Big Data Preprocessing as the Bridge between Big Data and Smart Data: BigDaPSpark and BigDaPFlink Libraries

Diego García-Gil, Alejandro Alcalde-Barros, Julián Luengo, Salvador García and Francisco Herrera
Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, Granada, 18071, Spain

Keywords: Big Data, Apache Spark, Data Preprocessing, Smart Data, Imbalanced, Classification.

Abstract: With the advent of Big Data, terabytes of data are generated and stored every second. This raw data is far from being perfect, it contains many imperfections (noise, missing values, etc.) and is not suitable for analysis, as it will led to wrong conclusions. Data preprocessing is the set of techniques devoted to polish, clean, fix, and improve that raw data. With this preprocessed data, we would be able to find more patterns in it, and to better explain the underlaying distribution of the data. This is what is called Smart Data, raw data that has been preprocessed and is ready for being analyzed, data that contains valuable information that will led to knowledge. In this work, we present two Big Data libraries for achieving Smart Data from Big Data, BigDaPSpark and BigDaPFlink. They are built on top of two Big Data frameworks, Apache Spark and Apache Flink. Both libraries contain a series of algorithms for Big Data preprocessing, ranging from noise cleaning, to discretization, or data reduction, among many others. Additionally, we ilustrate the usage of the libraries with two cases of use.

1 INTRODUCTION

In the Big Data era, the lack of human supervision, and the automation in the data obtaining and storing process have led to the acceptance that data will be of low quality due to the presence of imperfections, redundancies or inconsistencies, among other pernicious traits. These imperfections can be produced by sensors failing, anomalous situations, or exogenous factors, among others. Low quality in data can make impossible the later learning process. The set of techniques devoted to tackle those imperfections, and to improve the quality of the data are known as Big Data preprocessing (García et al., 2014). There are different families of Big Data preprocessing algorithms, being the most widely used the data reduction techniques, imperfect data methods, and imbalance data handling. The term Smart Data (Iafrate, 2014) is used to refer to the challenging process of transforming that raw and low quality Big Data, into data that is suitable for the posterior data mining or knowledge extraction process. Therefore, achieving Smart Data stands as the challenge of extracting useful information from Big Data.

In the Big Data environment, we can find a set of frameworks devoted to work with that raw data. Apache Spark is the most popular framework for static Big Data processing. On the other hand, Apache Flink (García-Gil et al., 2017) is focused on online data streaming processing. Although both of them include a library for machine learning, their functionality for data preprocessing is very limited, as they only include a few classic and basic algorithms. This lack of Big Data preprocessing algorithms, makes the step from Big Data to Smart Data an even more challenging task.

In this paper, we introduce two Big Data preprocessing libraries, BigDaPSpark and BigDaPFlink, with all the latest algorithms for data preprocessing in Big Data. Most of them are new proposals for Big Data, while others are distributed and parallel versions of existing algorithms. These algorithms represent the state-of-the-art in Big Data preprocessing. BigDaPSpark is focused on static data preprocessing, built on top of Apache Spark. On the other hand, BigDaPFlink is oriented to online data preprocessing for Apache Flink.

We have carried out a case of study as a sample of the use of the libraries. A noise filtering algorithm from BigDaPSpark have been tested using *SUSY* dataset (5,000,000 instances & 18 attributes). For BigDaPFlink, a discretization algorithm is selected with *ht_sensor* dataset (929,000 instances & 11 attributes).

García-Gil, D., Alcalde-Barros, A., Luengo, J., García, S. and Herrera, F.
Big Data Preprocessing as the Bridge between Big Data and Smart Data: BigDaPSpark and BigDaPFlink Libraries.
DOI: 10.5220/0007738503240331
In *Proceedings of the 4th International Conference on Internet of Things, Big Data and Security (IoTBDS 2019)*, pages 324-331
ISBN: 978-989-758-369-8

The rest of the paper is organized as follows: Section 2 introduces the concepts of data reduction, imperfect data and imbalanced learning, and describes two existing Big Data libraries, MLlib and FlinkML. Section 3 depicts our proposal, two Big Data libraries for Big Data Preprocessing, one for Apache Spark, for batch data preprocessing, and other in Apache Flink, for data streaming preprocessing. Section 4 shows two cases of use of the libraries. Finally, Section 5 concludes the paper.

