too much data for machine learning?

To appropriately answer this question, we need to know the characteristics of the dataset to be addressed before applying any big data preprocessing or machine learning algorithm. In that way, we may avoid running time-consuming techniques without knowing if they are necessary. To achieve this, there are metrics that mainly measure three aspects: complexity [14], which is defined as the difficulty in classifying unseen samples; redundancy [15] that refers to the existence of instances where the information they provide is already present in other instances; and density [16] which represents a high number of instances in relation to the domain of the problem.

These metrics are commonly used in the field of auto machine learning [17] as extracted features from a dataset, which help determine the best pipelines (i.e. combination of preprocessing and learning algorithms) for a new given dataset [18]. However, existing metrics were developed for standard problems [19], quality measures present problems of computational scalability in order to tackle big datasets. These problems come from their design, for example: density metrics based on the pruning of completely connected graphs [20], or complexity metrics based non-linearity of classifier based on sequential classification algorithms [21], both with very high computational complexity.

In this paper, we postulate that the big data literature is often neglecting the fact that there is redundancy in the data. Collect and store data for the sake of it may cause data storage and computational problems. Therefore, it is necessary to characterize a problem by means of complexity, redundancy and density metrics prior to applying big data preprocessing or machine learning algorithms.

We propose two new big data metrics to measure density and complexity, called Neighborhood Density (ND) and Decision Tree Progression (DTP) respectively, to detect the redundancy of information in big datasets and reduce their size when necessary, alleviating the issues mentioned above.

The main contributions of this paper are:

A) We proposed two new big data metrics:

- ND presents the proximity of samples by calculating the percentual difference of the Euclidean distance, which is calculated with all available data, and with the half of them randomly chosen.
- DTP measures complexity and redundancy by training two decision trees with the totality of the data, and discarding half of them randomly. The percentual difference of the accuracy obtained with each model is calculated to reflect the loss of information.

Moreover, we implement some of the best-known metrics in the literature [19] re-designed for execution in big datasets. An open source Spark-based [22] package has been developed that includes the two proposed metrics and a set of literature metrics, which is available on the spark-packages platform: `https://spark-packages.org/package/JMailloH/ComplexityMetrics`

B) Redundancy has been analized.An experimental study has been carried out composed of ND and DTP, as well as literature metrics adapted to the big data environment and three classification algorithms. In addition, a random data subsampling analyis has been carried out at different levels to investigate the effect of the sample size.

The remainder of this paper is organized as follows. Section 2 introduces state-of-the-art on scalable complexity metrics selected for experimental study. Then, Section 3 details the two proposed metrics and analyze their complexity. Section 4 and Section 5 describe the experimental setup and multiple analyses of results, respectively. Finally, Section 6 outlines the conclusions and future work.

2 Complexity measures

This section provides insights about the complexity metrics existing in the literature that have been selected to be developed in Spark. Thus, these metrics can be calculated over large datasets. Lorena et al. [19] perform an extensive review of existing metrics in the literature to study the complexity of problems.

For the definition of metrics, we consider a dataset T formed by n samples. Each sample (x, y) is described by $[x_1, ..., x_m]$ input variables, from now on features, and belonging to an output variable y_{n_c}, composed by n_c classes.

2.1 F1. Maximum Fisher's discriminant ratio

This metric measures the overlap between the features of the different classes of the problem. Specifically, it calculates the overlap of each feature separately, and takes the highest.

Orriols puig et al. [23] proposes how to calculate F1, but has the disadvantage of being for binary problems. Mollineda et al. [24] extends the proposal to multiclass problems, and for this reason we select that proposal to be implemented. F1 is calculated for each feature separately, and finally the most restrictive of all is returned:

$$F1 = \max_{i=1}^{m} r_{f_i} \tag{1}$$

r_{f_i} is computed as defined Equation 2.

$$r_{f_i} = \frac{\sum_{j=1}^{n_c} n_{c_j} (\mu_{c_j}^{f_i} - \mu^{f_i})^2}{\sum_{j=1}^{n_c} \sum_{l=1}^{n_{c_j}} (x_{li}^{j} - \mu_{c_j}^{f_i})^2} \tag{2}$$

Where $\mu_{c_j}^{f_i}$ is the average of the i-th feature of the samples of the class j, μ^{f_i} is the average of the i-th feature of all instances and x_{li}^{j} is the specific value of the i-th feature for a particular sample x.

Its computational complexity is $O(m \cdot n)$. The metric domain is $[0, +\infty]$, and it is inversely proportional, meaning that if the resulting value is high, the complexity of the problem will be low.

2.2 F2. Volume of overlapping region

The F2 metric calculates the overlap between the samples of the different classes. In this case, it considers the domain (maximum and minimum values) of all features. For this reason, it is called "Volume" of overlapping region. Commins [25] proposes the metric defined in Equation 3.

$$F2 = \prod_{i}^{m} \frac{\max\{0, minmax(f_i) - maxmin(f_i)\}}{maxmax(f_i) - minmin(f_i)} \tag{3}$$

Let f_i and c_j be the i-th feature and the j-th class respectively, where:

- $minmax(f_i) = min(max(f_i^{c_1}), max(f_i^{c_2}) \dots max(f_i^{c_j}))$
- $minmin(f_i) = min(min(f_i^{c_1}), min(f_i^{c_2}) \dots min(f_i^{c_j}))$
- $maxmax(f_i) = max(max(f_i^{c_1}), max(f_i^{c_2}) \dots max(f_i^{c_j}))$
- $maxmin(f_i) = max(min(f_i^{c_1}), min(f_i^{c_2}) \dots min(f_i^{c_j}))$

Its computational complexity is $O(m \cdot n \cdot n_c)$ and its domain is $[0, 1]$. It is directly proportional, therefore, a value 1 in the metric means a high complexity.

2.3 F3. Maximum Individual Feature Efficiency

The basis of this complexity metrics is to account for whether classes are linearly separable by a single feature. To do this, it calculates the ratio of examples that are not in the overlap area and the total number of examples:

$$F3 = \max_{i=1}^{m} \frac{n - n_o(f_i)}{n} \tag{4}$$

Where $n_o(f_i)$ is the number of samples found in the overlap area, whose membership is defined by Equation 5.

$$n_o(f_i) = \sum_{j=1}^{n} I(x_{ji} > maxmin(f_i) \wedge x_{ji} < minmax(f_i)) \tag{5}$$

I returns value 1 if the condition is satisfied, and 0 if the condition is unsatisfied. Thus, it counts the number of samples in the overlap area.

Its computational complexity is $O(m \cdot n \cdot n_c)$, with a domain of $[0, 1]$. The metric is inversely proportional to the complexity.

2.4 F4. Collective Feature Efficiency

A natural extension of F3 is to count in the same way but considering all features [23]. Concretely apply the metric F3 iteratively following the next procedure:

1. F3 is computed to determine which feature is the most discriminatory.
2. The instances that are outside the overlap area corresponding to the feature selected in step 1 are discarded.
3. The feature selected in step 1 is removed, and the procedure is repeated until all features are considered.

