Data Discretization: Taxonomy and Big Data Challenge

Journal:	*WIREs Data Mining and Knowledge Discovery*
Manuscript ID:	DMKD-00215
Wiley - Manuscript type:	Advanced Review
Date Submitted by the Author:	23-Apr-2015
Complete List of Authors:	Ramírez-Gallego, Sergio; University of Granada, Computer Science and Artificial Intelligence Garcia, Salvador; University of Granada, Computer Science and Artificial Intelligence Mouriño-Talín, Héctor; University of A Coruña, Computer Science Martínez-Rego, David; University of A Coruña, Computer Science; University College London, Computer Science Bolón-Canedo, Verónica; University of A Coruña, Computer Science Alonso-Betanzos, Amparo; University of A Coruña, Computer Science Benítez, José; University of Granada, Computer Science and Artificial Intelligence Herrera, Francisco; University of Granada, Computer Science and Artificial Intelligence
Keywords:	Data Discretization, Taxonomy, Big Data, Data Mining, Apache Spark
Choose 1-3 topics to categorize your article:	Data Preprocessing (DEAF) < Technologies (DEAA), Classification (DEAC) < Technologies (DEAA)
Note: The following files were submitted by the author for peer review, but cannot be converted to PDF. You must view these files (e.g. movies) online.	
crossreflist.tex	

SCHOLARONE™
Manuscripts

Article type: Advanced Review

Data Discretization: Taxonomy and Big Data Challenge

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

1. Department of Computer Science and Artificial Intelligence, University of Granada, 18071, Spain, {sramirez|salvagl|j.m.benitez|herrera}@decsai.ugr.es
2. Department of Computer Science, University of A Coruña, 15071 A Coruña, Spain. {h.mtalin|dmartinez|veronica.bolon|ciamparo}@udc.es
3. Department of Computer Science, University College London, WC1E 6BT London, United Kingdom.

*. Corresponding Author.

Keywords

Data Discretizacion, Taxonomy, Big Data, Data Mining, Apache Spark

Abstract

Discretization of numerical data is one of the most influential data preprocessing tasks in knowledge discovery and data mining. The purpose of attribute discretization is to find concise data representations as categories which are adequate for the learning task retaining as much information in the continuous original attribute as possible. In this paper, we present an updated overview of discretization techniques in conjunction with a complete taxonomy of the leading discretizers. Despite the great impact of discretization as data preprocessing technique, few elementary approaches have been developed in the literature for Big Data. Because of this, we design in this paper a distributed implementation using Apache Spark platform of the most probably well-known discretizer: the entropy minimization discretizer proposed by Fayyad and Irani. Our scheme goes beyond a simple parallelization and it is intended to be the first to face the Big Data challenge.

INTRODUCTION

Data is present in diverse formats, for example in categorical, numerical or continuous values. Categorical or nominal values are unsorted, whereas numerical or continuous values are assumed to be sorted or represent ordinal data. It is well-known that Data Mining (DM) algorithms depend very much on the domain and type of data. In this way, the techniques belonging to the field of statistical learning prefer numerical data (i.e., support vector machines and instance-based learning) whereas symbolic learning

For Peer Review Only

methods require inherent finite values and also prefer to perform a branch of values that are not ordered (such as in the case of decision trees or rule induction learning). These techniques either expect to work on discretized data or they integrate themselves internal mechanisms to perform discretization.

The process of discretization has aroused general interest in recent years [29, 45] and has become one of the most effective data pre-processing technique in DM [28]. Roughly speaking, discretization translates quantitative data into qualitative data, procuring a non-overlapping division of a continuous domain. It also ensures an association between each numerical value and a certain interval. Actually, discretization is considered a data reduction mechanism since it diminishes data from a large domain of numeric values to a subset of categorical values.

There is a necessity to use discretized data by many DM algorithms which can only deal with discrete attributes. For example, three of the ten methods pointed out as the top ten in DM [67] demand a data discretization in one form or another: C4.5 [53], Apriori [1] and Naïve Bayes [69]. Among its main benefits, discretization causes in learning methods remarkable improvements in learning speed and accuracy. Besides, some decision tree-based algorithms produce shorter, more compact, and accurate results when using discrete values [36, 45].

The specialized literature gathers a huge number of proposals for discretization. In fact, some surveys have been developed attempting to organize the available pool of techniques [29, 45, 70]. It is crucial to determine, when dealing with a new real problem or data set, the best choice in the selection of a discretizer. This will condition the success and the suitability of the subsequent learning phase in terms of accuracy and simplicity of the solution obtained. In spite of the effort done in [29] to categorize the whole family of discretizers, the probably most well-known and surely most effective ones are included in a new taxonomy presented in this paper, which is now updated up to the moment of writing.

Classical data reduction methods are not expected to scale well when managing huge data -both in number of features and instances- so that its application can be undermined or even become impracticable. Scalable distributed techniques and frameworks have appeared along with the problematic of Big Data. MapReduce [21] and its open-source version Apache Hadoop [3, 63] were the first distributed programming techniques to face this problem. Apache Spark [5, 32] is one of these new frameworks, designed as a fast and general engine for large-scale data processing based on in-memory computation. Through this Spark's ability, it is possible to speed up iterative processes present in many DM problems. Similarly, several DM libraries for Big Data have appeared as support for this task. The first one was Mahout [4] (as part of Hadoop), subsequently followed by MLlib [48] which is part of the Spark project [5]. Although many state-of-the-art DM algorithms have been implemented in MLlib, it is not the case for discretization algorithms yet.

In order to fill this gap, we face the Big Data challenge by presenting a distributed version of the entropy minimization discretizer proposed by Fayyad and Irani in [23] using Apache Spark, which is based on Minimum Description Length Principle. Our

main objective is to prove that well-known discretization algorithms as MDL-based discretizer (henceforth called MDLP) can be parallelized in these frameworks, providing good discretization solutions for Big Data analytics. Furthermore, we have transformed the iterativity yielded by the original proposal in a single-step computation. Notice that this new version for distributed environments has supposed a deep restructuring of the original proposal and a challenge for the authors. Finally, to demonstrate the effectiveness of our framework, we perform an experimental evaluation with two large datasets, namely, ECBDL14 and epsilon.

