Received: 2 May 2018 | Revised and accepted: 26 September 2018

DOI: 10.1002/widm.1289

WILEY WIREs DATA MINING AND KNOWLEDGE DISCOVERY

ADVANCED REVIEW

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

Isaac Triguero[1] | Diego García-Gil[2] | Jesús Maillo[2] | Julián Luengo[2] | Salvador García[2] | Francisco Herrera[2]

[1]School of Computer Science, University of Nottingham, Nottingham, UK

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

Correspondence
Isaac Triguero, School of Computer Science, University of Nottingham, Nottingham, UK.
Email: isaac.triguero@nottingham.ac.uk

The k-nearest neighbors algorithm is characterized 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 neighbors 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 are investigated. This includes a brief overview of Smart Data, current and future trends for the k-nearest neighbor 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 neighbor 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 analyzed.

This article is categorized under:
Technologies > Data Preprocessing
Fundamental Concepts of Data and Knowledge > Big Data Mining
Technologies > Classification

KEYWORDS

big data, data preprocessing, instance reduction, K nearest neighbours, imperfect data, smart data, instance reduction, spark

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 (Chen, Chiang, and Storey 2012; Al-Fuqaha, Guizani, Mohammadi, Aledhari, and Ayyash 2015; Figueredo et al. 2017; Ramírez-Gallego,

Fernández, García, Chen, and Herrera 2018). 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 Marx (2013). Nevertheless, both companies and researchers are facing major challenges to cope with the Volume, Velocity, Veracity, and Variety (among others V's) that characterize this flood of data. These V's define the main issues of the Big Data problem Fernández et al. (2014).

The premise of Big Data is that having a world rich in data may enable machine learning and data mining techniques Aha, Kibler, and Albert (1991) 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 Dean and Ghemawat (2010); Zaharia et al. (2012), some classical data mining algorithms are being adapted to this new data-intensive scenario Philip-Chen and Zhang (2014); Gupta, Sharma, and Jindal (2016). 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 Iafrate (2014) refers to the challenge of transforming raw data into quality data that can be appropriately exploited to obtain valuable insights Lenk, Bonorden, Hellmanns, Rödder, and Jähnichen (2015). 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 twofold: higher quality data mining and reduction of data storage costs.

Data preprocessing García, Luengo, and Herrera (2015) 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 neighbor algorithm (k-NN) Cover and Hart (1967) (Figure 1). 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 Garcia, Feldman, Gupta, and Srivastava (2010), 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

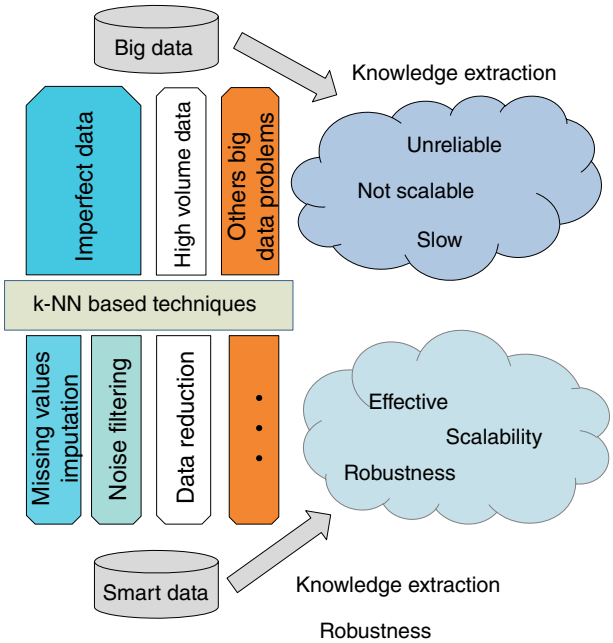

FIGURE 1 The k-nearest Neighbors algorithm plays a key role to cope with Big Data by transforming it into Smart data that is free of redundant information, noise and/or missing values. Gleaning quality data is essential for a correct data mining process that will uncover valuable insights

inability to work with incomplete information. Based on MapReduce, different distributed alternatives have recently emerged to enable k-NN to handle Big Data Zhang, Li, and Jestes (2012); Sun, Kang, and Park (2015); Maillo, Luengo, García, Herrera, and Triguero (2017), alleviating memory and 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 García, Derrac, Cano, and Herrera (2012); Triguero, Derrac, García, and Herrera (2012); Triguero, Peralta, Bacardit, García, and Herrera (2015)) or attributes (feature selection Liu and Motoda (2007); Peralta et al. (2016)) 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 neighbors). Dealing with incomplete information such as missing values is a big challenge for most data mining techniques Luengo, García, and Herrera (2012), 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, Batista and Monard (2003)) based on the k nearest neighbors. 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 Cano, Herrera, and Lozano (2003); Luengo et al. (2012). This work reviews the current specialized literature that revolves around the idea of the k-NN to come up with Smart Data, greatly extending our preliminary contribution in Triguero, Maillo, Luengo, García, and Herrera (2016) 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 characterize the behavior 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 summarize 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 characterized 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 many different faces such as data privacy/security, storage infrastructure, visualization 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) Snir and Otto (1998). 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, minimizing 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 Dean and Ghemawat (2010) and its open-source implementation in Hadoop White (2012) were the precursors of 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 slowdown. Many other frameworks have been made available to address these limitations of Hadoop, and one of the most popular platform nowadays is Apache Spark Zaharia et al. (2012). 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 Ramírez-Gallego et al. (2018). 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 behavior 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 Lenk et al. (2015). Once Smart Data are obtained, real time interactions with other business intelligence or transactional applications are affordable, evolving from data-centered to learning organizations, where knowledge is the core instead of data management Iafrate (2014).