It is formally described in Equation 6.

$$F4 = \frac{n - n_o(f_{max}(T_l))}{n} \tag{6}$$

Considering that the set T_i is subject to the changes described in the iterative procedure, $f_{max}(T_i)$ is:

$$f_{max}(T_i) = \{f_j | \max_{j=1}^{m} n - n_o(f_j))\}T_i \tag{7}$$

Its computational complexity is higher than F3, because it iterates on all features $O(m^2 \cdot n \cdot n_c)$. In the same way as F3, complexity is inversely proportional to the value of the metric, and its domain is $[0, 1]$.

2.5 C1. Entropy of class proportions

Lorena et al. [26] proposes an entropy-based metric to measure the imbalance between classes [27]. The mathematical expression is presented in Equation 8.

$$C1 = -\frac{1}{\log(n_c)} \sum_{i=1}^{n_c} p_i \log(p_i) \tag{8}$$

Where p_i represents the proportion of instances of the class i ($p_i = n_i/n$).

It has a computational complexity of $O(n)$. The metric domain is $[0, 1]$ and is inversely proportional to the complexity. A value of 1 indicates a perfect balance between the number of instances of the different classes.

2.6 C2. Imbalance ratio

It is the most extended metric for class imbalance problems, specifically, the multiclass version proposed at [28]. Its computation is presented in Equation 9.

$$C2 = \frac{n_c - 1}{n_c} \sum_{i=1}^{n_c} \frac{n_i}{n - n_i} \tag{9}$$

Its computational complexity is $O(n)$, and the metric domain is $[1, +\infty]$. The relationship between the metric and the complexity of the problem is directly proportional, so a value of 1 indicates a perfect balance between classes.

3 Big Data Metrics: Neighborhood Density and Decision Tree Progression

This section presents the two proposed metrics specifically designed to deal with big datasets. Neighborhood Density (Section 3.1) takes as its basis the distance between samples and how discarding half of the samples affects it. Decision Tree Progression (Section 3.2) shows the progression of the accuracy obtained by Decision Tree with all instances and dropping half of them. Finally, it summarizes the implemented metrics (Section 3.3) that compose the open source package *ComplexityMetrics*.

3.1 ND. Neighborhood Density

In this subsection, we present an original proposal for estimating density loss in a dataset based on neighborhood. For the design of the ND metric, we based on the hypothesis that the distance ratio represents the density of the dataset. However, simply the distance between the samples is not enough information, as it varies depending on the dataset without implying a higher or lower density. In order to provide valuable information, we will calculate the variation of the mean distance between the samples of a dataset, counting the whole dataset, and reducing it by half.

To do this, the mean distance between all instances is calculated, considering the nearest neighbor. Afterwards, half of the samples are randomly drawn, and the procedure is repeated to obtain the mean distance again. The percentage increase of the distance will be the value that indicates the density.

Figure 1 and Algorithm 1 describe the workflow for calculating the metric Neighborhood Density (ND), which is explained below:

1. We start from the complete dataset, and split it into 2, leaving 90% of the data in a set that we will call neighborhood and the remaining 10% in one that will be named validation.
2. The average distance of all instances of the $validation$ set is calculated, along to the $neighborhood$ set. The distance is calculated as performed by the nearest neighbors algorithm (1NN). The average distance obtained will then be named $\overline{d}$.
3. It takes half of the instances of the $neighborhood$ set and calculates again the average distance of all the instances from $validation$ set. The average distance obtained will be named $\overline{d_s}$.
4. Once calculated $\overline{d}$ and $\overline{d_s}$, the result of the metric will be the percentage difference of the distances. Equation 10 presents the mathematical expression performed.

$$ND = \frac{\overline{d} - \overline{d_s}}{\overline{d}} \cdot 100 \tag{10}$$

Algorithm 1 Neighborhood Density

```
Require: $data$
1: $neighborhood, validation \leftarrow$ randomSplit($data$,90%,10%)
2: $neighborhood_{sub} \leftarrow$ sample(50%)
3:
4: $\overline{d} \leftarrow$ averageDistance($neighborhood, validation$)
5: $\overline{d_s} \leftarrow$ averageDistance($neighborhood_{sub}, validation$)
6: return $(\overline{d} - \overline{d_s}/\overline{d}) \cdot 100$
```

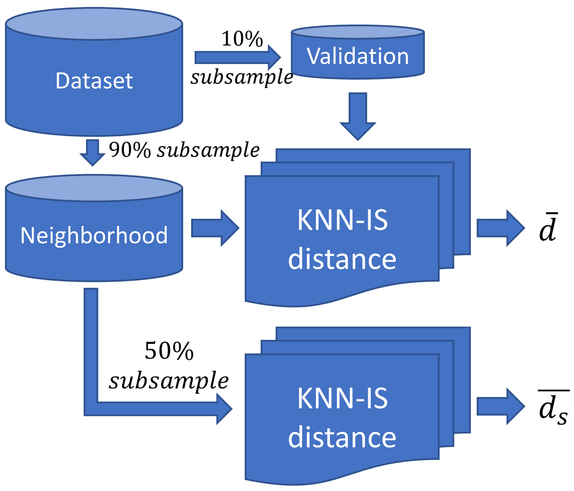


Figure 1: Neighborhood Density Workflow

Going deeper at the technical level, the development of the ND metric code uses the kNN-IS algorithm [29] to calculate 1NN, KNN-IS algorithm gets the exact nearest neighbors, implemented on the Spark platform.

3.2 DTP. Decision Tree Progression

In this section, we detail the second original proposal for estimating accuracy loss in a dataset with the decision tree algorithm. In this occasion, for the design of the DTP metric, we take the accuracy of the decision tree classifier as a measure of complexity. However, accuracy by itself does not enable us to know how complexity evolves with respect to the number of instances of the dataset. To obtain valuable information, we calculate the accuracy loss by excluding half of the instances in the training.

To do this, a small sample is taken to be used as a test set and then a DT is trained with the complete set and half of the data. Accuracy is calculated with the two trained models by classifying the same test set. The metric consists of the percentage difference between the accuracy. If it returns a negative value, it implies that you have obtained a better result with the model trained with half of the data.

The metric workflow is presented in Figure 2 and the Algorithm 2, which is composed of the following steps:

1. We start from the complete dataset, and split it into 2, leaving 90% of the data in a set that we will call $training$ and the remaining 10% in one that will receive the name of $test$.
2. Afterwards, the DT is trained with the training set, and the test set is classified, calculating the accuracy (acc).
3. One-half of the instances of the $training$ set are discarded, and will be called $training_{sub}$. We train a new DT with $training_{sub}$ set, and classify the $test$ set, keeping the accuracy (acc_s).
4. Once calculated acc and acc_s, we calculate the accuracy percentage difference, following the Equation 11.

$$DTP = \frac{acc - acc_s}{acc} \cdot 100 \tag{11}$$

Algorithm 2 Decision Tree Progression

```
Require: $data$
1: $training, test \leftarrow$ randomSplit($data$,90%,10%)
2: $training_{sub} \leftarrow$ sample(50%)
3:
4: $acc \leftarrow$ accuracyDT($training, test$)
5: $acc_s \leftarrow$ accuracyDT($training_{sub}, test$)
6: return $(acc - acc_s/acc) \cdot 100$
```