2 BACKGROUND

In this section, we describe the most popular families of data preprocessing algorithms: data reduction in Section 2.1, imperfect data in Section 2.2, and imbalance learning in Section 2.3. Finally, we provide an insight into the Big Data libraries MLlib and FlinkML in Section 2.4.

2.1 Data Reduction

Data Reduction is the set of techniques devoted to reduce the size of the original data, retaining as much information as possible. These techniques not only aim at obtaining a reduced set of the original data, but also achieve a lower space requirement version of the dataset. These reduced version of the datasets are achieved by removing noisy instances, redundant and irrelevant data that will led the learner to learn faster and on a better quality data.

There are three different ways of performing data reduction. Feature Selection (FS) methods and feature extraction techniques select the most relevant set of features, or construct a new one. From the instances point of view, we can differentiate between Instance Selection (IS) methods (Garcia et al., 2012), and Prototype Generation (PG) methods (Triguero et al., 2012). The goal of an IS method is to obtain a subset of the original data S, such that S does not contain noisy, redundant or irrelevant instances, and that the accuracy of the original data and the reduced set is similar. On the other hand, PG methods can generate artificial data points if necessary for a better representation of the original data. As stated previously, the objective of data reduction methods is not to just obtain a smaller version of the dataset.

The third way of performing data reduction is through discretization. Discretization is the process of transforming continuous values into categorical ones. In other words, it transforms numerical attributes into discrete ones, with a finite number of values (or intervals). The objective is to reduce the complexity of the data, and/or to remove outliers, as they will fall into one of the top or bottom intervals.

2.2 Imperfect Data

Automation in data acquisition and the lack of manual supervision entails that data can be imperfect. This can be even more severe as the number of instances and attributes grow (Fan et al., 2014). Although most techniques and algorithms presume that the data is accurate, data in the real world can be redundant or inconsistent. Data can contain imperfections that will disrupt the learning process if it is not taken into consideration. These alterations can be caused by many factors, but one of the most common are the presence of noise and missing values (MVs).

Noise is an external process that changes or alters the values of the attributes or classes of the instances(Wu and Zhu, 2008). It leads to excessively complex models with deteriorated performance. It displaces or removes instances located in key areas within a concrete class or can even disrupt the boundaries of the classes resulting in an increased boundaries overlap. Alleviating the effects of noise suppose the identification of noisy instances and their removal or relabelling.

Another alteration present in the data is the presence of MVs. MVs deserve a special attention as it has a critical impact in the learning process, as most learners suppose that the data is complete. One simple technique is to discard the MVs, but this can lead to poor performance due to the elimination of information. There are mechanisms in the literature to impute (fill-in) these MVs following some statistical procedures.

2.3 Imbalanced Learning

Among different classification scenarios, class imbalance occurs when there is an uneven representation of instances for the different classes. In the case of binary classification, if one class is over-represented against the other, the classifier will tend to focus on the majority class. In some cases with extreme imbalance, the minority class can be completely ignored by the classifiers. For this reason, numerous efforts have been carried out for correcting this imbalance (Fernández et al., 2018).

We can categorize them in three categories: data level approaches that rebalance the dataset, algorithmic level approaches that adapt the learning process towards the minority classes, and cost-sensitive solutions that adapt the cost with respect to the different classes.

2.4 MLlib & FlinkML

Apache Spark and Apache Flink are two of the most popular Big Data frameworks. The former is focused on static data processing, while the second is oriented to online data streaming. Both of them include a Big Data library for machine learning with basic algorithms for data preprocessing, namely MLlib for Apache Spark, and FlinkML for Apache Flink.

MLlib is a very powerful machine learning library built on top of Apache Spark (Meng et al., 2016). It is prepared for working with huge amounts of data. It is composed of two separated packages:

- *mllib*: the first API of the library. It is built on top of RDDs. In the future it will be replaced by the new *ml* API.
- *ml*: the latest addition to the library. It is built on top of DataFrames and DataSets and enables the use of pipelines.