In the traditional knowledge discovery process in databases, extracting the *value* in the data was achieved by means of data preprocessing García et al. (2015). Big Data preprocessing has now become an open, emergent topic that draws much attention nowadays García, Ramírez-Gallego, Luengo, Benítez, and Herrera (2016). 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 Lenk et al. (2015) (Figure 2).

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

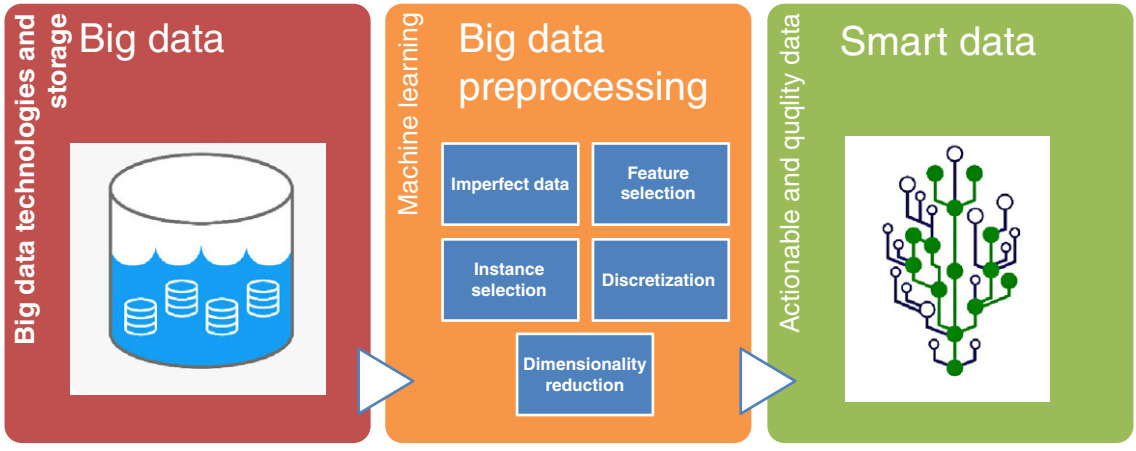

FIGURE 2 Big data preprocessing is the key to transform raw big data into quality and smart data

process, the usage of ontologies has recently emerged as a popular option Fadili and Jouis (2016); Chen, Dosyn, Lytvyn, and Sachenko (2017). Graph databases are also a common choice, storing the data in a relational schema, especially in health care domains Raja, Sivasankar, and Pitchiah (2015).

Even when the integration phase ends, data may still be far from being "smart". As data grows (especially in dimensionality), noise accumulates Fan and Fan (2008) and algorithmic instability appears Fan, Han, and Liu (2014). 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 Frénay and Verleysen (2014).

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 Peralta et al. (2016); Ramírez-Gallego et al. (2017); Tan, Tsang, and Wang (2014), instance selection techniques Triguero et al. (2015) and discretization procedures Ramírez-Gallego et al. (2016).

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 del Río, López, Benítez, and Herrera (2014); Rastogi, Narang, and Siddiqui (2018), the introduced artificial minority examples increment the data size. For this reason, novel preprocessing approaches are being explored by researchers Fernández, del Río, Chawla, and Herrera (2017).

The implementation of the aforementioned approaches is gathered in specialized packages of the main Big Data programming frameworks (such as Spark's MLlib Meng et al. (2016)). 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 neighbors to predict a target output (i.e., a class label) that a queried object or sample should have by the principle of similarity Biau and Devroye (2015). 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 Cover and Hart (1967): a sample is classified by estimating the majority vote of its neighbors, with the new object assigned to the class that is most common among its nearest neighbors (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 x_p is a vector $(x_{p1}, x_{p2}, ..., x_{pD}, \omega)$, where, 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 neighbors.

The k-NN algorithm belongs to the family of lazy learning Garcia et al. (2010), 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 Indyk and Motwani (1998). More advanced versions of the k-NN algorithm that are capable of improving the performance could be found by varying the voting weights, neighborhood sizes, similarity metrics, etc. Datta, Misra, and Das (2016); Zou, Wang, Chen, and Chen (2016); Pan, Wang, and Ku (2017).

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 neighbor 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 neighbors, we typically need to maintain a priority queue with the top nearest neighbors, 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.

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, Maillo, Ramírez, Triguero, and Herrera (2017) proposed a technological solution based on Apache Spark Zaharia et al. (2012) 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 neighbors. 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 neighbors 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 (Section 4.1) and correct data imperfections (Section 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 while 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 Liu and Motoda (2007), 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 García et al. (2012), and Instance Generation (IG) methods Triguero et al. (2012). 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 Neighbor (FCNN), which is an order-independent algorithm to find a consistent subset of the training dataset using the NN rule Angiulli (2007). Another simple yet powerful example is the Random Mutation Hill Climbing (RMHC) Skalak (1994), 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 optimization problem which consists of whether or not to select a training example Eiben, Smith, et al. (2003). 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 Cano et al. (2003). To date, one of the best performing algorithms for evolutionary PS is García, Cano, and Herrera (2008), 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 neighbors. The most popular strategy is to use merging of nearest examples to set the new artificial samples Chang (1974). We can also find clustering based approaches Bezdek and Kuncheva (2001) or evolutionary-based schemes Triguero, García, and Herrera (2010), but the vast majority of them are based on the idea of computing nearest neighbors to reduce the training set. A complete survey on this topic can be found in Triguero et al. (2012).