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

BACKGROUND AND PROPERTIES

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

Discretization Process

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

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

A typical discretization process generally consists of four steps (seen in Figure 1): (1) *sorting* the continuous values of the feature to be discretized, either (2) *evaluating* a cut

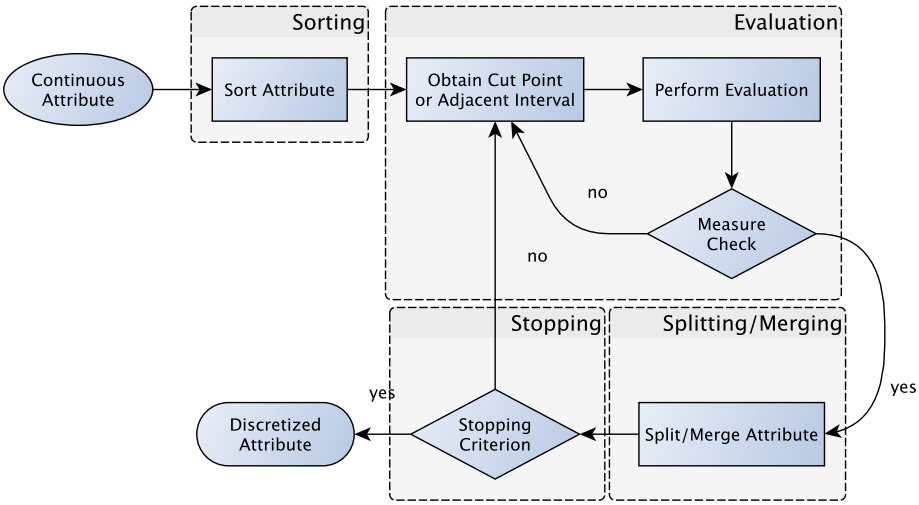

Figure 1: Discretization Process

point for splitting or adjacent intervals for merging, (3) *splitting or merging* intervals of continuous values according to some defined criterion, and (4) *stopping* at some point. Next, we explain these four steps in detail.

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

Discretization Properties

In [29, 45, 70], various pivots have been used in order to make a classification of discretization techniques. This sections reviews and describe them, underlining the major aspects and alliances found among them. The taxonomy presented after will be founded on these characteristics (acronyms of the methods correspond with the presented in Table 1):

- *Static vs. Dynamic:* This property refers to the level of independence between the discretizer and the learning method. A static discretizer is run prior to the learning task and is autonomous from the learning algorithm [45], as a data preprocessing algorithm [28]. Almost all isolated known discretizers are static. By contrast, a dynamic discretizer responds when the learner requires so, during the building of the model. Hence, they must belong to the local discretizers family (see later) embedded in the learner itself, producing accurate and compact outcome together with the associated learning algorithm. Good examples of classical dynamic techniques are ID3 discretizer [53] and ITFP [6].
- *Univariate vs. Multivariate:* Univariate discretizers only operate with a single attribute simultaneously. This means that they sort the attributes independently, and then, the derived discretization disposal for each attribute keeps unchanged in the next phases. Conversely, multivariate techniques, concurrently consider all or various attributes to determine the initial set of cut points or to make a decision about the best cut point chosen as a whole. They may accomplish discretization handling the complex interactions among several attributes to decide also the attribute in which the next cut point will be splitted or merged. Currently, interest has recently appear in developing multivariate discretizers since they are decisive in complex predictive problems where univariate operations may ignore important interactions between attributes [50, 54] and in deductive learning [51].
- *Supervised vs. Unsupervised:* Supervised discretizers consider the class label whereas unsupervised ones do not. The interaction between the input attributes and the class output and the measures used to make decisions on the best cut points (entropy, correlations, etc.) will define the supervised manner to discretize. Although most of the discretizers proposed are supervised, there is a growing interest in unsupervised discretization for descriptive tasks [25, 51]. Unsupervised discretization can be applied on both supervised and unsupervised learning, because its operation does not require to specify an output attribute. Nevertheless, this does not occur in supervised discretizers, which can only be applied over supervised learning. Unsupervised also open the door to transfer the learning between tasks since the discretization is not tailored to a specific problem.
- *Splitting vs. Merging:* These two options refer to the approach used to define or generate new intervals. The former methods search for a cut point to divide the domain into two intervals among all the possible boundary points. On the contrary, merging techniques begin with a pre-defined partition and search for a

candidate cut point to mix both adjacent intervals after removing it. In the literature, the terms *Top-Down* and *Bottom-up* are highly related to these two operations, respectively. In fact, top-down and bottom-up discretizers are thought for hierarchical discretization developments, so they consider that the process is incremental, property which will be described later. Splitting/merging is more general than top-down/bottom-up because it is possible to have discretizers whose procedure manages more than one interval at a time [42, 56]. Furthermore, we consider the *hybrid* category as the way of alternating splits with merges during running time [18, 54].

- *Global vs. Local:* In the time a discretizer must select a candidate cut point to be either splitted or merged, it could consider either all available information in the attribute or only partial information. A local discretizer makes the partition decision based only on partial information. MDLP [23] and ID3 [53] are classical examples of local methods. By definition, all the dynamic discretizers and some top-down based methods are local, which explains the fact that few discretizers apply this form. The dynamic discretizers search for the best cut point during internal operations of a certain DM algorithm, thus it is impossible to examine the complete data set. Besides, top-down procedures are associated with the divide-and-conquer scheme, in such manner that when a split is considered, the data is recursively divided, restricting access to partial data.

- *Direct vs. Incremental:* For direct discretizers, the range associated to an interval must be divided into k intervals simultaneously, requiring an additional criterion to determine the value of k. One-step discretization methods and discretizers which select more than a single cut point at every step are included in this category. However, incremental methods begin with a simple discretization and pass through an improvement process, requiring an additional criterion to determine when it is the best moment to stop. At each step, they find the best candidate boundary to be used as a cut point and, afterwards, the rest of the decisions are made accordingly.