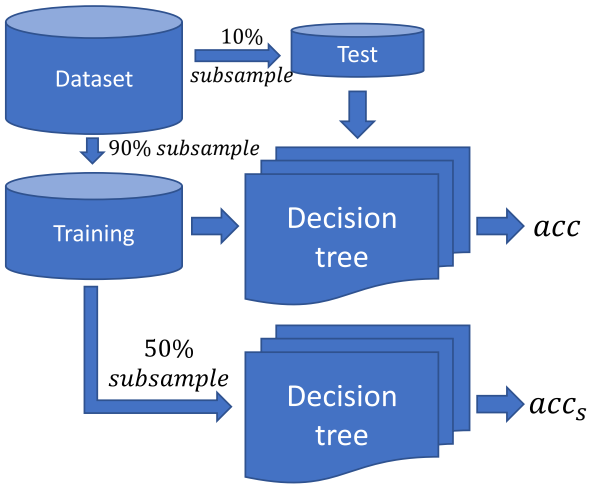


Figure 2: Decision Tree Progression Workflow

The DT code used is the one available in the MLLib library. Its parameters are: Gini as impurity measure. Maximum depth equal to 20 and maximum number of samples per bins set to 32.

3.3 Software package: Complexity Metrics

All metrics presented in this paper are as a free software package *ComplexityMetrics* hosted in the spark-packages [30] library available at: `https://spark-packages.org/package/JMailloH/ComplexityMetrics`.

The metrics have been developed under the Map Reduce paradigm [31] providing them with scalability to address large datasets. Specifically, the Apache Spark framework [22] has been selected due to its popularity and results against other distributed proposals [32]. In particular, the literature metrics have been implemented using the official machine learning library, MLlib [33], specifically with the Statistics class. The Statistics class calculates in a very efficient way the maximum, minimum and average values of each feature of the complete dataset. With these statistical values, the mathematical expressions for each metric described in the Section 2 are computed, obtaining the overlap by filtering the instances when it is necessary. The technical details of the original proposals have already been described in the Sections 3.1 and 3.2.

Table 3 summarizes the abbreviation and name of each metric, indicating also the minimum and maximum value they can take, whether the complexity is directly or inversely proportional ($\propto$ and $1/\propto$, respectively) to the value of the metric (Column Proportionality), and the computational complexity.

Table 1: Summary of the metrics

Abbreviation	Namel	Minimum	Maximum	Proportionality	Computational Complexity
F1	Maximum Fisher's discriminant ratio	0	$+\infty$	$1/\propto$	$O(m \cdot n)$
F2	Volume of overlapping region	0	1	$\propto$	$O(m \cdot n \cdot n_c)$
F3	Maximum individual feature efficiency	0	1	$1/\propto$	$O(m \cdot n \cdot n_c)$
F4	Collective feature efficiency	0	1	$1/\propto$	$O(m^2 \cdot n \cdot n_c)$
C1	Entropy of class portions	0	1	$1/\propto$	$O(n)$
C2	Imbalance ratio	1	$+\infty$	$\propto$	$O(n)$
ND	Neighborhood Density	$-\infty$	$+\infty$	$\propto$	$O(n^2 \cdot m + \frac{n^2 \cdot m}{2})$
DTP	Decision Tree Progression	$-\infty$	$+\infty$	$\propto$	$O(n \cdot m \cdot \log(n) + \frac{n \cdot m \cdot \log(n)}{2})$

The computational complexity indicated is the sequential execution one. All implementations have been adapted to be executed in a distributed way using Spark's primitive operations, providing high scalability to all of them.

4 Experimental set-up

This section presents the details of the experimental set-up. It describes the datasets used (Section 4.2), the classification algorithms used and their parameters (Section 4.3), and finally, the hardware and software characteristics under which the experimentation has been carried out (Section 4.1).

4.1 Software and hardware specification

The experiments have been executed in a cluster dedicated to distributed computing. The cluster is composed of a master node, and 14 compute nodes. Regarding software configuration: Spark (version 2.2.1), Scala (version 2.11.6) and HDFS (Version 2.6.0-cdh5.8.0) on the CentOS operating system (version 6.5).

The hardware performance of each machine is as follows: two Intel Xeon CPU E5-2620 processors (2 GHz), with 12 threads each (6 cores), 64 GB main memory and 15 MB cache memory. The connection between the machines is Infiniband at 40 Gb/s speed. With this configuration, the cluster can host a total of 256 map operations in parallel.

4.2 Datasets

The experimental study consists of 6 standard big classification datasets extracted from the UCI repository [34]. They have been selected for their high relevance in previous experimental studies in the field of big data classification. Table 2 summarizes the number of samples, features, and classes for each dataset.

Table 2: Description of the datasets

Dataset	#Samples	#Features	#Classes
Higgs	11,000,000	28	2
Ht_sensor	928,991	11	3
Skin	245,057	3	2
Susy	5,000,000	18	2
Watch_acc	3,540,962	20	7
Watch_gyr	3,205,431	20	7

For the experimentation carried out, a 5 fold cross-validation scheme was followed, with 80% dedicated to training and 20% to testing. In addition, the experimentation has the particularity of making versions of each dataset by random subsampling, this technique is typically known as random undersampling (RUS) [35]. RUS is used in problems of class imbalance, to reduce the number of samples of the majority class and facilitate the learning of the classification algorithm used later. However, our objective is different, we want to know if we need all the samples or if they contain redundant information. Thus, following the cross validation scheme, on the one hand, RUS is applied to the training partition maintaining the same proportion of classes. On the other hand, RUS is not applied to the test partition, allowing to compare the accuracy results between the different classifiers and different sub-sampling levels performed. Table 3 shows the number of instances in the test and train partitions for each applied subsampling level.

Table 3: Instances for each dataset version

Dataset	#Instances	#Instances Training			
	Test	100%	75%	50%	25%
Higgs	2,200,000	8,800,000	6,600,000	4,400,000	2,200,000
Ht_sensor	185,798	743,193	557,395	371,596	185,798
Skin	49,011	196,046	147,034	98,023	49,011
Susy	1,000,000	4,000,000	3,000,000	2,000,000	1,000,000
Watch_acc	708,192	2,832,770	2,124,577	1,416,385	708,192
Watch_gyr	641,086	2,564,345	1,923,259	1,282,172	641,086

4.3 Classifiers and parameters

All metrics described in Section 2 have been used for experimentation. In addition, in order to cover a larger behavior in the experimental study, we have used three classification algorithms with different characteristics. These three algorithms are developed for Big Data problems, and represent three families of algorithms: based on instances or similarity, entropy and weight optimization. The algorithms used and their parameters are listed below:

- Local Hybrid Spill tree Fuzzy k Nearest Neighbors (LHS-FkNN) [36][1]: This algorithm is based on similarity, namely the Euclidean distance. The parameter used is k = 7, both in the class membership degree stage and the classification stage.
- Decision Tree (DT)[2] : This classifier is based on entropy and information gain. In the experimentation carried out, gini impurity was used, with a maximum depth of 20 and a maximum number of samples per leaf equal to 32.
- Multilayer Perceptron (MLP)[3] : classifier based on weight adjustment, is a type of artificial feed-forward neural network. For this experiment, we have used 2 hidden layers of 10 and 5 neurons, respectively. With a block size of 1000 and the maximum number of iterations equal to 500.