In terms of FS, a variety of strategies such as wrappers, filters and embedded methods have been proposed in the literature Iguyon and Elisseeff (2003). Nevertheless, we can still find that the k-NN algorithm has also played an important role in many existing FS proposals Navot, Shpigelman, Tishby, and Vaadia (2006). One of the classic and most relevant methods is ReliefF Kononenko (1994) that ranks features according to how well an attribute allows us to distinguish the nearest neighbors 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 Xue, Zhang, Browne, and Yao (2016) 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 Derrac, García, and Herrera (2010), 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 Triguero, García, and Herrera (2011) 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 Dean and Ghemawat (2010). In particular, we can find approaches based on k-NN for Big Data such as Peralta et al. (2016) 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 Palma-Mendoza, Rodriguez, and de-Marcos (2018). In Arnaiz-González, González-Rogel, Díez-Pastor, and López-Nozal (2017) 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 Triguero et al. (2015) 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 analyze the behavior 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 Triguero et al. (2015) (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 Maillo et al. (2017), 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 Frénay and Verleysen (2014). 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 categorization. The impact of noise in data has drawn the attention of researchers in the specialized literature Garcia, de Carvalho, and Lorena (2015). 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 generalization abilities, lower precision and higher computational cost Zhong, Khoshgoftaar, and Seliya (2004); Zhu and Wu (2004).

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 specialized 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 Hernández and Stolfo (1998) or misclassifications Zhu and Wu (2004), 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 García-Laencina, Sancho-Gómez, and Figueiras-Vidal (2010). Until recently, practitioners opted to discard the examples containing MVs, but this praxis often leads to severe bias in the inference process Little and Rubin (2014). 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 specialized literature, resulting in a large set of techniques than can be applied to cope with MVs in the data Luengo et al. (2012).

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 Little and Rubin (2014) 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 utilize imputators to deal with MVs. Inspired by this classification, Frénay and Verleysen Frénay and Verleysen (2014) 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 missingness mechanism by using expert knowledge and introduce it in statistical techniques as Multiple Imputation Royston et al. (2004). To avoid improperly application of correcting techniques, some test have been developed to evaluate the underlying mechanisms Little (1988) 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 Fan et al. (2014). 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 Zerhari (2016); García-Gil, Luengo, García, and Herrera (2017). 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 García et al. (2012) and Triguero et al. (2012), the authors categorized noise filtering techniques based on k-NN as sub-families of PS and PG methods: edition-based methods and class-relabeling 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-relabeling methods do not always remove the noisy instances, but they may amend those labels that the method found mistakenly assigned Sánchez, Barandela, Marqués, Alejo, and Badenas (2003).

Among all the previous categories, one of the most popular methods is the Edited Nearest Neighbor (ENN) Wilson (1972), which removes all incorrectly labeled instances that do not agree with their k nearest neighbors. If the labels are different, the instance is considered as noisy and removed. Other relevant examples of this family of methods are: All-kNN Tomek (1976), NCN-Edit Sánchez et al. (2003) or RNG Sánchez, Pla, and Ferri (1997). A distributed version of the ENN algorithm based on Apache Spark is proposed in García-Gil et al. (2017) 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 Triguero et al. (2015) 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 modeling 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 Little and Rubin (2014) leading to incorrect conclusions.

Further developments in imputation are to solve the limitations of the two previous strategies. Statistical techniques such as Expectation-Maximization Schneider (2001) or Local Least Squares Imputation Kim, Golub, and Park (2004) 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 Luengo et al. (2012).

One of the most popular imputation approaches is based on k-NN (denoted as kNN-I) Batista and Monard (2003). In this algorithm, for each instance that contains one or more MVs, it calculates the k nearest neighbors and the gaps are imputed based on the existing values of the selected neighbors. 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 neighbors. A similarity function is used to obtain the k nearest neighbors. 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 neighbors. Most imputation algorithms only utilize 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 Neighbors 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

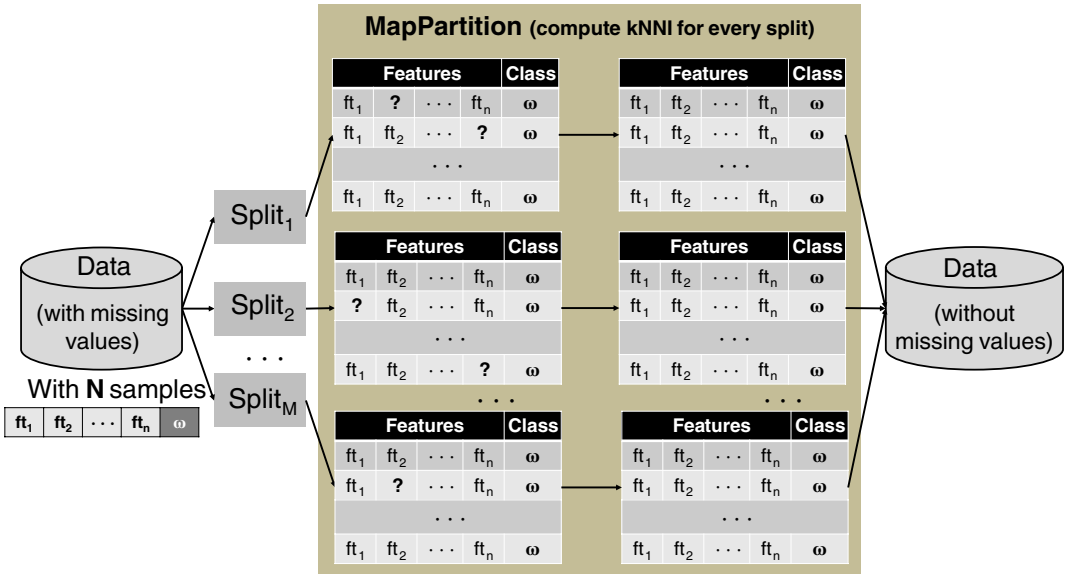

FIGURE 3 Flowchart of the kNN-LI algorithm. The dataset is split into M chunks (Map function) that are processed locally by a standard kNN-I algorithm. The resulting amended partitions are then gathered together

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 Triguero et al. (2015) 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.

