Data & Knowledge Engineering 112 (2017) 94–105

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Training set selection for monotonic ordinal classification

J.-R. Cano[a,*], S. García[b]

[a] Dept. of Computer Science, University of Jaén, EPS of Linares, Avenida de la Universidad S/N, Linares 23700, Jaén, Spain
[b] Department of Computer Science and Artificial Intelligence, University of Granada, 18071, Granada, Spain

ARTICLE INFO

Keywords:
Monotonic classification
Ordinal classification
Training set selection
Data preprocessing
Machine learning

ABSTRACT

In recent years, monotonic ordinal classification has increased the focus of attention for machine learning community. Real life problems frequently have monotonicity constraints. Many of the monotonic classifiers require that the input data sets satisfy the monotonicity relationships between its samples. To address this, a conventional strategy consists of relabeling the input data to achieve complete monotonicity. As an alternative, we explore the use of preprocessing algorithms without modifying the class label of the input data.

In this paper we propose the use of training set selection to choose the most effective instances which lead the monotonic classifiers to obtain more accurate and efficient models, fulfilling the monotonic constraints. To show the benefits of our proposed training set selection algorithm, called MonTSS, we carry out an experimentation over 30 data sets related to ordinal classification problems.

1. Introduction

Learning with ordinal data sets has increased the attention of the machine learning community in recent years [1,2]. These data sets are characterized by the presence of an ordinal output and they are commonly found in real life [3].

Monotonic classification is an ordinal classification problem where monotonic constraints are present in the sense that a higher value of a feature in an instance, fixing the other values, should not decrease its class assignment [4–6]. As an example, we can contemplate the problem of ranking documents according to the relevance to a certain query. They can be categorized into relevant, moderately relevant and non-relevant. Considering a usual feature of count of matching terms with the query, it is expected that an increase in this feature would influence the final relevance of the document. These relationships between the inputs features and the response are termed as monotonic.

In the specialized literature we can find multiple monotonic classifiers proposed, like neural networks and support vector machines [7,8], classification trees and rule induction [9–16] and instance-based learning [17–21]. As a restriction, some of them require the training set to be purely monotone to work properly. Other classifiers can handle non-monotonic data sets, but they do not guarantee monotone predictions.

In addition, real-life data sets are likely to have noise, which obscures the relationship between features and the class [22]. This fact affects the prediction capabilities of the learning algorithms which learn models from those data sets.

In order to address these shortcomings and to test the prediction competences of the monotonic classifiers, the usual trend is to generate data sets which completely satisfy the monotonicity conditions [23]. The intuitive idea behind this is that the models trained on monotonic data sets should offer better predictive performance than the models trained on the original data. In the specialized literature, we find two possible techniques to generate monotonic data sets. Monotonic data sets can be created by

* Corresponding author.
E-mail addresses: jrcano@ujaen.es (J.-R. Cano), salvagl@decsai.ugr.es (S. García).

http://dx.doi.org/10.1016/j.datak.2017.10.003
Received 4 December 2016; Received in revised form 14 September 2017; Accepted 14 October 2017
Available online 16 October 2017

generating artificial data [24] and by relabeling the real data [19,23,25]. The latter restores the monotonicity of the data set by changing the class labels in those instances which violate the monotonicity constraints. Class relabeling is the only approach which can be applied in real life data sets, and has shown promising results in the literature [26].

As an alternative to relabel, Training Set Selection (TSS) is known as an application of instance selection methods [27–29] over the training set used to build any predictive model. Thus, TSS can be employed as a way to improve the behavior of predictive models, precision and interpretability [30–34].

In this paper we propose a TSS algorithm to manage monotonic classification problems, called Monotonic Training Set Selection (MonTSS). It is a data preprocessing technique which, by means of a suitable TSS process for monotonic domains, offers an alternative without modifying the class labels of the data set, it instead removes harmful instances. MonTSS incorporates proper measurements to identify and select the most suitable instances in the training set to enhance both the accuracy and the monotonic nature of the models produced by different classifiers. We have compared the results offered by well-known classical monotonic classifiers over 30 data sets with and without the use of MonTSS as a data preprocessing stage. The results show that MonTSS is able to select the most representative instances, which leads monotonic classifiers to always offer equal or better results than without preprocessing.

The paper is organized as follows. In Section 2, the monotonic classification problem is presented, together with some state-of-the-art monotonic classifiers. Section 3 describes the proposed MonTSS algorithm. Section 4 describes the experimental framework, with respect to data sets, parameters and quality metrics considered. In Section 5 the results and analysis are included. Finally, Section 6 concludes the paper.