MLlib contains many algorithms devoted to classification, regression, clustering, etc. But also includes some algorithms for data preprocessing, like feature extraction, transformation, dimensionality reduction, and selection. Although it may seem that it contains plenty of algorithms for data preprocessing, it contains only basic algorithms, such as normalizers, data scalers, Principal Components Analysis or $\mathcal{X}^2$ for FS, etc.

FlinkML is a machine learning library focused on data streaming. It is part of the Apache Flink project. It contains only three algorithms for data preprocessing, two of them being data scalers.

As we can see, both MLlib and FlinkML have algorithms for data preprocessing, but they only offer a limited set of them.

3 BIG DATA PREPROCESSING LIBRARIES

In this section we explain in detail the two Big Data libraries for data preprocessing, BigDaPSpark and BigDaPFlink. These libraries contains a series of state-of-the-art algorithms for two Big Data frameworks, Apache Spark and Apache Flink. These libraries are born with the objective of improving the Big Data ecosystem with new algorithms for Big Data preprocessing, in order to achieve Smart Data.

3.1 BigDaPSpark

This library is composed of a series of algorithms for Big Data preprocessing under the Apache Spark framework. It contains algorithms for feature selection, discretization, noise filtering, data reduction, missing values imputation and imbalanced learning, among others. The library is publicly available in https://sci2s.ugr.es/BigDaPSpark.

3.1.1 Feature Selection

The library contains a FS framework, implemented in a distributed fashion. It contains multiple information-theory based FS algorithms, like mRMR, InfoGain, JMI and other commonly used FS filters (Ramírez-Gallego et al., 2018b). It is also available as an Apache Spark package in https://spark-packages.org/package/sramirez/spark-infotheoretic-feature-selection

3.1.2 Discretization

The library also contains two distributed and parallel discretizers for dealing with huge amounts of data: A Distributed Evolutionary Multivariate Discretizer (DEMD) (Ramírez-Gallego et al., 2018), and Minimum Description Length Discretizer (MDLP) (Ramírez-Gallego et al., 2016). Both of these algorithms are also available as Apache Spark packages.

- DEMD is an evolutionary discretizer. It uses binary chromosomes with a wrapper fitness function that optimizes the interval selection problem by compensating two factors: the simpleness of the solutions, and the classification accuracy. In order to make DEMD able to cope with huge amounts of data, the evaluation phase has been distributed, splitting the set of chromosomes and the dataset into different partitions. Then a random cross-evaluation process is performed. It is available as an Apache Spark package in https://spark-packages.org/package/sramirez/spark-DEMD-discretizer.
- MDLP is a distributed discretizer that implements Fayyad's discretizer (Fayyad and Irani, 1993). It is based on Minimum Description Length Principle for treating non discrete datasets from a distributed perspective. It supports sparse data, multi-attribute processing and also is capable of dealing with attributes with a huge number of boundary points ($<$100K boundary points per attribute). It is available as an Apache Spark package in https://spark-packages.org/package/sramirez/spark-MDLP-discretization.

3.1.3 Noise Filtering

This section of the library is composed of two sub-libraries. The first one contains three algorithms

for removing noise in Big Data datasets: Homogeneous Ensemble (HME-BD), Heterogeneous Ensemble (HTE-BD), and ENN-BD. These algorithms are based on ensembles of classifiers, they were originally proposed in (García-Gil et al., 2019). These algorithms are also available as an Apache Spark package in https://spark-packages.org/package/djgarcia/NoiseFramework.

- HME-BD is based on a partitioning scheme of the dataset. It performs a k-fold of the input data, splitting the data into k partitions. The test partition is an unique 1kth of the fold, and the train is the rest of the partition. Then it learns a deep Random Forest (a Random Forest with deep trees) in each fold, using the train partition as input. Once the learning process is finished, each of the k models learned predict the corresponding test partition of each fold. That way, the models will predict the data that they didn't see while they were learned. The final step is to remove the noisy instances. This is done by a comparison of the original test labels with the predicted by the learners. If the labels are different, the instance is considered as noisy and removed. Finally, all the filtered partitions are joined together to compose a dataset clean of noise.
- HTE-BD shares the same workflow as HME-BD, but instead of using a unique classifier, it uses three of them. HTE-BD partitions the data performing a k-fold of the input data the same way as was described in HME-BD. Then it learns a deep Random Forest, a Logistic Regression and a 1NN. With the predictions of the three models, a voting strategy is used to determine if an instance is noisy. There are two strategies available, *majority* and *consensus*. With the former only two classifiers have to agree to take a decision. With the second, all classifiers must agree to consider an instance as noisy. The filtered partitions are joined to recompose the dataset without noise.
- ENN-BD is much simpler that the previous two. It is based on the similarity between instances (Wilson, 1972). It performs a kNN (typically k=1 or k=3) to the input data, and uses that same input data for prediction. That way, the closest neighbors for each instance are found. In order to remove the noisy instances, those neighbors are compared with the instance. If the label of the neighbors differs from the original, the instance is removed.