- *Evaluation Measure:* This is the metric used by the discretizer to compare two candidate discretization schemes and decide which is more suitable to be used. We consider five main families of evaluation measures:

  - *Information:* This family includes *entropy* as the most used evaluation measure in discretization (MDLP [23], ID3 [53], FUSINTER [72]) and other derived from information theory (*Gini index*, *Mutual Information*) [38].
  - *Statistical:* Statistical evaluation involves the measurement of dependency/correlation among attributes (Zeta [33], ChiMerge [39], Chi2 [46]), interdependency [40], probability and bayesian properties [66] (MODL [13]), contingency coefficient [62], etc.
  - *Rough Sets:* This class is composed of methods that evaluate the discretization schemes by using rough set properties and measures [37], such as class separability, lower and upper approximations, etc.

- *Wrapper*: This collection comprises methods that rely on the error provided by a classifier or a set of classifiers that are used in each evaluation. Representative examples are MAD [41], IDW [26] and EMD [54].
- *Binning*: In this category of techniques, there is not an evaluation measure. This refers to discretize an attribute with a predefined number of bins in a simple way. A bin assigns a certain number of values per attribute by using a non sophisticated procedure. EqualWidth and EqualFrequency discretizers are the most well-known unsupervised binning methods.

Table 1 Most Important Discretizers

Acronym	Ref.	Acronym	Ref.	Acronym	Ref.
EqualWidth	[64]	EqualFrequency	[64]	Chou91	[20]
D2	[15]	ChiMerge	[39]	1R	[34]
ID3	[53]	MDLP	[23]	CADD	[18]
MDL-Disc	[52]	Bayesian	[66]	Friedman96	[27]
ClusterAnalysis	[19]	Zeta	[33]	Distance	[16]
Chi2	[46]	CM-NFD	[35]	FUSINTER	[72]
MVD	[8]	Modified Chi2	[61]	USD	[30]
Khiops	[12]	CAIM	[40]	Extended Chi2	[60]
Heter-Disc	[47]	UCPD	[49]	MODL	[13]
ITPF	[6]	HellingerBD	[42]	DIBD	[65]
IDD	[56]	CACC	[62]	Ameva	[31]
Unification	[38]	PKID	[69]	FFD	[69]
CACM	[43]	DRDS	[7]	EDISC	[59]
U-LBG	[25]	MAD	[41]	IDF	[26]
IDW	[26]	NCAIC	[68]	Sang14	[57]
IPD	[51]	SMDNS	[37]	TD4C	[50]
EMD	[54]				

For Peer Review

Minimum Description Length-based Discretizer

Minimum Description Length-based discretizer (MDLP) [23], proposed by Fayyad and Irani in 1993, is one of the most important splitting methods in discretization. This univariate discretizer uses the Minimum Description Length Principle to control the partitioning process. This also introduces an optimization based on a reduction of whole set of candidate points, only formed by the *boundary points* in this set.

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

This method also introduces other important improvements. One of them is related to the number of cut points to derive in each iteration. In contrast to discretizers like ID3 [53], the authors proposed a multi-interval extraction of points demonstrating that better classification models -both in error rate and simplicity- are yielded by using these schemes.

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

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

where E represents the class entropy [1] of a given subset following the Shannon's definitions [58].

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

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

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

TAXONOMY

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

Table 1 enumerates the discretizers considered in this paper, providing the abbreviation name and reference for each one. We do not include the descriptions of these discretizers in this paper. Their definitions are contained in the original references, thus we recommend to consult them to understand how the discretizers of interest work. In

[1] Logarithm in base 2 is used in this function

For Peer Review

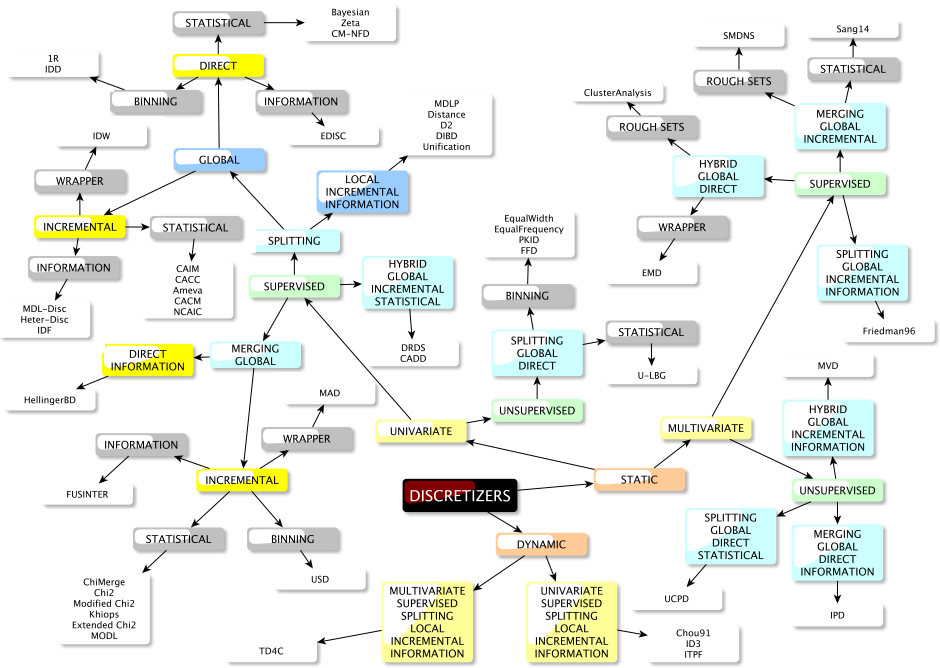

Figure 2: Discretization Taxonomy

Table 1, 30 discretizers included in KEEL software are considered. Additionally, implementations of these algorithms in Java can be found [2].