2. The monotonic classification problem

In this section, we define the monotonic classification domain and describe some classical algorithms proposed in this field.

2.1. Problem definition

Monotonicity is a property commonly found in many environments of our lives like economics, natural language or game theory [4]. A classical example of monotonicity is in the case of bankruptcy prediction in companies, where appropriate actions can be taken in time, considering the information based on financial indicators taken from their annual reports. The comparison of two companies where one dominates the other on all financial indicators shows clearly where the monotonicity is present, which supposes that the overall evaluation of the second cannot be higher than the evaluation of the first. This strategy could be applied to the credit rating score used by banks [8] as well as for the bankruptcy prediction strategy [35].

To define monotonic ordinal data, let D be a data set with f ordinal features $A_1, \ldots, A_f$ and one output class feature Y having c possible ordinal values ($\{y_1, \ldots, y_c\} \in Y$). The data set consists of n instances $x_1, \ldots, x_n$ ($(x_i, y_i) \in D$). A partial ordering $\preceq$ on D is defined as

$$x \preceq x' \Leftrightarrow A_j(x) \leq A_j(x'), \forall\, j = 1, \ldots, f. \tag{1}$$

Two instances x and x' in space D are *comparable* if either $x \preceq x'$ or $x' \preceq x$, otherwise x and x' are *incomparable*. Two instances x and x' are *identical* if $x = x'$ and *non-identical* if $x \neq x'$;.

Considering this notation, we denote a pair of comparable instances (x, x') *monotone* if

$$x \preceq x' \wedge x \neq x' \wedge Y(x) \leq Y(x'), \tag{2}$$

or

$$x = x' \wedge Y(x) = Y(x'). \tag{3}$$

With this definition in mind, we can designate a data set D with n instances as monotone if all possible pairs of instances are either monotone or incomparable.

Monotonic classification is usually used for well-known problems where it is known a priori that there is a monotone relationship between features and class labels. It may be the case that not all features present monotonicity relationship, in which case they would not be taken into account in the construction of the classifier. The definition of monotonicity in Eq. (1) refers to all features which present monotonic relation with the class. For this reason it is noted as a partial order ($\preceq$).

However, it could be the case that a data set must be used for which we do not have sufficient a priori information to know the relationship between features and class. To address this situation we consider the analysis of the relationship between each feature and the class, using the rank mutual information metric (RMI [36]). RMI indicates, with a positive value, that the relationship (between feature and class) is direct and negative if it is inverse (when feature grows, class decreases). Zero means that there is no relation between feature and class, and then the feature is not used. If the relation of order between features and class is direct, the expression considered is that used in Eq. (2). In case of the existence of some inverse relations, Eq. (2) could be generalized as follows:

$$x \preceq x' \wedge Y(x) \leq Y(x'), \quad \forall\, j = 1, \ldots, f. \begin{cases} A_j(x) \leq A_j(x') & \textbf{if } \mathbf{RMI}(A_j(x), Y) > 0 \\ A_j(x) \geq A_j(x') & \textbf{if } \mathbf{RMI}(A_j(x), Y) < 0 \end{cases} \tag{4}$$

Those cases are considered in our proposal, and the rank mutual information metric definition is extended in Section 3.3.

2.2. Classic monotonic classifiers

The classical monotonic classification algorithms considered in this study are:

- Ordinal Learning Model (OLM [17]). OLM was the first algorithm proposed for ordinal classification with monotonic restrictions. The aim is to select a subset $D' \subseteq D$ of training instances where all of items satisfy the monotonicity constraints. To classify a new instance, the following function must be used:

$$f_{OLM}(x) = max\{y_i: x_i \in D', x_i \preceq x\}. \quad (5)$$

  If x does not dominate any object in D', then the class label predicted is selected by the nearest neighbor rule. D' must be consistent and without redundant instances. One instance x_i is redundant when there is any x_j where $x_i \succcurlyeq x_j$ and $y_i = y_j$.
- Ordinal Stochastic Dominance Learner (OSDL [18,37]): It is an instance-based method based on stochastic order consisting of two mapping functions. The first one is based on instances that are stochastically dominated by x_i with the maximum label, and the second one is based on instances that cover x_i with the smallest label.
- Monotone Induction of Decision trees (MID [9]): MID is based on the idea of modifying the conditional entropy in ID3 algorithm [38], including a term called *order-ambiguity-score*. The aim is to adapt the splitting strategy towards the building of monotonic trees. This algorithm does not guarantee that the trees generated result in a completely monotone function.
- Monotonic k-Nearest Neighbor Classifier (MkNN [19]): This algorithm considers the classical nearest neighbor rule ([39]), for a new instance x_0 the prediction of its class label must be in the range $[y_{min}, y_{max}]$ to preserve the monotonicity constraints, where