The second part of the noise library consists of three algorithms for noise filtering based on kNN (Triguero et al.,): AllKNN_BD, NCNEdit_BD and RNG_BD. These algorithms are available as an Apache Spark package in https://spark-packages.org/package/djgarcia/SmartFiltering

- AllKNN_BD: this method shares the same working scheme as ENN-BD with some exceptions. Instead of learning a 1NN, it learns several times kNN with different values of k (typically 1, 3 and 5) (Tomek, 1976). Each iteration it removes the instances that does not agree with its closest neighbors. As can be expected, it is a much aggressive noise filter than ENN-BD, as it applies kNN repeatedly.
- NCNEdit_BD: this algorithm uses the k nearest centroid neighborhood classification rule with the leave-one-out error estimate (Sánchez et al., 2003). It discard instances if it is misclassified using the kNCN classification rule. In the NCN classification rule, the neighborhood is not only defined by the proximity of prototypes to a given instance, but also for their *symmetrical distribution* around it.
- RNG_BD: this noise filter computes the proximity graph of the data (Sánchez et al., 1997). Then, all the graph neighbors of each instance give a vote for its class. If the label differs from the original label, the instance is considered as noise and removed.

3.1.4 Data Reduction

The library contains four algorithms for performing data reduction based on the kNN algorithm: FCNN_MR, SSMASFLSDE_MR, RMHC_MR and MR_DIS. As stated previously, the purpose of these algorithms is to obtain a reduced set of the original data that represents it as perfectly as possible. Some of these algorithms are implemented using a distributed framework, named MRPR (Triguero et al., 2015). This framework enables the use of iterative algorithms in Big Data environments by partitioning the input data in several chunks, and applying the corresponding algorithm independently to each one of them. After that process is finished, all the partitions are joined together using different strategies. All these algorithms are available as an Apache Spark Package in https://spark-packages.org/package/djgarcia/SmartReduction.

- FCNN_MR: this algorithm is one of the most extended and widely used in data reduction (Angiulli, 2007). It is an order-independent algorithm, based on the NN rule, to find a consistent subset of the training dataset. It has a quadratic time complexity in the worst-case. It also have

showed to scale well on large and multidimensional datasets.

- SSMASFLSDE_MR: this algorithm is a hybrid and evolutionary algorithm composed of two methods. The first one is a steady-state memetic algorithm (SSMA) (García et al., 2008), that selects the most representative instances of the training set. While the second one improves this subset by modifying the values of the selected instances with a scale factor local search in differential evolution (SFLSDE)(Triguero et al., 2011).
- RMHC_MR: Random Mutation Hill Climbing (RMHC) is a powerful yet simple algorithm for data reduction (Skalak, 1994). It starts by selecting a random sample of the data S. Then it randomly replaces an instance of the sample with one of the original data S^*. Next it uses both samples to calculate the classification accuracy in the complete dataset, using the kNN algorithm. The sample with the best accuracy is kept for the next iteration, were another instance will be substituted. After a determined number of iteration, the best sample is chosen.
- MR_DIS: is a parallel implementation of the democratic IS algorithm (Arnaiz-González et al., 2017). This algorithm applies a classic IS algorithm over an equally partitioned training dataset. The selected instances receive a vote. After a determined number of rounds, instances with most votes are removed from the data.

3.1.5 Missing Values Imputation

The library also contains two approaches, a global and a local implementation, for MVs imputation using the k-Nearest Neighbor Imputation, k-Nearest Neighbor - Local Imputation and k-Nearest Neighbor Imputation - Global Imputation. The difference among them is that the local version takes into account only the instances that are in the same partition, and the global version considers all the instances in the datasets. These algorithms are also available as an Apache Spark package in https://spark-packages.org/package/JMailloH/Smart_Imputation.