In Map Reduce-like implementations, it is also important to know the number of map tasks used. In all cases, 256 map operations have been used, which coincide with the maximum available in the cluster.

As we are dealing with standard classification problems, the accuracy metric was used to measure the quality of the results of the three classification algorithms used. The accuracy is calculated by dividing the number of well-classified samples by the number of total samples.

[1] https://spark-packages.org/package/saurfang/spark-knn
[2] https://spark.apache.org/docs/2.2.1/ml-classification-regression.html#decision-tree-classifier
[3] https://spark.apache.org/docs/2.2.1/ml-classification-regression.html#multilayer-perceptron-classifier

5 Analysis of results

In this section, we study the results obtained by the classification algorithms and the metrics developed (Section 5.1), its implications with data redundancy (Section 5.2) and the scalability through the runtime (Section 5.3).

5.1 Metrics and accuracy analysis

The study is designed to analyze the importance of the quality and quantity of the data available in a big data problem. Specifically, a sub-sampling study is performed at 75%, 50% and 25% to analyze whether a large amount of available data is necessary, or the dataset contains redundant information.

Table 4 show for the three classifiers used and each one of the metrics, the value obtained with the complete set (100%), and with the subsamples made, keeping 75%, 50% and 25% of the samples.

According to the results obtained, we can present the following conclusions:

- Focusing on the ND metric, there is an incremental progression from 100% of the samples as they are discarded in blocks of 25%. This shows how dropping instances also reduces the density of the dataset. Reducing the density in the dataset leads to a lack of representation in the problem, and consequently, to an increase in its complexity. However, if we compare the density obtained with the complete dataset, and the density with 25%, the difference presented is very small. This shows us that we can discard instances without drastically affecting the density obtained. To ensure this behavior, we can see the slight decrease in accuracy in the classifiers used, even slightly increasing with MLP.
- If we consider the DTP metric, it always keeps under 1 except for the Higgs dataset, up to 3. These low values represent the low loss of accuracy involved in discarding half of the dataset while training the DT classifier. In fact, if we compare DTP with 100% versus 25%, the differences are minimal. This information shows us that by discarding 75% of the instances, there is a minimal difference in the percentage loss of accuracy with respect to having all the instances.
- The accuracy of the classification algorithms does not drastically change even when 75% of the samples are drop randomly, which shows a clear redundancy of information. Going deeper into the analysis, we see how LHS-FkNN and DT are affected more by density loss. In the case of LHS-FkNN it is because it bases its learning on similarity, specifically on the Euclidean distance, thus defining the boundaries between classes. DT bases its learning on entropy, and specifically on the value taken by each node of the tree when deciding which class it belongs to. However, MLP learns by adjusting the weights of each neuron. For this reason, accuracy is maintained at similar values, improving slightly its results if we compare having 100% of the samples versus taking 25%.
- The F1 metric remains stable despite discarding instances. This shows how discarding instances does not affect the complexity of the problem. In addition, if we support the results of F1 with the accuracy obtained, it consolidates the existence of redundancy, and how discarding instances does not significantly harm the classifiers, improving the results for the MLP algorithm.
- C1 and C2 metrics, related to the problem of class imbalance, measure the entropy of classes and the ratio of imbalance respectively. Both show the almost perfect balance of all the datasets, except Skin, where C2 indicates us that there are double as many instances of one class with respect to the other. These metrics alone do not provide all the information desired to address a big data problem, and therefore require new metrics specific to large datasets. Joining several metrics gets useful

Table 4: Progression of results with each subsampling

Higgs dataset

Metric	100%	75%	50%	25%
LHS-FkNN	0.6200	0.6178	0.6153	0.6076
DT	0.6995	0.6969	0.6927	0.6830
MLP	0.6738	0.6728	0.6730	0.6756
ND	3.5079	3.8493	4.2407	4.6521
DTP	3.0444	3.1627	3.3830	3.1459
F1	0.0112	0.0112	0.0111	0.0112
F2	0.0000	0.0000	0.0000	0.0000
F3	0.9411	0.9411	0.9411	0.9411
F4	0.5002	0.5002	0.5001	0.4999
C1	0.9974	0.9974	0.9974	0.9975
C2	1.0071	1.0071	1.0071	1.0070

Ht_sensor dataset

Metric	100%	75%	50%	25%
LHS-FkNN	0.9999	0.9998	0.9997	0.9996
DT	0.9997	0.9997	0.9997	0.9992
MLP	0.7268	0.7346	0.7279	0.7308
ND	48.0838	54.4227	59.5184	61.9404
DTP	0.0180	0.0261	0.0396	0.1009
F1	0.0228	0.0227	0.0226	0.0228
F2	0.0000	0.0000	0.0000	0.0000
F3	0.4036	0.4034	0.4036	0.4038
F4	0.4036	0.4034	0.4036	0.4038
C1	0.9961	0.9962	0.9961	0.9962
C2	1.0065	1.0064	1.0065	1.0063

Skin dataset

Metric	100%	75%	50%	25%
LHS-FkNN	0.9932	0.9929	0.9929	0.9924
DT	0.9987	0.9986	0.9984	0.9982
MLP	0.9921	0.9982	0.9912	0.9942
ND	6.9607	7.5177	9.2297	11.4003
DTP	0.0327	0.0367	0.0401	0.0396
F1	0.3813	0.3812	0.3828	0.3845
F2	0.3091	0.3068	0.3026	0.2890
F3	0.3694	0.3710	0.3718	0.3732
F4	0.1094	0.1097	0.1112	0.1147
C1	0.7367	0.7369	0.7384	0.7396
C2	2.0402	2.0395	2.0310	2.0245

Susy dataset

Metric	100%	75%	50%	25%
LHS-FkNN	0.7476	0.7448	0.7410	0.7343
DT	0.7771	0.7736	0.7689	0.7601
MLP	0.7992	0.7992	0.7991	0.7990
ND	8.9968	9.0249	9.0141	9.0726
DTP	0.9894	1.1731	1.1063	1.2900
F1	0.1091	0.1092	0.1094	0.1095
F2	0.0000	0.0000	0.0000	0.0000
F3	0.5542	0.5557	0.5557	0.5596
F4	0.1000	0.1000	0.3001	0.3000
C1	0.9948	0.9948	0.9948	0.9949
C2	1.0145	1.0144	1.0144	1.0142

Watch_acc dataset

Metric	100%	75%	50%	25%
LHS-FkNN	0.9528	0.9485	0.9423	0.9290
DT	0.9113	0.9118	0.9080	0.9036
MLP	0.6978	0.7016	0.6957	0.7047
ND	57.2443	61.0654	70.8231	91.8570
DTP	0.2299	0.5808	0.3975	0.6139
F1	0.0558	0.0558	0.0558	0.0556
F2	0.0083	0.0080	0.0075	0.0064
F3	1.0000	1.0000	1.0000	1.0000
F4	1.0000	1.0000	1.0000	1.0000
C1	0.9958	0.9958	0.9958	0.9957
C2	1.0033	1.0033	1.0033	1.0034

Watch_gyr dataset

Metric	100%	75%	50%	25%
LHS-FkNN	0.9771	0.9741	0.9698	0.9605
DT	0.9006	0.9005	0.8973	0.8874
MLP	0.6780	0.6809	0.6950	0.7026
ND	57.4098	69.5357	81.5532	91.4943
DTP	0.6598	0.7048	0.9522	0.6138
F1	0.0867	0.0867	0.0867	0.0867
F2	0.0007	0.0007	0.0007	0.0008
F3	1.0000	1.0000	1.0000	1.0000
F4	1.0000	1.0000	1.0000	1.0000
C1	0.9985	0.9985	0.9985	0.9985
C2	1.0011	1.0011	1.0011	1.0011

information. An example would be the following: we have a dataset with C2 greater than 1, with DTP and ND with low values. This presents a high density and redundancy of information, with a moderate complexity. Thus, it would be more appropriate to apply sub-sampling techniques (such as instance selection or random undersampling) to reduce the size of the dataset as opposed to applying over-sampling techniques (such as prototype generation or random oversampling).