$$y_{max} = min\{y|(x, y) \in D \wedge x_0 \preceq x\}, \quad (6)$$

  in other words, given a new instance x_0 to be classified, its associated y_{max} will be determined as the minimum class value of all those instances (x) in the data set whose attribute values are all bigger than, or equal to, those of x_0.

$$y_{min} = max\{y|(x, y) \in D \wedge x \preceq x_0\}. \quad (7)$$

  In this case, given a new instance x_0 to be classified, its associated y_{min} will be determined as the maximum class value of all those instances (x) in the data set whose attributes values are all smaller than, or equal to, those of x_0. The predictions are carried out according to majority voting of the k nearest neighbors of x_0 from D whose labels are included in $[y_{min}, y_{max}]$. As [19] indicates, a previous relabeling stage is necessary before the Monotonic k-Nearest Neighbor is applied. In [19] MkNN incorporates as data preprocessing an optimal relabeling procedure which changes the labels of pairs of instances that violate the monotonic constraints, using the minimum number of alterations.

3. Monotonic Training Set Selection

In this section, we describe the proposed algorithm MonTSS (Monotonic Training Set Selection). MonTSS can be considered as the first in the literature for performing TSS in monotonic classification problems. As we mentioned, in TSS the aim is to reduce the size of the training data set by selecting the most representative instances. The effects produced by such data preprocessing, according to [33,40,41], are: reduction in space complexity, decrease in computational cost and the selection of the most representative instances by discarding noisy ones.

In monotonic classification, besides the previously enumerated advantages, the most representative instances must also keep or improve the monotonicity constraints of the training set. In this manner, the learned models from the data sets resulting from the TSS process are expected to improve their monotonic condition with respect to those obtained from the original training data set.

Next, we introduce the required concepts in Sections 3.1 and 3.2. Section 3.3 summarizes the description of the MonTSS algorithm.

3.1. Probabilistic Collision Removal

Two instances produce a collision if they do not satisfy the Eqs. (2) and (3). Before carrying out the TSS, we apply an ordered probabilistic removal based on the collisions produced by these instances. The process is stochastic to avoid falling in local optima. For this reason, two parameters are introduced: *Candidates*, which points out the rate of the best candidates to be selected, and *CollisionsAllowed*, which represents the minimal rate of collisions permitted to stop the removal process.

The probabilistic algorithm is given in Algorithm 1: it first calculates the number of collisions that each instance produces (line 3), *maxCandidates* (line 4) based on the size of the data set and the candidate rate used as a parameter (*Candidates*) and the minimum number of collisions to stop the iterative process (*minCol* in line 5); in every iteration, it sorts the instances in decreasing order of number of collisions (line 8); in line 9, is assigned to variable *candSelected* a random value generated between 1 and *maxCandidates*. This value will be used to decide which element to remove from the ordered list S (line 10), with the idea of eliminating one of the instances with maximum number of collisions in the range [1,maxCandidates]; finally, the collisions are recalculated and this process is repeated until *CollisionsAllowed* is reached (lines 11-18). The efficiency of the algorithm is $O(n^2)$, where n is the number of instances in the training set T.