3.1.6 Imbalance Learning

Two popular methods for balancing a dataset are available in the library: Random UnderSampling (RUS) and Random OverSampling (ROS) (Batista et al., 2004). The former balances the dataset randomly removing instances from the majority class until the number of instances for both classes are identical. This approach works best when there is a high redundancy in the dataset, and achieves a lighter representation of the data storage-wide.

On the other hand, ROS reaches a balance in the data by replicating randomly instances from the minority class from the original data, until the number of instances from both classes is the same (or until a replication factor is reached). Depending on the posterior learning algorithm, the replication of instances may lead to overfitting.

Both algorithms are available as an Apache Spark package in https://spark-packages.org/package/saradelrio/Imb-sampling-ROS_and_RUS.

3.1.7 Random Discretization and PCA Classifier

The library also contains a classifier based on preprocessing, named PCARDE (García-Gil et al., 2018). This classifier is a distributed ensemble method that performs Random Discretization and Principal Components Analysis, both to the input data, and then joins the two resulting datasets. It is also available as an Apache Spark package in https://spark-packages.org/package/djgg/PCARD.

3.2 BigDaPFlink

This library contains six of the most popular and widely used algorithms for data preprocessing in data streaming. It is composed of three feature selection algorithms and three discretization algorithms. The library is publicly available in https://sci2s.ugr.es/BigDaPFlink.

3.2.1 Feature Selection

The library contains three of the most popular feature selection algorithms for data streaming in the literature: Information Gain, Online Feature Selection (OFS), and Fast Correlation-Based Filter (FCBF).

- Information Gain is a feature selection algorithm composed of two steps, an incremental feature ranking method, and an incremental learning algorithm that can consider a subset of the features during prediction (Naïve Bayes) (Katakis et al., 2005). First, the conditional entropy with respect to the class is computed. Then, the information gain is calculated for each attribute. Finally, once the algorithm has all the information gains for each feature, it selects the best N as features.
- OFS is an ε-greedy online feature selection method based on feature weights generated by an online classifier (in this case a neural network)

which makes a trade-off between exploration and exploitation of features (Wang et al., 2014).

- FCBF is a feature selection algorithm where the class relevance and the correlation between each feature pair of features are taken into account (Yu and Liu, 2003). It is based on information theory, it uses symmetrical uncertainty to calculate dependencies of features and the class importance. It starts with the full set of features and, using a backward selection technique with a sequential search strategy, it removes all the irrelevant and redundant features. Finally, it stops when no more features are left to eliminate.

3.2.2 Discretization

In this section we show the three online discretization algorithms for data streaming available in the library: Incremental Discretization Algorithm (IDA), Partition Incremental Discretization Algorithm (PiD) and Local Online Fusion Discretizer (LOFD). Discretization in data streaming have the challenge of the concept drift. These three methods tackle it in three different ways.

- IDA performs an approximate quantile-based discretization on the entire encountered data stream to date by keeping a random sample of the data (Webb, 2014). This sample is then used to calculate the cut points of the dataset. It uses the reservoir sampling algorithm to maintain this sample randomly updated from the entire stream. In IDA a sample of the data is used because it is not feasible nor possible to keep the complete data stream in memory.
- PiD discretizes data streams in an incremental manner (Gama and Pinto, 2006). The discretization process is performed in two steps. The first step discretizes the data using more intervals than required, keeping some statistics of it. The second and final step is to use that statistics to create the final discretization. It is constant in time and space even for infinite streams, as PiD processes all the streaming examples in a single scan.
- LOFD is a very recent proposal for online data streaming discretization. It is an online and self-adaptive discretizer (Ramírez-Gallego et al., 2018a). LOFD is capable of smoothly adapt its interval limits, reducing the negative impact of shifts (concept drift), and also to analyze the interval labeling and interaction problems. The interaction between the discretizer and the learner algorithm is addressed by providing two alike solutions. LOFD generates an online and self-adaptive discretization for streaming classification whose objective is to reduce the negative impact of fluctuations in evolving intervals.