- Finally, to highlight a weakness detected in the metrics F2, F3 and F4, that belonging to the state-of-the-art in non-big data classification problems. The information they provide is contrary to that reflected by the accuracy reported by the classifiers, generating interest and relevance to the proposed metrics.

5.2 Redundancy analysis

Once all metrics have been analyzed, we are ready to answer the question raised:

When is Big Data too much data for machine learning?

Much data is not necessary, in the datasets used, based mainly on three aspects that occur when 75% of the instances are randomly dropped:

- First, DTP shows how complexity remains very low either with the complete set or after discarding instances.
- Second, ND shows a slight increase despite gradually discarding 25% of the instances, if the density of the datasets were low, this increase should be more abrupt and the metric values should be higher.
- Third, accuracy does not suffer a high loss for LHS-FkNN and DT, increasing slightly for MLP.

5.3 Scalability analysis

Below we present the runtime results of classifiers and metrics, with the aim of analyzing the scalability of the models and the influence of the number of samples. Figures 3 and 4 plot the runtime for literature metrics and our proposals, respectively, showing for each of them the 4 sub-sampling levels.

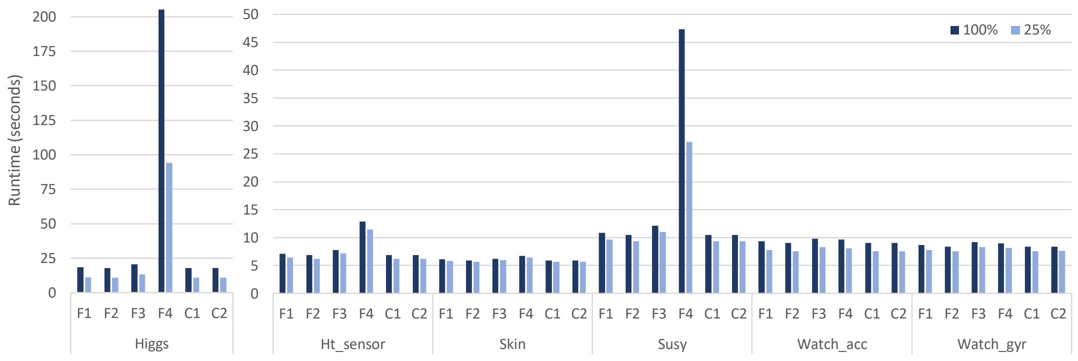
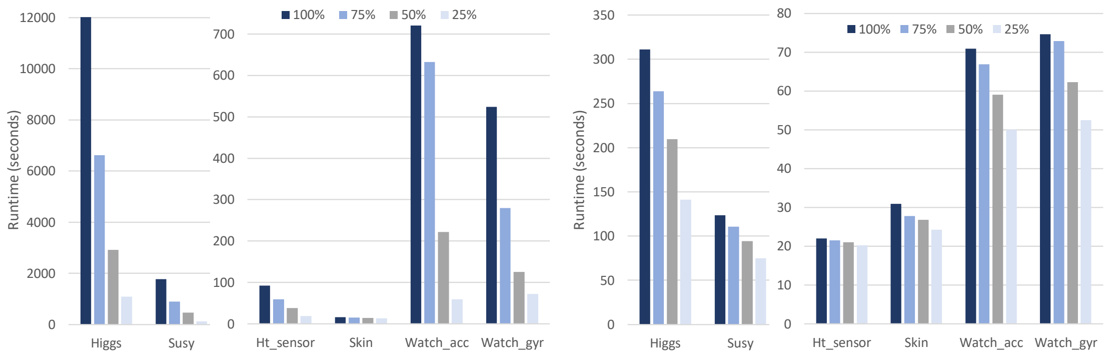


Figure 3: Runtime of literature metrics



(a) ND: Neighborhood Density (b) DTP: Decision Tree Progression

Figure 4: Runtime of proposed metrics

According to these runtimes, we extract the following analysis:

- The metrics in the literature have very fast runtimes, reaching a maximum of approximately 200 seconds for the Higgs dataset. In addition, the difference between the runtime of 100% of the data and 25% is not very high, which shows an excelent scalability of the metrics.
- In relation to the proposed metrics, ND obtains higher runtimes than the other metrics. In addition, it increases considerably if we compare 25% against 100%. This shows how the number of instances affects runtime. DTP is faster than ND and is more robust in scalability, as it is less affected by the number of instances. It is very important to remember the results obtained in the previous section, where it is shown that ND and DTP are the metrics that provide best information to the problem. Thus, obtaining the values of the proposed metrics allows us to know if we are facing problems where we can discard instances and keep the results very close.

After analyzing the scalability of the metrics, it is necessary to analyze the impact of the instance reduction in the classifiers. For this purpose, Figure 5 shows the runtime of the three classifiers with the 6 datasets, for their full version (100%) and maximum subsample applied (25%).

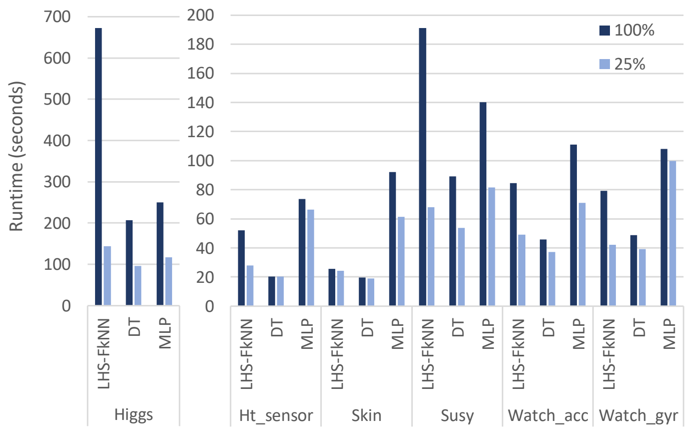


Figure 5: Runtime of classifiers

As expected, all the algorithms show a great reduction in runtime. LHS-FkNN and MLP achieve the greatest reduction in runtime. The reason is the LHS-FkNN algorithm is an instance-based method and reducing the number of instances decreases the number of comparisons to be made at the classification stage. On the other hand, MLP is based on weighting through an iterative process, for this reason, a high number of instances is affected in complexity by the number of iterations performed to train the model. DT remains more stable, slightly affected by the number of instances due to the design of the algorithm to train the tree in a distributed way.