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

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

When managing huge data, most of them become impracticable in real settings, due to the complexity they arise (for example, in the case of MDLP, among others). The adaptation of these classical methods imply a deep redesign that becomes mandatory if we want to exploit the advantages derived from the use of discrete data on large datasets [11, 10]. As reflected in our taxonomy, no relevant methods in the field of Big Data have been proposed to solve this problem. Some works have tried to deal with

large-scale discretization. For example, in [14] the authors proposed a scalable implementation of Class-Attribute Interdependence Maximization algorithm by using GPU technology. In [71], it is proposed a discretizer based on windowing and hierarchical clustering to improve the performance of classical tree-based classifiers. However, none of these methods have been proved to cope with the data magnitude presented here.

BIG DATA BACKGROUND

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

Gartner [9] introduced the popular denomination of Big Data and the 3Vs terms that define it as high volume, velocity and variety information that require a new large-scale processing. This list was then extended with 2 additional terms. All of them are described in the followings: **Volume**, the massive amount of data that is produced every day is still exponentially growing (from terabytes to exabytes); **Velocity**, data needs to be loaded, analyzed and stored as quickly as possible; **Variety**, data come in many formats and representations; **Veracity**, the quality of data to process is also an important factor. The Internet is fulfilled of missing, incomplete, ambiguous, and sparse data; **Value**, extracting value from data is also established as a relevant objective in big analytics.

The unsuitability of many knowledge extraction algorithms in the Big Data field has made that new methods are developed to manage such amounts of data effectively and at a pace that allows to extract value from it.

MapReduce Model and Other Distributed Frameworks

The MapReduce framework [21], designed by Google in 2003, is currently one of the most relevant tools in Big Data analytics. It was aimed at processing and generating large-scale datasets, automatically processed in an extremely distributed fashion through several machines.[2] The MapReduce model defines two primitives to work with distributed data: **Map** and **Reduce**. These two primitives imply two stages in the distributed process, which we describe below. In the first step, the master node breaks up the dataset into several splits, distributing them across the cluster for a parallel processing. Each node then hosts several Map threads that transform the generated key-value pairs in a set of intermediate pairs. After all Map tasks have finished, the master node distributes the matching pairs across the nodes according to a key-based partitioning scheme. Now, the Reduce phase starts, combining those coincident pairs so as to form the final output.

[2]For a complete description of this model and other distributed models, please review [24].

Apache Hadoop [3, 63] is presented as the most popular open-source implementation of MapReduce for large-scale processing. Despite its popularity, Hadoop presents some important weaknesses, such as a poor performance on iterative and online computing, and a poor inter-communication capability or inadequacy for in-memory computation, among others [44]. Recently, Apache Spark [5, 32] has appeared and integrated with the Hadoop Ecosystem. This novel framework is presented as a revolutionary tool capable of performing even faster large-scale processing than Hadoop through in-memory primitives, making this framework a leading tool for iterative and online processing and, thus, suitable for DM algorithms. Spark is built on distributed data structures called Resilient Distributed Datasets (RDDs), which were designed as fault-tolerant collection of elements that can be operated in parallel by means of data partitioning.

DISTRIBUTED MDLP DISCRETIZATION

In Background Section, a discretization algorithm based on an information entropy minimization heuristic was presented [23]. In this work, the authors proved that multi-interval extraction of points and the use of boundary points can improve the discretization process, both in efficiency and error rate. Here, we adapt this well-known algorithm for distributed environments, proving its discretization capability against real-world large problems.

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

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

Spark primitives extend the idea of MapReduce to implement more complex operations on distributed data. In order to implement our method, we have used some extra primitives from Spark's API, such as: mapPartitions, sortByKey, flatMap and reduceByKey[3].

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

For Peer Review

Main discretization procedure

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

Algorithm 1 Main discretization procedure

Input: S Data set
Input: M Feature indexes to discretize
Input: mb Maximum number of cut points to select
Input: mc Maximum number of candidates per partition
Output: Cut points by feature

1: $comb \leftarrow$
2: **map** $s \in S$
3: $v \leftarrow zeros(|c|)$
4: $ci \leftarrow class\_index(v)$
5: $v(ci) \leftarrow 1$
6: **for all** $A \in M$ **do**
7: $EMIT < (A, A(s)), v >$
8: **end for**
9: **end map**
10: $distinct \leftarrow reduce(comb,\ sum\_vectors)$
11: $sorted \leftarrow sort\_by\_key(distinct)$
12: $first \leftarrow first\_by\_part(sorted)$
13: $bds \leftarrow get\_boundary(sorted, first)$
14: $bds \leftarrow$
15: **map** $b \in bds$
16: $< (att, point), q > \leftarrow b$
17: $EMIT < (att, (point, q)) >$
18: **end map**
19: $(small, big) \leftarrow divide\_attributes(bds, mc)$
20: $sth \leftarrow select\_thresholds(small, mb, mc)$
21: **for all** $a \in keys(big)$ **do**
22: $bth \leftarrow bth + select\_thresholds(big(a), mb, mc)$
23: **end for**
24: $return(union(bth, sth))$

The first step creates combinations from instances through a Map function in order to separate values by feature. It generates tuples with the value and the index for each feature as key and a class counter as value ($< (A, A(s)), v >$). Afterwards, the tuples are reduced using a function that aggregates all subsequent vectors with the same key, obtaining the class frequency for each distinct value in the dataset. The resulting tuples are sorted by key so that we obtain the complete list of distinct values ordered by feature index and feature value. This structure will be used later to evaluate all these points in a single step. The first point by partition is also calculated (line 11) for this process. Once such information is saved, the process of evaluating the boundary points can be started.

Boundary points selection

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

For Peer Review

algorithm checks whether the current point is a boundary with respect to the previous point or not. If it is so, this emits a tuple with the midpoint between these points as key and the accumulated counter as value.