5 | EXPERIMENTAL STUDY AND ANALYSIS OF RESULTS

The effectiveness of k-NN-based algorithms to obtain smart data has been widely analyzed 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 Section 5.1, we conduct a series of experiments with relevant methods for smart instance reduction (IS and IG) in Section 5.2, noise filtering in Section 5.3 and missing values imputation in Section 5.4. All the implementations of the techniques analyzed 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 ECB (2014). 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 analyze how this characteristic affects to our experiments. The remaining 6 datasets are extracted from the UCI machine learning and KEEL datasets repositories Lichman (2013); Triguero et al. (2017). 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 fivefold 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	#ω
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 analyzed 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 analyzed methods. The parameters used by these algorithms are summarized 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 analyze 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 Quinlan (1993), which is available on the Machine Learning library of Apache Spark Zaharia et al. (2012). 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.

TABLE 2 Spark packages for smart data gleaning: Brief description of the MapReduce-based methods included in these packages

Smart-reduction:	**https://spark-packages.org/package/djgarcia/SmartReduction**
FCNN_MR Angiulli (2007)	This method applies FCNN locally in separate chunks of the data, using the MRPR framework Triguero et al. (2015). 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 Arnaiz-González et al. (2017)	MR-DIS applies a condensed nearest neighbor algorithm Hart (1968) 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 Triguero et al. (2011)	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 optimized with a differential evolution algorithm. The resulting partial reduced sets are joined together.
RMHC_MR Skalak (1994)	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 Wilson (1972)	ENN_MR performs a global 1NN (based on Maillo et al. (2017) to the input data and removes examples whose nearest neighbor does not agree with its class.
All-kNN_MR Tomek (1976)	All-KNN performs ENN_MRrepeatedly with different values of k, removing instances whose class differ from its nearest neighbors. Therefore, this is also a global model.
NCNEdit_MR Sánchez et al. (2003)	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 Sánchez et al. (1997)	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 neighbors 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 Batista and Monard (2003)	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 neighbors.

TABLE 3 Parameter settings for the data preprocessing algorithms utilized

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 = .1, ReducerType = join
ENN_MR	k = 3
All-kNN_MR	k = 3
NCNEdit_MR	k = 3, ReducerType = join
RNG_MR	graphOrder = first 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

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 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.

5.2 | Smart instance reduction

The aim of this section is to analyze the behavior of relevant instance reduction methods and characterize 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 1,000 examples per partition. We acknowledge that local models may be affected by the number of partitions considered as discussed in Triguero et al. (2015) and further investigation may be required. Nevertheless, this has established a fair comparison framework for all the considered data reduction techniques.

Table 5 summarizes 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 analyzed 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 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 2,700 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.

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

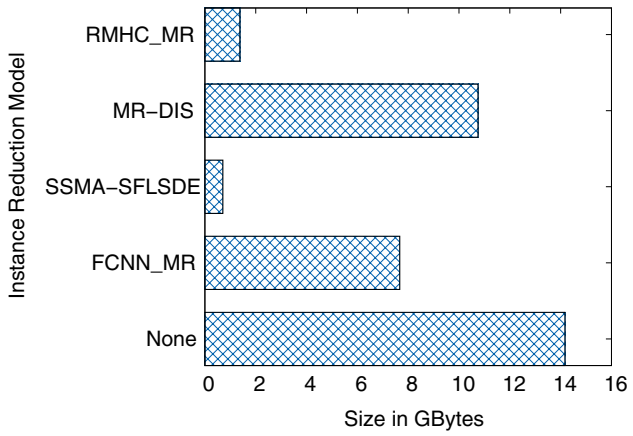

FIGURE 4 Storage requirements reduction on ECBDL'14 dataset

- 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 neighbor 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 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. Analyzing further these results, we can conclude that:

- 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 analyze 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.

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 optimizing parameters, which is typically a very time consuming operation and almost impossible on a Big Data scenario).

In this study, we have analyzed 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

TABLE 6 Impact of instance reduction on k-NN (test accuracy)

	Method				
Dataset	**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

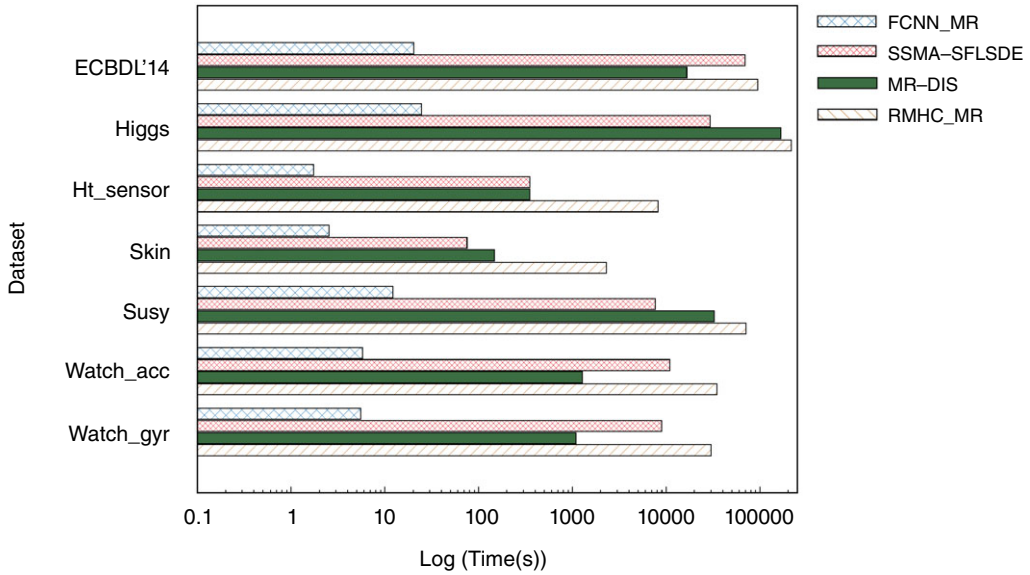

FIGURE 5 Runtime chart in logarithmic scale to perform smart reduction

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 analyze 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 analyze 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 neighbors against the entire datasets. For the NCN-Edit_MR and the RNG_MR algorithms, we have used the MRPR framework Triguero et al. (2015), 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 Triguero et al. (2015)). As before, the number of partitions has been established as a number that results in no less than 1,000 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 characterize 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 behavior of all of the analyzed 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 performance, since it is able to endure some noise while exploring clean instances. The choice of the noise filtering technique is crucial not to penalize the performance of a Decision Tree.