The most relevant analysis that can be extracted involves the runtime. The time spent in obtaining the metrics can lead us to the conclusion of reducing the dataset by half, or keeping only 25% without significantly affecting the quality of the classifier. In addition, it allows us to perform more experiments to determine which is the algorithm that learns most about our problem and optimize the parameters of the classifiers. For example, we can see a realistic scenario: we find a problem where the ND and DTP metrics obtain low values, and in addition the C1 and C2 metrics show us a class imbalance problem. We can apply random undersampling, to produce a balance between classes and improve the quality of the results. Moreover, by reducing the size of the dataset, we can spend more time on finding a better solution to the problem, such as using preprocessing techniques to filter noisy instances or optimize the parameters of the classifier.

6 Conclusions and further work

In this paper, two metrics have been proposed to study the complexity and density in big data problems: ND and DTP study density and accuracy progression by discarding half of the samples randomly. In addition, some basic metrics have been adapted from the literature to handle big dataset. The design based on Spark allows us to characterize large datasets in a short period of time, obtaining valuable information. The developed metrics are available in the open source repository Spark-packages called *ComplexityMetrics* at: `https://spark-packages.org/package/JMailloH/ComplexityMetrics`

According to the study carried out through the proposed metrics, it is common for big datasets to show redundancy information in their samples. This high redundancy allows us to reduce the size to 25% of the samples without drastically affecting the accuracy obtained by the classifiers, achieving a significant faster runtimes. This shows that the number of instances in big datasets used is more than necessary, and highlights the need to prioritize preprocessing techniques to obtain smart data.

As a final conclusion, we have to emphasize the fact of redundancy in many big data classification problems, where with a much smaller set, a small quality dataset, we can have similar or better results. Here the challenge is in obtaining smart data with the minimum necessary size.

As future work we believe that the proposed metrics have a great potential to be integrated in the area of auto machine learning techniques [17] in the big data context. A good starting point would be to design a technique that allows us to determine the necessary size to tackle a big data classification problem, reducing the number of instances significantly without affecting the results obtained, toward a reduced smart data.

Acknowledgments

This contribution has been supported by the Spanish National Research Project TIN2017-89517-P. J. Maillo hold a FPU scholarship from the Spanish Ministry of Education.

References

[1] Tu Bao Ho. Knowledge discovery. In *Knowledge Science*, pages 70–93. CRC Press, 2016.

[2] Peter V Coveney, Edward R Dougherty, and Roger R Highfield. Big data need big theory too. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2080):20160153, 2016.

[3] Roberto J Gonzalez. Hacking the citizenry?: Personality profiling, big data and the election of donald trump. *Anthropology Today*, 33(3):9–12, 2017.

[4] Zeynep Tufekci. Big questions for social media big data: Representativeness, validity and other methodological pitfalls. In *Eighth International AAAI Conference on Weblogs and Social Media*, 2014.

[5] Travis B Murdoch and Allan S Detsky. The inevitable application of big data to health care. *Jama*, 309(13):1351–1352, 2013.

[6] Jo Arao Ramos, John Baxter Rollins, and David Giddens Wilhite. Smart data caching using data mining, March 22 2011. US Patent 7,912,812.

[7] Danilo Ardagna, Cinzia Cappiello, Walter Samá, and Monica Vitali. Context-aware data quality assessment for big data. *Future Generation Computer Systems*, 89:548–562, 2018.

[8] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Springer, 2015.

[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] Thomas M Cover, Peter Hart, et al. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[11] Julián Luengo, Salvador García, and Francisco Herrera. On the choice of the best imputation methods for missing values considering three groups of classification methods. *Knowledge and Information Systems*, 32(1):77–108, 2012.

[12] Muhammad Habib ur Rehman, Chee Sun Liew, Assad Abbas, Prem Prakash Jayaraman, Teh Ying Wah, and Samee U Khan. Big data reduction methods: a survey. *Data Science and Engineering*, 1(4):265–284, 2016.

[13] Diego García-Gil, Francisco Luque-Sánchez, Julián Luengo, Salvador García, and Francisco 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.

[14] L. P. F. Garcia, A. C. Lorena, M. C. P. de Souto, and T. K. Ho. Classifier recommendation using data complexity measures. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 874–879, Aug 2018.

[15] Ion Muslea, Steven Minton, and Craig A Knoblock. Selective sampling with redundant views. In *AAAI/IAAI*, pages 621–626, 2000.

[16] Masashi Sugiyama, Taiji Suzuki, and Takafumi Kanamori. *Density ratio estimation in machine learning*. Cambridge University Press, 2012.

[17] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Machine Learning*. Springer, 2019.

[18] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. *Auto-sklearn: Efficient and Robust Automated Machine Learning*, pages 113–134. Springer International Publishing, Cham, 2019.

[19] Ana C Lorena, Luís PF Garcia, Jens Lehmann, Marcilio CP Souto, and Tin Kam Ho. How complex is your classification problem?: A survey on measuring classification complexity. *ACM Computing Surveys (CSUR)*, 52(5):107, 2019.

[20] Luís PF Garcia, André CPLF de Carvalho, and Ana C Lorena. Effect of label noise in the complexity of classification problems. *Neurocomputing*, 160:108–119, 2015.

[21] Aarnoud Hoekstra and Robert PW Duin. On the nonlinearity of pattern classifiers. In *Proceedings of 13th International Conference on Pattern Recognition*, volume 4, pages 271–275. IEEE, 1996.

[22] 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 *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pages 15–28, 2012.

[23] Albert Orriols-Puig, Núria Macia, and Tin Kam Ho. Documentation for the data complexity library in c++. *Universitat Ramon Llull, La Salle*, 2010.

[24] Ramón A Mollineda, J Salvador Sánchez, and José M Sotoca. Data characterization for effective prototype selection. In *Iberian Conference on Pattern Recognition and Image Analysis*, pages 27–34. Springer, 2005.

[25] Lisa Cummins. Combining and choosing case base maintenance algorithms. *PhD Thesis, University College Cork*, 2013.

[26] Ana C Lorena, Ivan G Costa, Newton Spolaôr, and Marcilio CP De Souto. Analysis of complexity indices for classification problems: Cancer gene expression data. *Neurocomputing*, 75(1):33–42, 2012.

[27] Nitesh V. Chawla. *Data Mining for Imbalanced Datasets: An Overview*, pages 875–886. Springer US, Boston, MA, 2010.

[28] Ajay Kumar Tanwani and Muddassar Farooq. Classification potential vs. classification accuracy: a comprehensive study of evolutionary algorithms with biomedical datasets. In *Learning Classifier Systems*, pages 127–144. Springer, 2009.

[29] Jesus Maillo, 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.

[30] Spark packages contributors. Spark packages: A community index of third-party packages for apache spark, 2020. `https://spark-packages.org/`, [Online; accessed 15 January 2020].