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

Input: $points$ An RDD of tuples $(< (att, point), q >)$, where att represents the feature index, $point$ the point to consider and q the class counter.
Input: $first$ A vector with all first elements by partition
Output: An RDD of points.

```
1: boundaries ←
2:   map partitions part ∈ points
3:     < (la, lp), lq >← next(part)
4:     accq ← lq
5:     for all < (a, p), q >∈ part do
6:       if a <> la then
7:         EMIT < (la, lp), accq >
8:         accq ← ()
9:       else if is_boundary(q, lq) then
10:        EMIT < (la, (p + lp)/2), accq >
11:        accq ← ()
12:      end if
13:      < (la, lp), lq >←< (a, p), q >
14:      accq ← accq + q
15:    end for
16:    index ← get_index(part)
17:    if index < npartitions(points) then
18:      < (a, p), q >← first(index + 1)
19:      if a <> la then
20:        EMIT < (la, lp), accq >
21:      else
22:        EMIT < (la, (p + lp)/2), accq >
23:      end if
24:    else
25:      EMIT < (la, lp), accq >
26:    end if
27:  end map
28: return(boundaries)
```

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

In Algorithm 1 (line 14), the bds variable is transformed by using a Map function, changing the previous key to a new key with a single value: the feature index $(< (att, (point, q)) >)$. This is done to group the tuples by feature so that we can divide them into two groups according to the total number of candidate points by feature. The *divide_attributes* function is then aimed to divide the tuples in two groups (big and $small$) depending on the number of candidate points by feature. Features in each group will be treated differently. The previous function performs this separation according to whether the total number of points by feature exceeds a threshold (mc) or not. The tuples are now re-formatted as follows: $(< point, q >)$.

MDLP evaluation

Features in each group are evaluated differently from we mentioned before. Small features are evaluated in a single step where each feature corresponds with a single parti-

For Peer Review

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

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

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

Input: $cands$ A RDD/array of tuples ($< point, q >$), where $point$ represents a candidate point to evaluate and q the class counter.
Input: mb Maximum number of intervals or bins to select
Input: mc Maximum number of candidates to eval in a partition
Output: An array of thresholds for a given feature

1: $st \leftarrow enqueue(st, (candidates, ()))$
2: $result \leftarrow ()$
3: **while** $|st| > 0$ & $|result| < mb$ **do**
4: $(set, lth) \leftarrow dequeue(st)$
5: **if** $|set| > 0$ **then**
6: **if** $type(set) = 'array'$ **then**
7: $bd \leftarrow arr\_select\_ths(set, lth)$
8: **else**
9: $bd \leftarrow rdd\_select\_ths(set, lth, mc)$
10: **end if**
11: **if** $bd <> ()$ **then**
12: $result \leftarrow result + bd$
13: $(left, right) \leftarrow divide(set, bd)$
14: $st \leftarrow enqueue(st, (left, bd))$
15: $st \leftarrow enqueue(st, (right, bd))$
16: **end if**
17: **end if**
18: **end while**
19: $return(sort(result))$

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

For Peer Review

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

Input: $cands$ An array of tuples ($< point, q >$), where $point$ represents a candidate point to evaluate and q the class counter.
Output: The minimum-entropy cut point

1: $total \leftarrow sum\_freqs(cands)$
2: $lacc \leftarrow ()$
3: **for** $< point, q > \in cands$ **do**
4: $\quad lacc \leftarrow lacc + q$
5: $\quad freqs \leftarrow freqs + (point, q, lacc, total - lacc)$
6: **end for**
7: $return(select\_best(cands, freqs))$

Algorithm 5 (rdd_select_ths) explains the selection process; in this case for "big" features (more than one partition). This process needs to be performed in a distributed manner since the number of candidate points exceeds the maximum size defined. For each feature, the subset of points is hence re-distributed in a better partition scheme to homogenize the quantity of points by partition and node ($coalesce$ function, line 1-2). After that, a new parallel function is started to compute the accumulated counter by partition. The results (by partition) are then aggregated to obtain the total accumulated frequency for the whole subset. In line 9, it is started a new distributed process with the aim of computing the accumulated frequencies by point from both sides (as explained in Algorithm 4). In this procedure, the process accumulates the counter from all previous partitions to the current one to obtain the first accumulated value (left one). Then the function computes the accumulated values for each inner point using the counter for points in the current partition, the left value and the total ones (line 7). Once these values are calculated ($< point, q, lq, rq >$), the algorithm evaluates all candidate points and their associated accumulators using the *select_best* function (as above).

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

Input: $cands$ An RDD of tuples ($< point, q >$), where $point$ represents a candidate point to evaluate and q the class counter.
Input: mc Maximum number of candidates to eval in a partition
Output: The minimum-entropy cut point

1: $npart \leftarrow round(|cands|/mc)$
2: $cands \leftarrow coalesce(cands, npart)$
3: $totalpart \leftarrow$
4: $\quad$ **map partitions** $partition \in cands$
5: $\quad\quad return(sum(partition))$
6: $\quad$ **end map**
7: $total \leftarrow sum(totalpart)$
8: $freqs \leftarrow$
9: $\quad$ **map partitions** $partition \in cands$
10: $\quad\quad index \leftarrow get\_index(partition)$
11: $\quad\quad ltotal \leftarrow ()$
12: $\quad\quad freqs \leftarrow ()$
13: $\quad\quad$ **for** $i = 0$ *until* $index$ **do**
14: $\quad\quad\quad ltotal \leftarrow ltotal + totalpart(i)$
15: $\quad\quad$ **end for**
16: $\quad\quad$ **for all** $< point, q > \in partition$ **do**
17: $\quad\quad\quad freqs \leftarrow freqs + (point, q, ltotal + q, total - ltotal)$
18: $\quad\quad$ **end for**
19: $\quad\quad return(freqs)$
20: $\quad$ **end map**
21: $return(select\_best(cands, freqs))$

For Peer Review

Algorithm 6 evaluates the discretization schemes yielded by each point by computing the entropy for each partition generated, also taking into account the MDLP criterion. Thus, for each point[4], it is calculated the entropy for the two generated partitions (line 8) as well as the total entropy for the whole set (lines 1-2). Using these values, the entropy gain for each point is computed and its MDLP condition, according to Equation 2. If the point is accepted by MDLP, the algorithm emits a tuple with the weighted entropy average of partition and the point itself. From the set of accepted points, the algorithm selects the one with the minimum class information entropy.