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 Accuracy	Reduction	All-kNN_MR Accuracy	Reduction	NCN-Edit_MR Accuracy	Reduction	RNG_MR 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

- 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.

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 1,024 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 characterize 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.

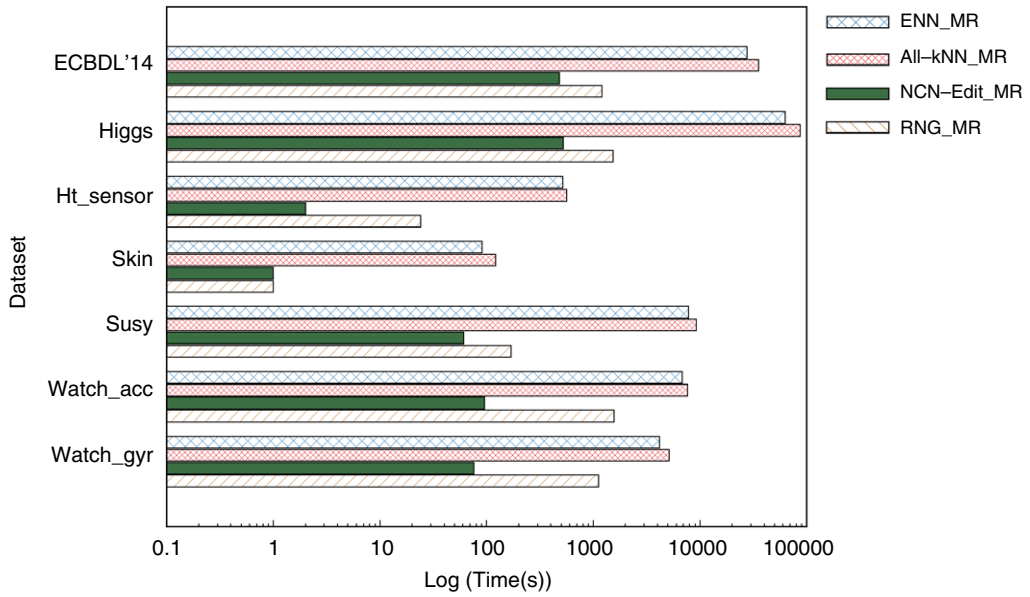

FIGURE 6 Runtime chart in logarithmic scale to perform smart filtering

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	9,994	**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

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 neighbors with 15% of MVs. Figure 7b presents the runtime with $k = 3$ for $MV\,s = 15$ and 30%.

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 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 behavior of the Decision Tree classifier in these datasets.
- Analyzing 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

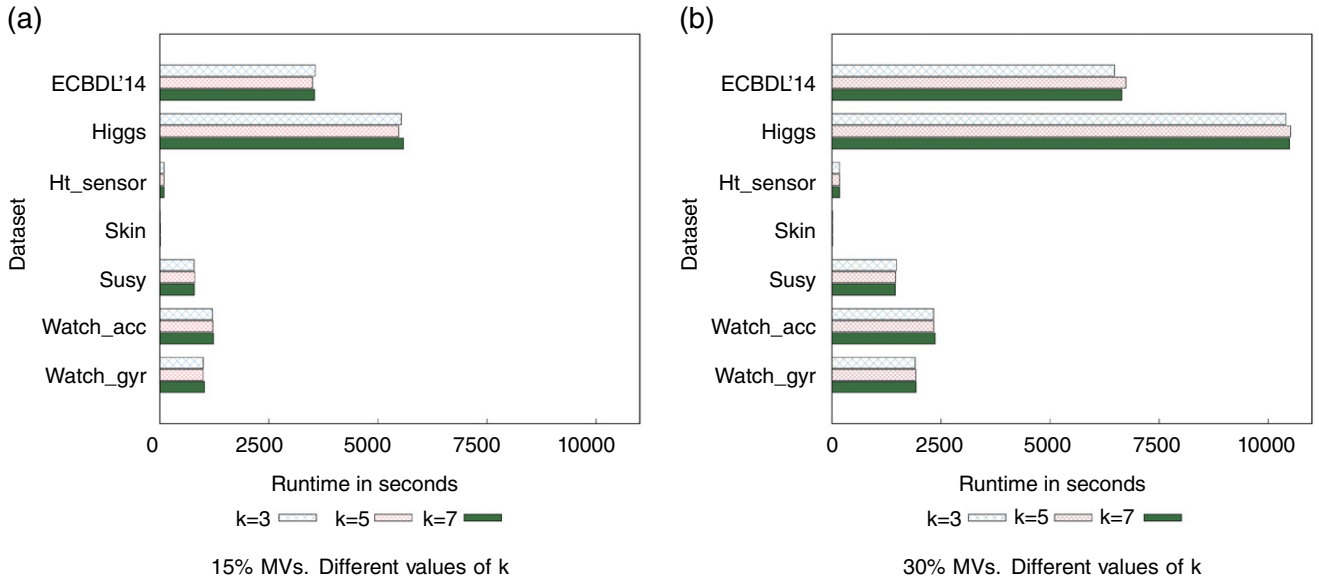

FIGURE 7 Runtime chart to perform kNN-LI

TABLE 9 Analysis of the performance of decision trees eliminating instances with MVs ("clear")

	Decision tree accuracy					
Dataset	**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

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.