[31] J. Dean and S. Ghemawat. Map Reduce: A flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[32] Chih-Fong Tsai, Wei-Chao Lin, and Shih-Wen Ke. Big data mining with parallel computing: A comparison of distributed and mapreduce methodologies. *Journal of Systems and Software*, 122:83 – 92, 2016.

[33] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, and et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, January 2016.

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

[35] Show-Jane Yen and Yue-Shi Lee. *Under-Sampling Approaches for Improving Prediction of the Minority Class in an Imbalanced Dataset*, pages 731–740. Springer Berlin Heidelberg, 2006.

[36] J. Maillo, S. García, J. Luengo, F. Herrera, and I. Triguero. Fast and scalable approaches to accelerate the fuzzy k nearest neighbors classifier for big data. *IEEE Transactions on Fuzzy Systems*, 2019 - In press.

Bibliography

[1] Travis B Murdoch and Allan S Detsky. The inevitable application of big data to health care. *Jama*, 309(13):1351–1352, 2013.

[2] Zeynep Tufekci. Big questions for social media big data: Representativeness, validity and other methodological pitfalls. In *Eighth International AAAI Conference on Weblogs and Social Media*, 2014.

[3] Roberto J Gonzalez. Hacking the citizenry?: Personality profiling, big data and the election of donald trump. *Anthropology Today*, 33(3):9–12, 2017.

[4] Peter V Coveney, Edward R Dougherty, and Roger R Highfield. Big data need big theory too. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2080):20160153, 2016.

[5] Alexandros Labrinidis and Hosagrahar V Jagadish. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5(12):2032–2033, 2012.

[6] Gregory Piateski and William Frawley. *Knowledge discovery in databases.* MIT press, 1991.

[7] R. K. Lomotey and R. Deters. Towards knowledge discovery in big data. In *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, pages 181–191, April 2014.

[8] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques.* Elsevier, 2011.

[9] Salvador García, Julián Luengo, and Francisco Herrera. *Data Preprocessing in Data Mining.* Springer Publishing Company, Incorporated, 2015.

[10] Fernando Iafrate. *A Journey from Big Data to Smart Data*, pages 25–33. Springer International Publishing, 2014.

[11] Alexander Lenk, Leif Bonorden, Astrid Hellmanns, Nico Rödder, and Stefan Jähnichen. Towards a taxonomy of standards in smart data. In *Proceedings of the 2015 IEEE International Conference on Big Data (Big Data)*, pages 1749–1754, 2015.

[12] John W Tukey. *Exploratory data analysis*, volume 2. Reading, Mass., 1977.

[13] Ana C Lorena, Luís PF Garcia, Jens Lehmann, Marcilio CP Souto, and Tin Kam Ho. How complex is your classification problem?: A survey on measuring classification complexity. *ACM Computing Surveys (CSUR)*, 52(5):107, 2019.

[14] 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, Nov 2016.

[15] Shichao Zhang, Chengqi Zhang, and Qiang Yang. Data preparation for data mining. *Applied artificial intelligence*, 17(5-6):375–381, 2003.

[16] Andreas Schmidt, Martin Atzmueller, and Martin Hollender. Data preparation for big data analytics: methods and experiences. In *Enterprise Big Data Engineering, Analytics, and Management*, pages 157–170. IGI Global, 2016.

[17] 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.

[18] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019.

[19] Julián Luengo, Salvador García, and Francisco Herrera. On the choice of the best imputation methods for missing values considering three groups of classification methods. *Knowledge and information systems*, 32(1):77–108, 2012.

[20] 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.

[21] Hang Yang and Simon Fong. Incrementally optimized decision tree for noisy big data. In *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, pages 36–44. ACM, 2012.

[22] Isaac Triguero, Daniel Peralta, Jaume Bacardit, Salvador García, and Francisco Herrera. Mrpr: A mapreduce solution for prototype reduction in big data classification. *neurocomputing*, 150:331–345, 2015.

[23] Sergio Ramírez-Gallego, Héctor Mouriño-Talín, David Martínez-Rego, Verónica Bolón-Canedo, José Manuel Benítez, Amparo Alonso-Betanzos, and Francisco Herrera. An information theory-based feature selection framework for big data under apache spark. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(9):1441–1453, 2017.

[24] Charu C Aggarwal. *Data mining: the textbook.* Springer, 2015.

[25] X. Wu, X. Zhu, G. Wu, and W. Ding. Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):97–107, Jan 2014.

[26] John A. Hartigan. *Clustering Algorithms.* John Wiley & Sons, Inc., 1975.

[27] Krzysztof J Cios, Roman W Swiniarski, Witold Pedrycz, and Lukasz A Kurgan. Unsupervised learning: association rules. In *Data Mining*, pages 289–306. Springer, 2007.

[28] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification.* John Wiley & Sons, 2012.

[29] Vladimir Cherkassky and Filip M Mulier. *Learning from data: concepts, theory, and methods.* John Wiley & Sons, 2007.

[30] Xiaojin Zhu and Andrew B Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009.

[31] Isaac Triguero, Salvador García, and Francisco Herrera. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information systems*, 42(2):245–284, 2015.

[32] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[33] Xindong Wu and Vipin Kumar, editors. *The Top Ten Algorithms in Data Mining.* Chapman & Hall/CRC Data Mining and Knowledge Discovery, 2009.

[34] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[35] Jeffrey K. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175 – 179, 1991.

[36] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, November 1998.

[37] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 459–468, Oct 2006.

[38] Jeffrey S Beis and David G Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *cvpr*, volume 97, page 1000. Citeseer, 1997.

[39] J. M. Keller, M. R. Gray, and J. A. Givens. A fuzzy k-nearest neighbor algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(4):580–585, July 1985.

[40] Joaquin Derrac, Salvador García, and Francisco Herrera. Fuzzy nearest neighbor algorithms: Taxonomy, experimental analysis and prospects. *Information Sciences*, 260:98 – 119, 2014.

[41] Joaquín Derrac, Francisco Chiclana, Salvador García, and Francisco Herrera. Evolutionary fuzzy k-nearest neighbors algorithm using interval-valued fuzzy sets. *Information Sciences*, 329:144–163, 2016.

[42] 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. *WIREs Data Mining and Knowledge Discovery*, 4(5):380–409, 2014.

[43] J. Dean and S. Ghemawat. Map Reduce: A flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[44] Chih-Fong Tsai, Wei-Chao Lin, and Shih-Wen Ke. Big data mining with parallel computing: A comparison of distributed and mapreduce methodologies. *Journal of Systems and Software*, 122:83 – 92, 2016.

[45] Tom White. *Hadoop: The Definitive Guide.* O'Reilly Media, Inc., 3rd edition, 2012.

[46] 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*, pages 1–14, 2012.

[47] Sean Owen, Robin Anil, Ted Dunning, and Ellen Friedman. *Mahout in Action.* Manning Publications Co., 2012.

[48] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, and et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, January 2016.

[49] Spark packages contributors. Spark packages: A community index of third-party packages for apache spark, 2020. `https://spark-packages.org/`, [Online; accessed 15 January 2020].

[50] Microsoft Corporation. MMLSpark: Microsoft Machine Learning for Apache Spark, 2020. `https://mmlspark.blob.core.windows.net/website/index.html`, [Online; accessed 15 January 2020].