Algorithm 1. Probabilistic Collision Removal algorithm.

```
1:  function ProbColRemoval(T - training data, Candidates - candidate rate, CollisionsAllowed - minimal rate of collisions allowed)
2:    initialize: totalCollisions=0,S=T
3:    [Col[], ColMatrix[][], totalCollisions]=Calculate_Collisions(S)
4:    maxCandidates=# S·Candidates
5:    minCol = totalCollisions·CollisionsAllowed
6:    newCol = totalCollisions
7:    repeat
8:      S=sort(S,Col[])
9:      candSelected=Rand(1, maxCandidates)
10:     S=S \ {x_candSelected}
11:     newCol=newCol - Col[candSelected]
12:     Col[candSelected] = 0
13:     for all x_j ∈ T do
14:       if (ColMatrix[j][candSelected] = true ) then
15:         ColMatrix[j][candSelected] = false
16:         Col[j] = Col[j] − 1
17:       end if
18:     end for
19:   until newCol < minCol
20:   return S
21: end function
22: function Calculate_Collisions(T - training data)
23:   for all x_i ∈ T do
24:     Col[i]=0
25:     for all x_j ∈ T do
26:       ColMatrix[i][j]=0
27:     end for
28:   end for
29:   for all x_i ∈ T do
30:     for all x_j ∈ T do
31:       if (i ≠ j ) then
32:         if (x_i ≼ x_j and Y(x_i) > Y(x_j)) or (x_i = x_j and Y(x_i) ≠ Y(x_j)) then
33:           Col[i]=Col[i] + 1
34:           Col[j]=Col[j] + 1
35:           ColMatrix[i][j] = true
36:           ColMatrix[j][i] = true
37:         end if
38:       end if
39:     end for
40:   end for
41:   totalCollisions = 0
42:   for all x_i ∈ T do
43:     totalCollisions = totalCollisions + Col[i]
44:   end for
45:   return [Col[], ColMatrix[][], totalCollisions]
46: end function
```

The more collisions an instance has, the higher is its probability to be removed. After this process, the resulting data is formed by instances with the allowed number of collisions. Although taking this process to the extreme could obtain collisions-free data, it is desirable to provide greater freedom of action to the quality metrics described next.

3.2. Quality metrics selection

We introduce two new metrics: *Delimitation* (Del) and *Influence* (Infl), to assess how representational the instances are. Both are calculated for each instance resulting from the previous stage and are used to determine which instances must be included in the final TSS.

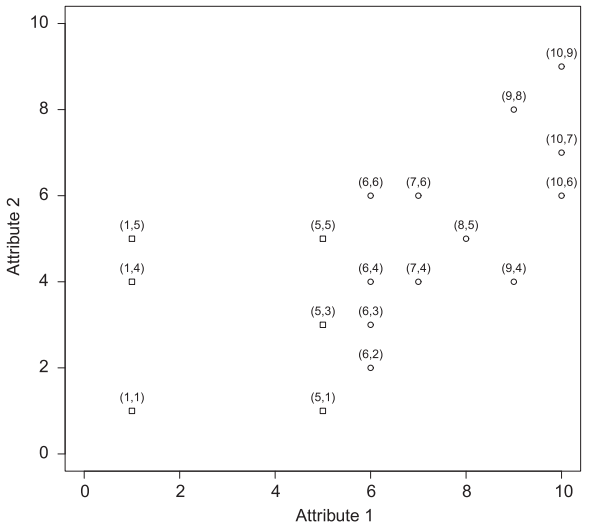

Fig. 1. Data set example for Delimitation Metric.

The instances located at the boundary of decision classes are of great importance in monotonic classification. They should be conserved to guarantee that the relationship of monotonicity between the inputs and the response remains constant. Hence, both metrics have been proposed to identify the subset of most important instances which belong to the decision boundaries. They are defined for an instance $x_i \in D$ as follows:

- *Delimitation* (Del):

$$\mathrm{Del}(x_i) = \frac{|\mathrm{Dom}(x_i) - \mathrm{NoDom}(x_i)|}{\mathrm{Dom}(x_i) + \mathrm{NoDom}(x_i)}, \tag{8}$$

$$\mathrm{Dom}(x_i) = \#X',\ x' \in X' \Leftrightarrow x_i \prec x' \wedge Y(x_i) = Y(x'), \tag{9}$$

$$\mathrm{NoDom}(x_i) = \#Z',\ \ x' \in Z' \Leftrightarrow\ x_i \succcurlyeq x'\ \wedge\ Y(x_i) = Y(x'), \tag{10}$$

  where $\mathrm{Dom}(x_i)$ represents the number of instances of the same class as x_i that dominate x_i, and $\mathrm{NoDom}(x_i)$ is the number of instances of the same class as x_i dominated by x_i. The range of $\mathrm{Del}(x_i)$ is $[0,\ 1)$. A value closer to 0 means that the instance is located near the central region of the class and the value near to 1 means that it is closer to the boundaries. As an example of its use we consider the data set which appears in Fig. 1. The figure represents a data set with two classes (class 1 as squared form and class 2 as circles), where Attribute 1 and Attribute 2 are inputs of the instances. Here, each example is denoted by its coordinates in the format (Attribute 1,Attribute 2). Calculating Delimitation for each one of the square class points produce the results present in Table 1. The points with higher values would be the selected ones. Considering a threshold of 0.9 for example (as in the experimental section), the selected points are (1,1) and (5,5). The rest of the instances would not be retained using this metric. In fact, these two selected instances can perfectly determine a safe region occupied by the square class.
- *Influence* (Infl). The *Influence* of each instance x_i is computed by using the number of neighbors of any different class label and their distance to x_i. First, the k nearest neighbors are obtained and their corresponding distance to x_i is computed. Each neighbor has an associated weight (nWeight), which depends on its distance to the instance x_i, normalized by the sum of the distances of all the neighbors (see Eq. (12)). nWeight is greater as closer the neighbor is to the instance x_i. The value of nWeight for each neighbor