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 underexplored, 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 analyzed 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 neighbors.

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 neighbors 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 Weinberger and Saul (2009); Nguyen, Morell, and Baets (2017), neighborhood sizes Pan et al. (2017) or weighting approaches Wettschereck, Aha, and Mohri (1997); Datta et al. (2016).

A very successful way to jointly handle similarity, neighborhood 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 Derrac, García, and Herrera (2014) to account for this issue, and in its easiest form - the Fuzzy k-NN algorithm Keller, Gray, and Givens (1985) - it computes a class membership degree for each single training sample, using that information to weigh the importance of the nearest neighbors. This simple idea has empirically highlighted as one of the most powerful fuzzy-based approaches to improve the k-NN algorithm Derrac et al. (2014).

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 Maillo et al. (2017) 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 parallelization. This algorithm improves upon the exact parallel k-NN algorithm Maillo et al. (2017) 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 Arya, Mount, Netanyahu, Silverman, and Wu (1998). Typically focused on domains with a large dimensionality Andoni and Indyk (2006), multiple methods have been proposed to perform a search that is approximate in nature, and therefore, assume that the actual nearest neighbors may not be found but they should be sufficiently close. Relaxing the goal of finding exactly the nearest neighbors allows to significantly run faster a search of nearest neighbors. Many of those methods are based on indexing Bertino et al. (1997), 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 neighbors.

When a Big Data problem is presented as a domain with a large number of characteristics, dimensionality reduction approaches may be needed Dutta and Ghosh (2016) to accelerate distance compensations in nearest neighbors classification. The Locality-sensitive hashing (LSH) Andoni and Indyk (2006) 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 Meng et al. (2016), and another implementation can be found at https://github.com/marufaytekin/lsh-spark.

When the data are characterized 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 Friedman, Bentley, and Finkel (1977) that perform axis parallel partitions of the data, metric trees Uhlmann (1991), which split the data with random hyperplanes, to spill-trees Liu, Moore, Yang, and Gray (2005)—a variant of metric trees where the children nodes can share objects. In Liu, Rosenberg, and Rowley (2009), a hybrid spill tree is proposed to compute parallel k-NN, hybridizing 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 neighbors 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 neighbor algorithm—as a powerful tool to obtain "Smart Data," which are data of 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 have conducted an empirical analysis of the behavior 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 Luengo, Shim, Alshomrani, Altalhi, and Herrera (2018) 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.

CONFLICT OF INTEREST

The author has declared no conflicts of interest for this article.

NOTES

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

[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.

ORCID

Isaac Triguero https://orcid.org/0000-0002-0150-0651
Diego García-Gil https://orcid.org/0000-0002-1927-8673
Julián Luengo https://orcid.org/0000-0003-3952-3629
Salvador García https://orcid.org/0000-0003-4494-7565
Francisco Herrera https://orcid.org/0000-0002-7283-312X

REFERENCES

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

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

Andoni, A., & Indyk, P. (2006). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *47th annual IEEE symposium on foundations of computer science (FOCS'06)* (pp. 459–468). Berkeley, CA, USA: IEEE.

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

Arnaiz-González, Á., González-Rogel, A., Díez-Pastor, J.-F., & López-Nozal, C. (2017). MR-DIS: Democratic instance selection for big data by MapReduce. *Progress in Artificial Intelligence*, *6*(3), 211–219.

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., & Wu, A. Y. (1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, *45*(6), 891–923.

Batista, G. E. A. P. A., & Monard, M. C. (2003). An analysis of four missing data treatment methods for supervised learning. *Applied Artificial Intelligence*, *17*(5–6), 519–533.

Bertino, E. C., Tan, K.-L., Ooi, B. C., Sacks-Davis, R., Zobel, J., & Shidlovsky, B. (1997). *Indexing techniques for advanced database systems*. Norwell, MA: Kluwer Academic Publishers.

Bezdek, J. C., & Kuncheva, L. I. (2001). Nearest prototype classifier designs: An experimental study. *International Journal of Intelligent Systems*, *16*(12), 1445–1473.

Biau, G., & Devroye, L. (2015). *Lectures on the nearest neighbor method*. Switzerland: Springer Series in the data Sciences.

Cano, J. R., Herrera, F., & Lozano, M. (2003). Using evolutionary algorithms as instance selection for data reduction in KDD: An experimental study. *IEEE Transactions on Evolutionary Computation*, *7*(6), 561–575.

Chang, C. L. (1974). Finding prototypes for nearest neighbor classifiers. *IEEE Transactions on Computers*, *100*(11), 1179–1184.

Chen, H., Chiang, R. H. L., & Storey, V. C. (2012). Business intelligence and analytics: From big data to big impact. *Management Information Systems Quarterly*, *36*(4), 1165–1188.

Chen, J., Dosyn, D., Lytvyn, V., & Sachenko, A. (2017). Smart data integration by goal driven ontology learning. In *Advances in intelligent systems and computing* (Vol. 529, pp. 283–292). Switzerland: Springer International Publishing.

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

Datta, S., Misra, D., & Das, S. (2016). A feature weighted penalty based dissimilarity measure for k-nearest neighbor classification with missing features. *Pattern Recognition Letters*, *80*, 231–237.

Dean, J., & Ghemawat, S. (2010). Map reduce: A flexible data processing tool. *Communications of the ACM*, *53*(1), 72–77.

del Río, S., López, V., Benítez, J. M., & Herrera, F. (2014). On the use of mapreduce for imbalanced big data using random forest. *Information Sciences*, *285*, 112–137.