[51] Mark Hamilton, Sudarshan Raghunathan, Ilya Matiach, Andrew Schonhoffer, Anand Raman, Eli Barzilay, Karthik Rajendran, Dalitso Banda, Casey Jisoo Hong, Manon Knoertzer, Ben Brodsky, Minsoo Thigpen, Janhavi Suresh Mahajan, Courtney Cochrane, Abhiram Eswaran, and Ari Green. Mmlspark: Unifying machine learning ecosystems at massive scales, 2018.

[52] Ellis Horowitz and Alessandro Zorat. Divide-and-conquer for parallel processing. *IEEE Transactions on Computers*, 32(6):582–585, 1983.

[53] D. Cheng M. Zong Z. Deng, X. Zhu and S. Zhang. Efficient knn classification algorithm for big data. *Neurocomputing*, 195:143 – 148, 2016. Learning for Medical Imaging.

[54] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[55] Kai Sun, Hoon Kang, and Ho-Hyun Park. Tagging and classifying facial images in cloud environments based on kNN using mapreduce. *Optik - International Journal for Light and Electron Optics*, 126(21):3227 – 3233, 2015.

[56] G. Chatzimilioudis, C. Costa, D. Zeinalipour-Yazti, W. C. Lee, and E. Pitoura. Distributed in-memory processing of all k nearest neighbor queries. *IEEE Transactions on Knowledge and Data Engineering*, 28(4):925–938, April 2016.

[57] Malak El Bakry, Soha Safwat, and Osman Hegazy. Big data classification using fuzzy k-nearest neighbor. *International Journal of Computers and Applications*, 13210:8–13, 2015.

[58] S. García, J. Derrac, J.R. Cano, and F. Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):417–435, 2012.

[59] I. Triguero, J. Derrac, S. García, and F. Herrera. A taxonomy and experimental study on prototype generation for nearest neighbor classification. *IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews*, 42(1):86–100, 2012.

[60] Isaac Triguero, Daniel Peralta, Jaume Bacardit, Salvador García, and Francisco Herrera. MRPR: A mapreduce solution for prototype reduction in big data classification. *Neurocomputing*, 150:331–345, 2015.

[61] Huan Liu and Hiroshi Motoda. *Computational methods of feature selection*. CRC Press, 2007.

[62] Daniel Peralta, Sara del Río, Sergio Ramírez-Gallego, Isaac Triguero, Jose M Benitez, and Francisco Herrera. Evolutionary feature selection for big data classification: A mapreduce approach. *Mathematical Problems in Engineering*, 2015, 2016.

[63] Gustavo E. A. P. A. Batista and Maria Carolina Monard. An analysis of four missing data treatment methods for supervised learning. *Applied Artificial Intelligence*, 17(5-6):519–533, 2003.

[64] Daniel Cheng, Peter Schretlen, Nathan Kronenfeld, Neil Bozowsky, and William Wright. Tile based visual analytics for twitter big data exploratory analysis. In *2013 IEEE International Conference on Big Data*, pages 2–4. IEEE, 2013.

[65] Ryan Hafen and Terence Critchlow. Eda and ml–a perfect pair for large-scale data analysis. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, pages 1894–1898. IEEE, 2013.

[66] Muhammad Fahim Uddin, Navarun Gupta, et al. Seven v's of big data understanding big data to extract value. In *Proceedings of the 2014 zone 1 conference of the American Society for Engineering Education*, pages 1–5. IEEE, 2014.

[67] Taher Haveliwala. Efficient computation of pagerank. Technical report, Stanford, 1999.

[68] Sarah M Ayyad, Ahmed I Saleh, and Labib M Labib. Gene expression cancer classification using modified k-nearest neighbors technique. *BioSystems*, 176:41–51, 2019.

[69] Simon Oh, Young-Ji Byon, and Hwasoo Yeo. Improvement of search strategy with k-nearest neighbors approach for traffic state prediction. *IEEE Transactions on Intelligent Transportation Systems*, 17(4):1146–1156, 2015.

[70] Wei-Chao Lin, Shih-Wen Ke, and Chih-Fong Tsai. Cann: An intrusion detection system based on combining cluster centers and nearest neighbors. *Knowledge-based systems*, 78:13–21, 2015.

[71] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[72] Fabrizio Angiulli. Fast nearest neighbor condensation for large data sets classification. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1450–1464, 2007.

[73] Yihua Chen, Eric K. Garcia, Maya R. Gupta, Ali Rahimi, and Luca Cazzanti. Similarity-based classification: Concepts and algorithms. *Journal of Machine Learning Research*, 10:747–776, 2009.

[74] Kilian Q. Weinberger and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, 2009.

[75] Ting Liu, Charles J Rosenberg, and Henry A Rowley. Performing a parallel nearest-neighbor matching operation using a parallel hybrid spill tree, January 6 2009. US Patent 7,475,071.

[76] Ting Liu, Andrew W Moore, Ke Yang, and Alexander G Gray. An investigation of practical approximate nearest neighbor algorithms. In *Advances in neural information processing systems*, pages 825–832, 2005.

[77] Malak El Bakry, Soha Safwat, and Osman Hegazy. Big data classification using fuzzy k-nearest neighbor. *International Journal of Computers and Applications*, 13210:8–13, 2015.

[78] S. García, J.R. Cano, and F. Herrera. A memetic algorithm for evolutionary prototype selection: A scaling up approach. *Pattern Recognition*, 41(8):2693–2709, 2008.

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

[80] Diego García-Gil, Francisco Luque-Sánchez, Julián Luengo, Salvador García, and Francisco 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.

[81] Tin Kam Ho and Mitra Basu. Complexity measures of supervised classification problems. *IEEE transactions on pattern analysis and machine intelligence*, 24(3):289–300, 2002.

[82] Luís PF Garcia, André CPLF de Carvalho, and Ana C Lorena. Effect of label noise in the complexity of classification problems. *Neurocomputing*, 160:108–119, 2015.

[83] Ramón A Mollineda, J Salvador Sánchez, and José M Sotoca. Data characterization for effective prototype selection. In *Iberian Conference on Pattern Recognition and Image Analysis*, pages 27–34. Springer, 2005.

[84] Lisa Cummins. Combining and choosing case base maintenance algorithms. *PhD Thesis, University College Cork*, 2013.

[85] Tin Kam Ho and Henry S Baird. Pattern classification with compact distribution maps. *Computer Vision and Image Understanding*, 70(1):101–110, 1998.

[86] Albert Orriols-Puig, Núria Macia, and Tin Kam Ho. Documentation for the data complexity library in c++. *Universitat Ramon Llull, La Salle*, 2010.

[87] Besay Montesdeoca, Julián Luengo, Jesus Maillo, Diego García-Gil, Salvador García, and Francisco Herrera. A first approach on big data missing values imputation. In *4th International Conference on Internet of Things, Big Data and Security. 2-4 May, 2019*, pages 315–323, 2019.

[88] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Machine Learning*. Springer, 2019.

[89] Albert Bifet, Gianmarco de Francisci Morales, Jesse Read, Geoff Holmes, and Bernhard Pfahringer. Efficient online evaluation of big data stream classifiers. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 59–68, 2015.