Table 1
Example of calculation of *Delimitation* metric for Class Square (Class 1).

(1,1) ⇒ Dom ((1,1))=5, NoDom ((1,1))=0 ⟹ Del((1,1))=$\frac{\lvert 5-0\rvert}{5}=\frac{5}{5}$=1
(1,4) ⇒ Dom ((1,4))=2, NoDom ((1,4))=1 ⟹ Del((1,4))=$\frac{\lvert 2-1\rvert}{3}=\frac{1}{3}$=0.333
(1,5) ⇒ Dom ((1,5))=1, NoDom ((1,5))=2 ⟹ Del((1,5))=$\frac{\lvert 1-2\rvert}{3}=\frac{1}{3}$=0.333
(5,1) ⇒ Dom ((5,1))=2, NoDom ((5,1))=1 ⟹ Del((5,1))=$\frac{\lvert 2-1\rvert}{3}=\frac{1}{3}$=0.333
(5,3) ⇒ Dom ((5,3))=1, NoDom ((5,3))=2 ⟹ Del((5,3))=$\frac{\lvert 1-2\rvert}{3}=\frac{1}{3}$=0.333
(5,5) ⇒ Dom ((5,5))=0, NoDom((5,5))=5 ⟹ Del((5,5))=$\frac{\lvert 0-5\rvert}{5}=\frac{5}{5}$=1

is normalized using the Eq. (13) so the *Influence* metric is situated in range [0,1]. The *Influence* value for the instance x_i is calculated by considering only the neighbors of different class than x_i among its k nearest neighbors (see Eq. (14)):

$$kNN_{x_i} = k\text{nearest neighbors of} x_i \tag{11}$$

$$\text{nWeight}(x_j) = \frac{\sum_{l=1}^{k} \text{Distance}(x_i, x_l) - \text{Distance}(x_i, x_j)}{\sum_{l=1}^{k} \text{Distance}(x_i, x_l)}, \quad \forall\, x_j \in kNN_{x_i}, \tag{12}$$

$$\text{influenceWeight}(x_j) = \frac{\text{nWeight}(x_j)}{\sum_{l=1}^{k} \text{nWeight}(x_l)}, \forall\, x_j \in kNN_{x_i}, \tag{13}$$

$$\text{Infl}(x_i) = \sum_{j=1}^{k} \text{influenceWeight}(x_j), \quad \text{where } Y(x_i) \neq Y(x_j) \wedge x_j \in kNN_{x_i}, \tag{14}$$

The value 0 reflects that the instance is surrounded by instances of the same class, showing a high degree of redundancy. The value 1 represents that it is surrounded by neighbor instances of other classes, which means that it is an instance that could introduce separation between classes.

As an example of *Influence* evaluation we consider two points of the data set which appear in Fig. 1. One point in the outer zone of the square class (calculation for point (1,4) in Table 2) and another near the instances of a neighboring class (calculation for point (5,3) in Table 3). *Influence* is able to associate a value in the range [0,1] to every instance according to its neighbors of different class and their distance. One instance whose neighbors belong to its same class present a *Influence* value of 0. This *Influence* value increases as the number of neighbors of different class grows and they are nearer to the reference instance. *Influence* determines which instances delimit class boundaries based on their neighbors of enemy classes.

3.3. The MonTSS algorithm

The whole process is presented in Fig. 2, and as can be seen it is composed of three stages:

1. The MonTSS process starts with a preprocessing step where MonTSS analyzes the original data set by quantifying the relationship between each input feature and the output class. This relation is estimated with a metric called Rank Mutual Information (RMI) defined in [36]. With it, we know the features which have a real direct or inverse monotonic relation with the class or no relation as well (including unordered categorical features). The RMI value is evaluated in the training data set to decide which features are used in the computation of collisions between instances.