Derrac, J., García, S., & Herrera, F. (2010). IFS-CoCo: Instance and feature selection based on cooperative coevolution with nearest neighbor rule. *Pattern Recognition*, *43*(6), 2082–2105.

Derrac, J., García, S., & Herrera, F. (2014). Fuzzy nearest neighbor algorithms: Taxonomy, experimental analysis and prospects. *Information Sciences*, *260*, 98–119.

Dutta, S., & Ghosh, A. K. (2016). On some transformations of high dimension, low sample size data for nearest neighbor classification. *Machine Learning*, *102*(1), 57–83.

Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing* (Vol. 53). Switzerland: Springer Verlag.

Fadili, H., & Jouis, C. (2016). 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* (pp. 223–230). New York, NY, USA: ACM.

Fan, J., & Fan, Y. (2008). High dimensional classification using features annealed independence rules. *Annals of Statistics*, *36*(6), 2605–2637.

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

Fernández, A., del Río, S., Chawla, N. V., & Herrera, F. (2017). An insight into imbalanced big data classification: Outcomes and challenges. *Complex & Intelligent Systems*, *3*(2), 105–120.

Fernández, A., del Río, S., López, V., Bawakid, A., del Jesús, M. J., Benítez, J. M., & Herrera, F. (2014). 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.

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

Frénay, B., & Verleysen, M. (2014). Classification in the presence of label noise: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(5), 845–869.

Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, *3*(3), 209–226.

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

Garcia, L. P. F., de Carvalho, A. C. P. L. F., & Lorena, A. C. (2015). Effect of label noise in the complexity of classification problems. *Neurocomputing*, *160*, 108–119.

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

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

García, S., Luengo, J., & Herrera, F. (2015). *Data preprocessing in data mining*. Switzerland: Springer International Publishing.

García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J. M., & Herrera, F. (2016). Big data preprocessing: Methods and prospects. *Big Data Analytics*, *1*(1), 9.

García-Gil, D., Luengo, J., García, S., & Herrera, F. (2017). Enabling smart data: Noise filtering in big data classification. *CoRR*, abs/1704.01770. Retrieved from http://arxiv.org/abs/1704.01770

García-Laencina, P. J., Sancho-Gómez, J.-L., & Figueiras-Vidal, A. R. (2010). Pattern classification with missing data: A review. *Neural Computing and Applications*, *19*(2), 263–282.

Gupta, P., Sharma, A., & Jindal, R. (2016). Scalable machine learning algorithms for big data analytics: A comprehensive review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *6*(6), 194–214.

Hart, P. E. (1968). The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, *18*, 515–516.

Hernández, M. A., & Stolfo, S. J. (1998). Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, *2*, 9–37.

Iafrate, F. (2014). *A journey from big data to smart data* (pp. 25–33). Switzerland: Springer International Publishing.

Iguyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, *3*, 1157–1182.

Indyk, P. & Motwani, R. (1998). : (pp. 604–613).

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

Kim, H., Golub, G. H., & Park, H. (2004). Missing value estimation for DNA microarray gene expression data: Local least squares imputation. *Bioinformatics*, *21*(2), 187–198.

Kononenko, I. (1994). Estimating attributes: Analysis and extensions of relief. In *Machine learning: ECML-94* (pp. 171–182). Berlin, Heidelberg: Springer Verlag.

Lenk, A., Bonorden, L., Hellmanns, A., Rödder, N., & Jähnichen, S. (2015). Towards a taxonomy of standards in smart data. In *Proceedings of the 2015 I.E. international conference on big data (big data)* (pp. 1749–1754). Santa Clara, CA, USA: IEEE.

Lichman, M. (2013). *UCI machine learning repository.* Retrieved from http://archive.ics.uci.edu/ml

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

Little, R. J. A., & Rubin, D. B. (2014). *Statistical analysis with missing data* (Vol. 333). Hoboken, NJ: John Wiley & Sons.

Liu, H., & Motoda, H. (2007). *Computational methods of feature selection*. Boca Raton, Florida: Chapman and Hall/CRC Press.

Liu, T., Moore, A. W., Yang, K., & Gray, A. G. (2005). An investigation of practical approximate nearest neighbor algorithms. In *Advances in neural information processing systems* (pp. 825–832). Cambridge, MA: MIT Press.

Liu, T., Rosenberg, C. J., & Rowley, H. A. (2009). Performing a parallel nearest-neighbor matching operation using a parallel hybrid spill tree. *U.S. Patent No.* 7,475,071.

Luengo, J., García, S., & Herrera, F. (2012). 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.

Luengo, J., Shim, S.-O., Alshomrani, S., Altalhi, A., & Herrera, F. (2018). Cnc-nos: Class noise cleaning by ensemble filtering and noise scoring. *Knowledge-Based Systems*, *140*, 27–49.

Maillo, J., Luengo, J., García, S., Herrera, F., & Triguero, I. (2017). Exact fuzzy k-nearest neighbor classification for big datasets. In *IEEE international conference on fuzzy systems (FUZZ-IEEE)* (pp. 1–6). Naples, Italy: IEEE.

Maillo, J., Ramírez, S., Triguero, I., & Herrera, F. (2017). kNN-IS: An iterative spark-based design of the k-nearest Neighbors classifier for big data. *Knowledge-Based Systems*, *117*, 3–15.

Marx, V. (2013). Biology: The big challenges of big data. *Nature*, *498*(7453), 255–260.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., … Talwalkar, A. (2016). Mllib: Machine learning in apache spark. *Journal of Machine Learning Research*, *17*(34), 1–7.

Navot, A., Shpigelman, L., Tishby, N., & Vaadia, E. (2006). Nearest neighbor based feature selection for regression and its application to neural activity. In *Advances in neural information processing systems* (pp. 996–1002). Cambridge, MA: MIT Press.