4 CASES OF STUDY

In this section we show a real case of use of the two proposed libraries, BigDaPSpark and BigDaPFlink. We have selected one algorithm of each library: HME-BD for noise filtering, and PiD for data streaming discretization. We show how to use them with snippets of code, and the achieved results.

4.1 HME-BD

As stated previously, HME-BD is a noise filtering algorithm that removes noisy instances in a dataset. Here we show how to use the algorithm in a real case of study. For the dataset we have chosen *SUSY* dataset (5,000,000 instances & 18 attributes), present in the UCI repository (Dheeru and Karra Taniskidou, 2017). To show the performance of the noise filtering process, we have added 4 levels of random noise (5%, 10%, 15% and 20%).

First, the data must be loaded in Apache Spark. The dataset is required to be in the RDD[LabeledPoint] format (default format for Spark's MLlib).

```
import org.apache.spark.mllib._

val nTrees = 100
val maxDepth = 10
val nPartitions = 4
val seed = 12345

val hme_model = new HME_BD(
  trainingData, //RDD[LabeledPoint]
  nTrees, //size of the RFs
  nPartitions, //number of partitions
  maxDepth, //depth of the RFs
  seed) //seed for the RFs

val hme = hme_model.runFilter()
```

Once the filtering process is finished, the algorithm returns a reduced RDD without the noisy instances. Now that the data has been filtered, we can use the several classifiers available in Spark's MLlib. Here we show the results using MLlib's Decision Tree with an increased depth to 20.

Table 1 shows the accuracy results using *SUSY* dataset. As we can see, HME-BD is able to keep almost the same accuracy with the increasing levels of

Table 1: Decision Tree test accuracy.

Dataset	Noise (%)	Original	HME-BD
SUSY	5	79.94	79.99
	10	79.15	79.85
	15	78.21	79.81
	20	77.09	79.71

noise. Also, from 5% of noise onward, HME-BD improves the *Original* accuracy.

HME-BD also achieves very low runtimes, for *SUSY* dataset, it takes 514 seconds to process it.

4.2 PiD

PiD is a discretizer for data streaming. Here we show an example of the usage of the algorithm. In this case of study, we are using the *ht_sensor* dataset (929,000 instances & 11 attributes), also present in the UCI repository. The first step it to load the data into Flink's DataSet format. Once the data is loaded, the algorithm can be used in the following way.

```
import com.elbauldelprogramador._

val pid = PIDiscretizerTransformer()
  .setAlpha(.10)
  .setUpdateExamples(50)
  .setL1Bins(5)
val scaler = MinMaxScaler()
val pipeline = scaler
  .chainTransformer(pid)
pipeline fit dataSet
val result = pipeline transform dataSet
```

The results using the *ht_sensor* dataset with a decision tree as a classifier show that the accuracy improves from a baseline of 70.13% without preprocessing, to a 71.06% using PiD. Regarding computing times, it takes 118 seconds of computing.

5 CONCLUSIONS

In this work, we have introduced two Big Data preprocessing libraries. They are built on top of two Big Data frameworks, one for Apache Spark, BigDaPSpark, and another for Apache Flink, BigDaPFlink. They contain several algorithms for performing data reduction, handling imperfect data, or dealing with imbalanced data. We plan to expand the list of available algorithms in the future. With these algorithms, we have enabled the practitioner to efficiently achieve Smart Data from raw Big Data.

As we have seen, we can find a wide spectrum of techniques for Big Data preprocessing. However, there is an open challenge related to the combination and arrangement of these methods in order to achieve the best possible outcome for a data mining process. In (García et al., 2016), authors present the most popular and widely used data preprocessing algorithms, studying the effects of different arrangements in the data preprocessing chain. This challenge is even more complex in Big Data scenarios, where there is a time restriction. Methods that increase the amount of data may affect posterior preprocessing techniques, making them unable to cope with that amount of data. This complexity may also be influenced by the dependency of intermediate results, or the input that a method requires and the output it provides.

ACKNOWLEDGMENTS

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

REFERENCES

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., and López-Nozal, C. (2017). Mr-dis: democratic instance selection for big data by mapreduce. *Progress in Artificial Intelligence*, 6(3):211–219.

Batista, G. E. A. P. A., Prati, R. C., and Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explor. Newsl.*, 6(1):20–29.