   In essence, rank mutual information can be considered as the degree of monotonicity between features $A_1, \ldots, A_f$ and the feature class Y. Given any feature A_j and feature class Y, the value of RMI for the feature A_j is calculated as follows:

$$\text{RMI}(A_j, Y) = -\frac{1}{n}\sum_{i=1}^{n} log\frac{\#[x_i]^{\leq}_{A_j} \cdot \#[x_i]^{\leq}_{Y}}{n \cdot \#([x_i]^{\leq}_{A_j} \cap [x_i]^{\leq}_{Y})} \tag{15}$$

   where n is the number of instances in data set D, $[x_i]^{\leq}_{A_j}$ is the set formed by all the instances of the set D whose feature A_j is less or equal than feature A_j of instance x_i, and $[x_i]^{\leq}_{Y}$ is the set composed of the instances of the set D whose feature class Y is less or equal than feature class Y of instance x_i.
2. In the second stage, the probabilistic collision removal mechanism is applied (see Section 3.1), which eliminates most of the instances which produce collisions. The remaining instances are used as input in the last stage.

Table 2
Example of calculation of *Influence* metric for point (1,4).

Considering k=5, $kNN_{(1,4)}$={(1,5), (1,1), (5,5), (5,3), (5,1)}
nWeight((1,5))=0.942 ⟹ influenceWeight((1,5))=0.235
nWeight((1,1))=0.825 ⟹ influenceWeight((1,1))=0.206
nWeight((5,5))=0.760 ⟹ influenceWeight((5,5))=0.190
nWeight((5,3))=0.760 ⟹ influenceWeight((5,3))=0.190
nWeight((5,1))=0.710 ⟹ influenceWeight((5,1))=0.177
Infl(1,4)=0 (No neighbor with different class than (1,4) exists

Table 3
Example of calculation of *Influence* metric for point (5,3).

Considering k=5, $kNN_{(5,3)}$={(5,5), (5,1), (6,4), (6,3), (6,2)}
nWeight((5,5))=0.744 ⟹ influenceWeight((5,5))=0.186
nWeight((5,1))=0.744 ⟹ influenceWeight((5,1))=0.186
nWeight((6,4))=0.819 ⟹ influenceWeight((6,4))=0.204
nWeight((6,3))=0.872 ⟹ influenceWeight((6,3))=0.218
nWeight((6,2))=0.819 ⟹ influenceWeight((6,2))=0.204
Infl(5,3)=influenceWeight((6,4)) + influenceWeight((6,3)) + influenceWeight((6,2))=0.626

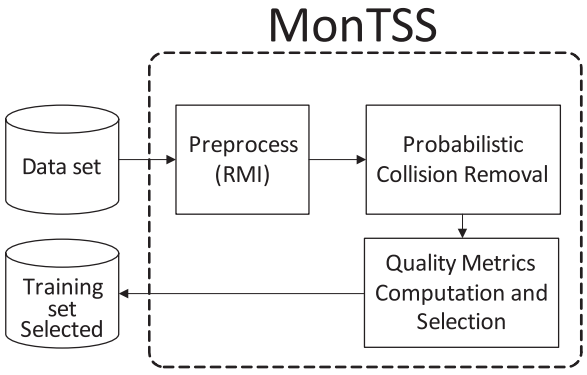

Fig. 2. MonTSS process.

3. Here, the quality metrics are computed and based on them, the selection procedure is developed considering the following rule:

$$\text{Select}x_i = \begin{cases} \text{true} & \textbf{if } \text{Del}(x_i) < \text{Infl}(x_i) \\ & \text{or } \text{Del}(x_i) \geq 0.9 \\ \text{false} & \text{otherwise.} \end{cases} \quad (16)$$

The rationale behind this rule is to retain the instances which are closer to the class boundaries, using a straightforward threshold of 0.9 which is independent from the diversity of their neighborhood. Furthermore, a relationship between Del (x_i) and Infl (x_i) can be easily established as they represent a measurement in the same range of the relative rate of the situation and the neighborhood variety of every instance. In this respect, the rule is built as a function of both measures. As a result, for instance, the rule preserves the instances belonging to central areas if there are instances of other classes around.

Considering the example presented in Fig. 1 of the previous subsection, some of the points selected based in the expression 16 would be (1,1) and (5,5) due to Del $(x_i) \geq 0.9$, and (5,3) related to $\text{Del}(x_i) < \text{Infl}(x_i)$. The point (1,4) is not selected (see Table 2) by either $\text{Del}(x_i) \geq 0.9$ or $\text{Del}(x_i) < \text{Infl}(x_i)$. As can be noted, $\text{Del}(x_i) \geq 0.9$ rule retains instances situated in the border of classes while $\text{Del}(x_i) < \text{Infl}(x_i)$ maintains those whose neighbors belongs to other classes and are near to the boundaries of the class.