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

Input: $freqs$ An array/RDD of tuples ($< point, q, lq, rq >$), where point represents a candidate point to evaluate, leftq the left accumulated frequency, rightq the right accumulated frequency and q the class frequency counter.
Input: $total$ Class frequency counter for all the elements
Output: The minimum-entropy cut point

1: $n \leftarrow sum(total)$
2: $totalent \leftarrow ent(total, n)$
3: $k \leftarrow |total|$
4: $accp \leftarrow$
5: **map** $< point, q, lq, rq > \in freqs$
6: $k1 \leftarrow |lq|; k2 \leftarrow |rq|$
7: $s1 \leftarrow sum(lq); s2 \leftarrow sum(rq);$
8: $ent1 \leftarrow ent(s1, k1); ent2 \leftarrow ent(s2, k2)$
9: $partent \leftarrow (s1 * ent1 + s2 * ent2)/s$
10: $gain \leftarrow totalent - partent$
11: $delta \leftarrow log_2(3^k - 2) - (k * hs - k1 * ent1 - k2 * ent2)$
12: $accepted \leftarrow gain > ((log_2(s-1)) + delta)/n$
13: **if** $accepted = true$ **then**
14: $EMIT < partent, point >$
15: **end if**
16: **end map**
17: $return(min(accp))$

The results produced by both groups (Algorithm 1, line 19-22) are joined into the final point set of cut points.

EXPERIMENTAL FRAMEWORK AND ANALYSIS

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

Experimental Framework

Two huge classification datasets are employed as benchmarks in our experiments. The first one (hereinafter called *ECBDL14*) was used as a reference at the ML competition of the Evolutionary Computation for Big Data and Big Learning held on July 14,

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

For Peer Review

2014, under the international conference GECCO-2014. This consists of 631 characteristics (including both numerical and categorical attributes) and 32 million instances. It is a binary classification problem where the class distribution is highly imbalanced: 2% of positive instances. For this problem, the MapReduce version of the Random OverSampling (ROS) algorithm presented in [55] was applied in order to replicate the minority class instances from the original dataset until the number of instances for both classes was equalized. As a second dataset, we have used *epsilon*, which consists of 500 000 instances with 2000 numerical features. This dataset was artificially created for the Pascal Large Scale Learning Challenge in 2008. It was further pre-processed and included in the LibSVM dataset repository [17].

Table 2 gives a brief description of these datasets. For each one, the number of examples for training and test (#Train Ex., #Test Ex.), the total number of attributes (#Atts.), and the number of classes (#Cl) are shown. For our algorithm, we have established a maximum number of 50 intervals and a maximum number of candidates per partition of 100,000. For evaluation purposes, Naive Bayes [22] has been chosen as reference in classification, using the implementation included in MLlib library [48]. This uses multinomial version of Naive Bayes, typically used for document classification, which can handle all kinds of discrete data. The unique parameter λ for Naive Bayes is set to 1.

Table 2 Summary description for classification datasets

Data Set	**#Train Ex.**	**#Test Ex.**	**#Atts.**	**#Cl.**
epsilon	400 000	100 000	2000	2
ECBDL14 (ROS)	65 003 913	2 897 917	631	2

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

For all experiments we have used a cluster composed of twenty computing nodes and one master node. The computing nodes hold the following characteristics: 2 processors x Intel Xeon CPU E5-2620, 6 cores per processor, 2.00 GHz, 15 MB cache, QDR InfiniBand Network (40 Gbps), 2 TB HDD, 64 GB RAM. Regarding software, we have used the following configuration: Hadoop 2.5.0-cdh5.3.1 from Cloudera's open-source Apache Hadoop distribution[5], Apache Spark and MLlib 1.2.0, 480 cores (24 cores/node), 1040 RAM GB (52 GB/node). Spark implementation of the algorithm can

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

For Peer Review

be downloaded from the first author' GitHub repository[6]. The design of the algorithm has been adapted to be integrated in MLlib Library.

Experimental Results and Analysis

Table 3 shows the classification accuracy results for both datasets[7]. According to these results, we can assert that using our discretization algorithm as a preprocessing step leads to an improvement in classification accuracy, for the two datasets tested (both for training and test). It is specially important in ECBDL14 where there is a improvement of 5%.

Table 3 Classification accuracy values

	NB		**NB-disc**	
Dataset	*Train*	*Test*	*Train*	*Test*
ECBDL14	0.5260	0.6276	**0.6659**	**0.7260**
epsilon	0.6542	0.6550	**0.7094**	**0.7065**

Table 4 shows classification runtime values for both datasets distinguishing whether discretization is applied or not. As we can see, there is a slight improvement in both cases.

Table 4 Classification and discretization time values: with vs. w/o discretization -first two columns-, and sequential vs. distributed -last two columns- (in seconds)

Dataset	**NB**	**NB-disc**	**Sequential**	**Distributed**
ECBDL14	31.06	**26.39**	295 508	**1 087**
epsilon	5.72	**4.99**	5 764	**476**

Table 4 also shows discretization time values for two different versions of MDLP discretizer, namely, sequential and distributed. For the sequential version, we have measured the time employed by the original proposals on smaller samples for both datasets. The sequential time values shown on the table was estimated from the previous measurements. Comparing both implementations, we can notice the advantage of using the distributed version against the sequential one. For ECBDL14, our version obtains a speedup ratio of 271.86 whereas for epsilon the ratio is equal to 12.11. This demonstrates that the bigger the dataset, the higher the efficiency improvement.

[6]https://github.com/sramirez/SparkFeatureSelection
[7]In all tables, the best result by column (best by method) is highlighted in bold.