Dheeru, D. and Karra Taniskidou, E. (2017). UCI machine learning repository.

Fan, J., Han, F., and Liu, H. (2014). Challenges of big data analysis. *National science review*, 1(2):293–314.

Fayyad, U. M. and Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *IJCAI*, pages 1022–1029.

Fernández, A., García, S., Galar, M., Prati, R. C., Krawczyk, B., and Herrera, F. (2018). *Learning from Imbalanced Data Sets*. Springer Publishing Company.

Gama, J. and Pinto, C. (2006). Discretization from data streams: applications to histograms and data mining. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 662–667. ACM.

García, S., Luengo, J., and Herrera, F. (2016). Tutorial on practical tips of the most influential data preprocessing algorithms in data mining. *Knowledge-Based Systems*, 98:1 – 29.

García, S., Cano, J., and Herrera, F. (2008). A memetic algorithm for evolutionary prototype selection: A scal-

ing up approach. *Pattern Recognition*, 41(8):2693–2709.

Garcia, S., Derrac, J., Cano, J., and Herrera, F. (2012). Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE transactions on pattern analysis and machine intelligence*, 34(3):417–435.

García, S., Luengo, J., and Herrera, F. (2014). *Data Preprocessing in Data Mining*. Springer Publishing Company, Incorporated.

García-Gil, D., Luengo, J., García, S., and Herrera, F. (2019). Enabling Smart Data: Noise filtering in Big Data classification. *Information Sciences*, 479:135 – 152.

García-Gil, D., Ramírez-Gallego, S., García, S., and Herrera, F. (2017). A comparison on scalability for batch big data processing on apache spark and apache flink. *Big Data Analytics*, 2(1):1.

García-Gil, D., Ramírez-Gallego, S., García, S., and Herrera, F. (2018). Principal Components Analysis Random Discretization Ensemble for Big Data. *Knowledge-Based Systems*, 150:166–174.

Iafrate, F. (2014). *A Journey from Big Data to Smart Data*, pages 25–33. Springer International Publishing.

Katakis, I., Tsoumakas, G., and Vlahavas, I. (2005). On the utility of incremental feature selection for the classification of textual data streams. In *Panhellenic Conference on Informatics*, pages 338–348. Springer.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al. (2016). Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241.

Ramírez-Gallego, S., García, S., and Herrera, F. (2018a). Online entropy-based discretization for data streaming classification. *Future Generation Computer Systems*, 86:59–70.

Ramírez-Gallego, S., García, S., Mouriño-Talín, H., Martínez-Rego, D., Bolón-Canedo, V., Alonso-Betanzos, A., Benítez, J. M., and 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., Mouriño-Talín, H., Martínez-Rego, D., Bolón-Canedo, V., Benítez, J. M., Alonso-Betanzos, A., and Herrera, F. (2018b). An information theory-based feature selection framework for big data under apache spark. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(9):1441–1453.

Ramírez-Gallego, S., García, S., Benítez, J., and Herrera, F. (2018). A distributed evolutionary multivariate discretizer for big data processing on apache spark. *Swarm and Evolutionary Computation*, 38:240 – 250.

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

Sánchez, J., Pla, F., and Ferri, F. (1997). Prototype selection for the nearest neighbour rule through proximity graphs. *Pattern Recognition Letters*, 18(6):507 – 513.

Skalak, D. B. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Machine Learning Proceedings 1994*, pages 293–301. Elsevier.

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

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

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

Triguero, I., García-Gil, D., Maillo, J., Luengo, J., García, S., and Herrera, F. Transforming big data into smart data: An insight on the use of the k-nearest neighbors algorithm to obtain quality data. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 0(0):e1289.

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

Wang, J., Zhao, P., Hoi, S. C., and Jin, R. (2014). Online feature selection and its applications. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):698–710.

Webb, G. I. (2014). Contrary to popular belief incremental discretization can be sound, computationally efficient and extremely useful for streaming data. In *2014 IEEE International Conference on Data Mining*, pages 1031–1036.

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

Wu, X. and Zhu, X. (2008). Mining with noise knowledge: error-aware data mining. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 38(4):917–932.

Yu, L. and Liu, H. (2003). Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 856–863.