4. Experimental framework

In this section we introduce the data sets used to test our proposed algorithm, the parameters fixed in the algorithms and the performance metrics considered to evaluate the results.

4.1. Data sets

The data sets have been collected from the KEEL-data set[1] [42] and UCI Repository [43]. They are split into two blocks. In the first one, we enumerate 20 standard classification data sets which can be addressed as ordinal classification problems where a monotonic relationship between features and class exists. The second block is composed of 10 regression data sets whose class feature is discretized into 4 categorical values, keeping the class distribution balanced. The algorithms are evaluated using a 10-fold cross validation schema (10-fcv). To evaluate data preprocessing methods [33], the original data set is divided into 10 folds, using each time one fold without preprocessing as test and the remaining 9 as training for MonTSS, hence MonTSS performs the selection considering only training data. In every one of the 10 evaluations, the test fold and their complementary train, are different.

Table 5 shows the name of the data sets.

[1] http://www.keel.es/datasets.php

Table 4
Parameters for the algorithms considered.

Algorithm	Parameters
MonTSS	Candidates = 0.01, CollisionsAllowed = 0.01, k for Influence estimation = 5, distance=euclidean
MkNN	distance=euclidean, k = 3
OLM	modeClassification=conservative modeResolution=conservative
MID	C4.5 as base classifier 2 items per leaf, confidence = 0.25, R = 1
OSDL	balanced=No, classificationType=media tuneInterpolationParameter=No, weighted=No, upperBound=1,lowerBound = 0 interpolationParameter = 0.5, interpolationStepSize = 10

4.2. Parameters

The parameters of the algorithms appear in Table 4 and are fixed for all the data sets. The values for those parameters have been selected according to the recommendations of the authors of each method. The parameter values of our proposal have been empirically determined to achieve a proper balance between reduction and prediction capabilities.

Both for MkNN an MonTSS algorithms, the euclidean distance is adopted to measure the similarity in numerical and categorical ordinal features.

4.3. Performance metrics

To analyze the results offered by the monotonic classification algorithms, with and without data preprocessing, we have considered the following Mean Absolute Error and data related metrics:

- Mean Absolute Error (MAE): This measure intends to evaluate the prediction capabilities of the algorithms. It is defined as the sum of the absolute values of the errors, divided by the number of classifications. The literature in various studies concludes that MAE is one of the best performance metrics in ordered classification [44,45].
- Data related metrics: In this case, the aim is to assess the monotonicity of the training data sets offered by the preprocessing proposal.
  - Non-Monotonicity Index (NMI [46]), calculated as the number of non-monotone instances divided by the total number of instances:

$$\text{NMI} = \frac{1}{n}\sum_{x\in D}\text{Collision}(x) \tag{17}$$

    where $\text{Collision}(x) = 0$ if x does not collide with any instances in D, and 1 otherwise.
  - Non-Comparable, defined as the number of pairs of non comparable instances in the data set. Two instances x and x' are non-comparable if they do no satisfy $x \preceq x' \wedge x \neq x'$. The reason for considering it as a metric is based on the fact that it is harder to build accurate models as the number of non-comparable pairs increases.
  - Size of the training set selected using the preprocessing algorithm proposed. It is included to analyze the reduction capabilities of the method.

To complete the analysis we have included as preprocessing an optimal relabeling procedure described in [19] which changes the labels of pairs of instances that violate the monotonic constraints, using the minimum number of alterations. This relabeling method is specifically designed to be used in combination with instance based learners like MkNN. In addition, we have introduced the Fig. 3 to analyze the effect of optimal relabeling and the proposed method in one artificial data set (noted Artiset and generated as [6] indicates) composed by 1000 instances with 2 attributes, 5 classes and with 5% of non-monotonic instances. The first subfigure shows the projection of original instances, the second one the outcome of Relabeling and the third one, the instances selected by MonTSS. The area is coloured according to the decision frontiers modeled by the MkNN algorithm considering 3 neighbors.