For Peer Review

Conclusion

Discretization, as an important part in DM preprocessing, has raised general interest in recent years. In this work, we have presented an updated taxonomy and description of the most relevant algorithms in this field. Although Big Data is currently a trending topic in science and business, no distributed approach have been developed in the literature, as we have shown in our taxonomy.
Here, we propose a completely distributed version of MDLP discretizer for large-scale processing. This version is capable of transforming the iterativity yielded by the original proposal in a single-step computation through a complete redesign of the original version. According to our experiments, our algorithm is capable to perform 270 times faster than the sequential version, improving the accuracy results in all used datasets.

Acknowledgments

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

References

[1] Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. In *Proceedings of the 20th Very Large Data Bases conference (VLDB)*, pages 487–499.

[2] Alcalá-Fdez, J., Sánchez, L., García, S., del Jesus, M. J., Ventura, S., Garrell, J. M., Otero, J., Romero, C., Bacardit, J., Rivas, V. M., Fernández, J. C., and Herrera, F. (2009). KEEL: a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3):307–318.

[3] Apache Hadoop Project (2015). Apache Hadoop. [Online; accessed March 2015].

[4] Apache Mahout Project (2015). Apache Mahout. [Online; accessed March 2015].

[5] Apache Spark: Lightning-fast cluster computing (2015). Apache spark. [Online; accessed March 2015].

[6] Au, W.-H., Chan, K. C. C., and Wong, A. K. C. (2006). A fuzzy approach to partitioning continuous attributes for classification. *IEEE Transactions on Knowledge Data Engineering*, 18(5):715–719.

[7] Augasta, M. G. and Kathirvalavakumar, T. (2012). A new discretization algorithm based on range coefficient of dispersion and skewness for neural networks classifier. *Applied Soft Computing*, 12(2):619–625.

[8] Bay, S. D. (2001). Multivariate discretization for set mining. *Knowledge Information Systems*, 3:491–512.

[9] Beyer, M. and Laney, D. (2001). 3d data management: Controlling data volume, velocity and variety. [Online; accessed March 2015].

[10] Bolón-Canedo, V., Sánchez-Maroño, N., and Alonso-Betanzos, A. (2011). Feature selection and classification in multiple class datasets: An application to KDD Cup 99 dataset. *Expert Syst. Appl.*, 38(5):5947–5957.

[11] Bolón-Canedo, V., Sánchez-Maroño, N., and Alonso-Betanzos, A. (2010). On the effectiveness of discretization on gene selection of microarray data. In *International Joint Conference on Neural Networks, IJCNN 2010, Barcelona, Spain, 18-23 July, 2010*, pages 1–8.

[12] Boulle, M. (2004). Khiops: A statistical discretization method of continuous attributes. *Machine Learning*, 55:53–69.

[13] Boullé, M. (2006). MODL: A bayes optimal discretization method for continuous attributes. *Machine Learning*, 65(1):131–165.

[14] Cano, A., Ventura, S., and Cios, K. J. (2014). Scalable CAIM discretization on multiple GPUs using concurrent kernels. *The Journal of Supercomputing*, 69(1):273–292.

[15] Catlett, J. (1991). On changing continuous attributes into ordered discrete attributes. In *European Working Session on Learning (EWSL)*, volume 482 of *Lecture Notes on Computer Science*, pages 164–178. Springer-Verlag.

[16] Cerquides, J. and Mantaras, R. L. D. (1997). Proposal and empirical comparison of a parallelizable distance-based discretization method. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 139–142.

[17] Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Datasets available at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

[18] Ching, J. Y., Wong, A. K. C., and Chan, K. C. C. (1995). Class-dependent discretization for inductive learning from continuous and mixed-mode data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:641–651.

For Peer Review

[19] Chmielewski, M. R. and Grzymala-Busse, J. W. (1996). Global discretization of continuous attributes as preprocessing for machine learning. *International Journal of Approximate Reasoning*, 15(4):319–331.

[20] Chou, P. A. (1991). Optimal partitioning for classification and regression trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:340–354.

[21] Dean, J. and Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. In *OSDI 2004*, pages 137–150.

[22] Duda, R. O. and Hart, P. E. (1973). *Pattern classification and scene analysis*, volume 3. Wiley New York.

[23] Fayyad, U. M. and Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1022–1029.

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

[25] Ferreira, A. J. and Figueiredo, M. A. T. (2012). An unsupervised approach to feature discretization and selection. *Pattern Recognition*, 45(9):3048–3060.

[26] Ferreira, A. J. and Figueiredo, M. A. T. (2014). Incremental filter and wrapper approaches for feature discretization. *Neurocomputing*, 123:60–74.

[27] Friedman, N. and Goldszmidt, M. (1996). Discretizing continuous attributes while learning bayesian networks. In *Proceedings of the 13th International Conference on Machine Learning (ICML)*, pages 157–165.

[28] García, S., Luengo, J., and Herrera, F. (2015). *Data Preprocessing in Data Mining*. Springer.

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

[30] Giráldez, R., Aguilar-Ruiz, J., Riquelme, J., Ferrer-Troyano, F., and Rodríguez-Baena, D. (2002). Discretization oriented to decision rules generation. In *Frontiers in Artificial Intelligence and Applications 82*, pages 275–279.

[31] González-Abril, L., Cuberos, F. J., Velasco, F., and Ortega, J. A. (2009). Ameva: An autonomous discretization algorithm. *Expert Systems with Applications*, 36:5327–5332.

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

For Peer Review

[33] Ho, K. M. and Scott, P. D. (1997). Zeta: A global method for discretization of continuous variables. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 191–194.

[34] Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–90.

[35] Hong, S. J. (1997). Use of contextual information for feature ranking and discretization. *IEEE Transactions on Knowledge and Data Engineering*, 9:718–730.

[36] Hu, H.-W., Chen, Y.-L., and Tang, K. (2009). A dynamic discretization approach for constructing decision trees with a continuous label. *IEEE Transactions on Knowledge and Data Engineering*, 21(11):1505–1514.

[37] Jiang, F. and Sui, Y. (2015). A novel approach for discretization of continuous attributes in rough set theory. *Knowledge-Based Systems*, 73:324–334.

[38] Jin, R., Breitbart, Y., and Muoh, C. (2009). Data discretization unification. *Knowledge and Information Systems*, 19:1–29.