Nguyen, B., Morell, C., & Baets, B. D. (2017). Supervised distance metric learning through maximization of the jeffrey divergence. *Pattern Recognition*, *64*, 215–225.

Palma-Mendoza, R. J., Rodriguez, D., & de-Marcos, L. (2018). Distributed reliefF-based feature selection in spark. *Knowledge and Information Systems*, *57*, 1–20.

Pan, Z., Wang, Y., & Ku, W. (2017). A new general nearest neighbor classification based on the mutual neighborhood information. *Knowledge-Based Systems*, *121*, 142–152.

Peralta, D., del Río, S., Ramírez-Gallego, S., Triguero, I., Benitez, J. M., & Herrera, F. (2016). Evolutionary feature selection for big data classification: A mapreduce approach. *Mathematical Problems in Engineering*, *2015*.

Philip-Chen, C., & Zhang, C. Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, *275*, 314–347.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Francisco, CA: Morgan Kaufmann Publishers.

Raja, P., Sivasankar, E., & Pitchiah, R. (2015). Framework for smart health: Toward connected data from big data. In *Advances in intelligent systems and computing* (Vol. 343, pp. 423–433). Switzerland: Springer.

Ramírez-Gallego, S., Fernández, A., García, S., Chen, M., & Herrera, F. (2018). Big data: Tutorial and guidelines on information and process fusion for analytics algorithms with mapreduce. *Information Fusion*, *42*, 51–61.

Ramírez-Gallego, S., García, S., Mouriño-Talín, H., Martínez-Rego, D., Bolón-Canedo, V., Alonso-Betanzos, A., … Herrera, F. (2016). Data discretization: taxonomy and big data challenge. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *6*(1), 5–21.

Ramírez-Gallego, S., Lastra, I., Martínez-Rego, D., Bolón-Canedo, V., Benítez, J. M., Herrera, F., & Alonso-Betanzos, A. (2017). Fast-mRMR: Fast minimum redundancy maximum relevance algorithm for high-dimensional big data. *International Journal of Intelligent Systems*, *32*(2), 134–152.

Rastogi, A. K., Narang, N., & Siddiqui, Z. A. (2018). 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* (pp. 14:1–14:6). New York, NY: ACM.

Royston, P. (2004). Multiple imputation of missing values. *Stata Journal*, *4*(3), 227–241.

Sánchez, J., Pla, F., & Ferri, F. (1997). Prototype selection for the nearest neighbor rule through proximity graphs. *Pattern Recognition Letters*, *18*, 507–513.

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

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

Skalak, D. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *11th international conference on machine learning (ML'94)* (pp. 293–301). New Brunswick, NJ: Rutgers University.

Snir, M., & Otto, S. (1998). *MPI-the complete reference: The MPI Core*. Cambridge, MA: MIT Press.

Sun, K., Kang, H., & Park, H.-H. (2015). 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.

Tan, M., Tsang, I. W., & Wang, L. (2014). Towards ultrahigh dimensional feature selection for big data. *Journal of Machine Learning Research*, *15*, 1371–1429.

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

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

Triguero, I., García, S., & Herrera, F. (2010). IPADE: Iterative prototype adjustment for nearest neighbor classification. *IEEE Transactions on Neural Networks*, *21*(12), 1984–1990.

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

Triguero, I., Gonzalez, S., Moyano, J., García, S., Alcala-Fdez, J., Luengo, J., … Herrera, F. (2017). Keel 3.0: An open source software for multi-stage analysis in data mining. *International Journal of Computational Intelligence Systems*, *10*, 1238–1249.

Triguero, I., Maillo, J., Luengo, J., García, S., & Herrera, F. (2016). From big data to smart data with the k-nearest neighbours algorithm. In *2016 I.E. international conference on smart data* (pp. 859–864). Chengdu, China: IEEE.

Triguero, I., Peralta, D., Bacardit, J., García, S., & Herrera, F. (2015). MRPR: A mapreduce solution for prototype reduction in big data classification. *Neurocomputing*, *150*, 331–345.

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

Weinberger, K., & Saul, L. (2009). Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, *10*, 207–244.

Wettschereck, D., Aha, D. W., & Mohri, T. (1997). A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, *11*(1), 273–314.

White, T. (2012). *Hadoop: The definitive guide* (3rd ed.). Sebastopol, CA: O'Reilly Media, Inc.

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

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

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., … Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on networked systems design and implementation* (pp. 1–14). Berkeley, CA: USENIX Association.

Zerhari, B. (2016). Class noise elimination approach for large datasets based on a combination of classifiers. In *2nd International conference on cloud computing technologies and applications (CloudTech)* (pp. 125–130). Marrakech, Morocco: IEEE.

Zhang, C., Li, F., & Jestes, J. (2012). Efficient parallel knn joins for large data in mapreduce. In *Proceedings of the 15th international conference on extending database technology, EDBT '12* (pp. 38–49). New York, NY: ACM.

Zhong, S., Khoshgoftaar, T. M., & Seliya, N. (2004). Analyzing software measurement data with clustering techniques. *IEEE Intelligent Systems*, *19*(2), 20–27.

Zhu, X., & Wu, X. (2004). Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, *22*, 177–210.

Zou, P.-C., Wang, J., Chen, S., & Chen, H. (2016). Margin distribution explanation on metric learning for nearest neighbor classification. *Neurocomputing*, *177*, 168–178.

How to cite this article: Triguero I, García-Gil D, Maillo J, Luengo J, García S, Herrera F. Transforming big data into smart data: An insight on the use of the k-nearest neighbors algorithm to obtain quality data. *WIREs Data Mining Knowl Discov*. 2018;e1289. https://doi.org/10.1002/widm.1289