[39] Kerber, R. (1992). Chimerge: Discretization of numeric attributes. In *National Conference on Artifical Intelligence American Association for Artificial Intelligence (AAAI)*, pages 123–128.

[40] Kurgan, L. A. and Cios, K. J. (2004). CAIM discretization algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 16(2):145–153.

[41] Kurtcephe, M. and Güvenir, H. A. (2013). A discretization method based on maximizing the area under receiver operating characteristic curve. *International Journal of Pattern Recognition and Artificial Intelligence*, 27(1).

[42] Lee, C.-H. (2007). A hellinger-based discretization method for numeric attributes in classification learning. *Knowledge-Based Systems*, 20:419–425.

[43] Li, M., Deng, S., Feng, S., and Fan, J. (2011). An effective discretization based on class-attribute coherence maximization. *Pattern Recognition Letters*, 32(15):1962–1973.

[44] Lin, J. (2012). Mapreduce is good enough? if all you have is a hammer, throw away everything that's not a nail! *CoRR*, abs/1209.2191.

[45] Liu, H., Hussain, F., Tan, C. L., and Dash, M. (2002). Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, 6(4):393–423.

[46] Liu, H. and Setiono, R. (1997). Feature selection via discretization. *IEEE Transactions on Knowledge and Data Engineering*, 9:642–645.

[47] Liu, X. and Wang, H. (2005). A discretization algorithm based on a heterogeneity criterion. *IEEE Transactions on Knowledge and Data Engineering*, 17:1166–1173.

For Peer Review

[48] Machine Learning Library (MLlib) for Spark (2015). Mllib. [Online; accessed March 2015].

[49] Mehta, S., Parthasarathy, S., and Yang, H. (2005). Toward unsupervised correlation preserving discretization. *IEEE Transactions on Knowledge and Data Engineering*, 17:1174–1185.

[50] Moskovitch, R. and Shahar, Y. (2015). Classification-driven temporal discretization of multivariate time series. *Data Mining and Knowledge Discovery. In press, DOI: 10.1007/s10618-014-0380-z*.

[51] Nguyen, H.-V., Müller, E., Vreeken, J., and Böhm, K. (2014). Unsupervised interaction-preserving discretization of multivariate data. *Data Mining and Knowledge Discovery*, 28(5-6):1366–1397.

[52] Pfahringer, B. (1995). Compression-based discretization of continuous attributes. In *Proceedings of the 12th International Conference on Machine Learning (ICML)*, pages 456–463.

[53] Quinlan, J. R. (1993). *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc.

[54] Ramírez-Gallego, S., García, S., Benítez, J. M., and Herrera, F. (2015). Multivariate discretization based on evolutionary cut points selection for classification. *IEEE Transactions on Cybernetics. In press, DOI: 10.1109/TCYB.2015.2410143*.

[55] Rio, S., Lopez, V., Benitez, J., and Herrera, F. (2014). On the use of mapreduce for imbalanced big data using random forest. *Information Sciences*, (285):112–137.

[56] Ruiz, F. J., Angulo, C., and Agell, N. (2008). IDD: A supervised interval Distance-Based method for discretization. *IEEE Transactions on Knowledge and Data Engineering*, 20(9):1230–1238.

[57] Sang, Y., Qi, H., Li, K., Jin, Y., Yan, D., and Gao, S. (2014). An effective discretization method for disposing high-dimensional data. *Information Sciences*, 270:73–91.

[58] Shannon, C. E. (2001). A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55.

[59] Shehzad, K. (2012). EDISC: A class-tailored discretization technique for rule-based classification. *IEEE Transactions on Knowledge Data Engineering*, 24(8):1435–1447.

[60] Su, C.-T. and Hsu, J.-H. (2005). An extended chi2 algorithm for discretization of real value attributes. *IEEE Transactions on Knowledge and Data Engineering*, 17:437–441.

For Peer Review

[61] Tay, F. E. H. and Shen, L. (2002). A modified chi2 algorithm for discretization. *IEEE Transactions on Knowledge and Data Engineering*, 14:666–670.

[62] Tsai, C.-J., Lee, C.-I., and Yang, W.-P. (2008). A discretization algorithm based on class-attribute contingency coefficient. *Information Sciences*, 178:714–731.

[63] White, T. (2012). *Hadoop, The Definitive Guide*. O'Reilly Media, Inc.

[64] Wong, A. K. C. and Chiu, D. K. Y. (1987). Synthesizing statistical knowledge from incomplete mixed-mode data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9:796–805.

[65] Wu, Q., Bell, D. A., Prasad, G., and McGinnity, T. M. (2007). A distribution-index-based discretizer for decision-making with symbolic ai approaches. *IEEE Transactions on Knowledge and Data Engineering*, 19:17–28.

[66] Wu, X. (1996). A bayesian discretizer for real-valued attributes. *The Computer Journal*, 39:688–691.

[67] Wu, X. and Kumar, V., editors (2009). *The Top Ten Algorithms in Data Mining*. Chapman & Hall/CRC Data Mining and Knowledge Discovery.

[68] Yan, D., Liu, D., and Sang, Y. (2014). A new approach for discretizing continuous attributes in learning systems. *Neurocomputing*, 133:507–511.

[69] Yang, Y. and Webb, G. I. (2009). Discretization for naive-bayes learning: managing discretization bias and variance. *Machine Learning*, 74(1):39–74.

[70] Yang, Y., Webb, G. I., and Wu, X. (2010). Discretization methods. In *Data Mining and Knowledge Discovery Handbook*, pages 101–116.

[71] Zhang, Y. and Cheung, Y.-M. (2014). Discretizing numerical attributes in decision tree for big data analysis. In *ICDM Workshops*, pages 1150–1157.

[72] Zighed, D. A., Rabaséda, S., and Rakotomalala, R. (1998). FUSINTER: a method for discretization of continuous attributes. *International Journal of Uncertainty, Fuzziness Knowledge-Based Systems*, 6:307–326.

For Peer Review