5. Results and analysis

The average results of the 10-fcv evaluations are shown in Table 5. In the first column of this table, we present the name of the data set. The second column contains the results corresponding to the combination of Relabeling+MkNN. In the third column we

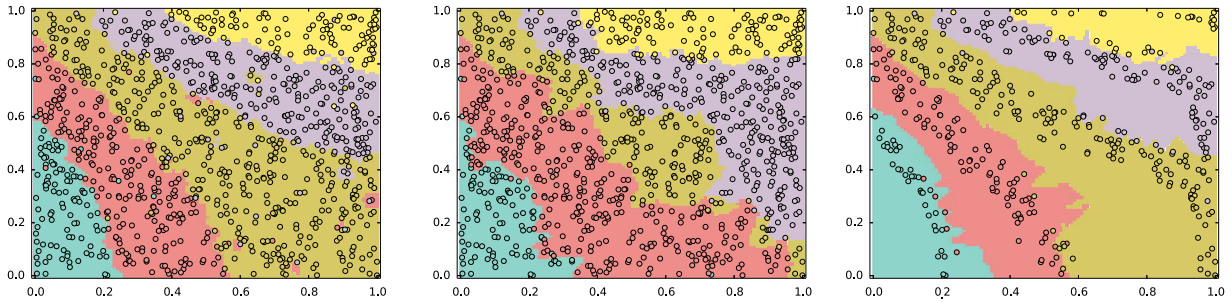

Fig. 3. Artificial data set preprocessed by Relabeling and MonTSS with the borders calculated by M*k*NN with 3 neighbors.

present the results associated with the evaluation of the original data set directly by the monotonic classifiers considered. The fourth column is devoted to the combination of MonTSS and monotonic classifiers. Fifth and sixth columns summarize the monotonicity indexes obtained using the original data sets and after the use of MonTSS. These metrics are NMI, Non-Comparable and Size. The last row in the table contains the average results over the 30 data sets evaluated. The best values for each row have been highlighted in bold.

To strengthen the analysis, we have included the statistical outcome based on non-parametric procedures to compare the performances offered by our proposal with respect to the original classifiers without preprocessing. For this, the Wilcoxon signed-rank test has been used to conduct pairwise comparisons between two algorithms [47]. The Wilcoxon test involves ranking all nonzero difference scores disregarding sign, reattaching the sign to the rank, and then evaluating the mean of the positive and the mean of the negative ranks. The Wilcoxon test was specially recommended by Demsar in [48] to make performance comparisons of pairs of algorithms on multiple data sets.

Observing Tables 5 and 6 and Fig. 3, we come up with the following analysis:

- In most of the cases, the prediction capabilities of the classifiers are improved when the TSS is applied. This conclusion is confirmed when statistical tests are applied, as Table 6 reflects. Using the Wilcoxon test we compare each classifier with its preprocessed version and the table indicates to us that MonTSS improves M*k*NN with and without relabeling, OLM and OSDL. MID is robust to noise and its predictive performances are the same with or without TSS, indicating that MonTSS does not change its behavior. The greatest improvement noted is the combination of MonTSS with M*k*NN, achieving similar performances to MID.
- Regarding data related metrics, in Table 5 it must be mentioned that the monotonic models extracted using our proposal are those which best satisfy the monotonicity constraints (with NMI near to 0). In addition, MonTSS significantly reduces the number of non-comparable pairs of instances and the size of the training data set. This occurs in all data sets evaluated.
- It is worth mentioning that relabeling clearly improves standard M*k*NN, a fact that was suggested in [19]. Nevertheless, MonTSS is able to improve even more the performance of M*k*NN, becoming a competitive algorithm.
- In Fig. 3, the behavior of relabeling and MonTSS is reflected. In Fig. 3b, it is clear how the relabeling modifies the class of those non-monotonic instances that appear in the wrong zones. This fact alters the decision regions, which are relabeled and now appear associated with another class. It may affect the classification performance when new test instances have to be classified using this new training data set. In the case of MonTSS, those instances which are closer to the borders have been selected in the search space (based on the *Del* measure) as well as those that have a significant influence by the neighbors that surround them in order to be referential in the classification (thanks to *Infl* measure). This reduces the size of the final set, and also class boundaries are respected. Hence, it improves the classification performance of the M*k*NN when new test instances have to be predicted.

As a final analysis of the MonTSS algorithm we introduce the Table 7 where we study the threshold proposed (0.9) in the expression 16. Table 7 shows, in first column, each threshold value evaluated and in the second and third columns, the results associated to that value. The second column contains the average in MAE metric of 10-fcv for all data sets considered in the study in Table 5. The third column offers the average of 10-fcv for all data sets in NMI, Non-Comparability and Size. The best behaviors have been highlighted in bold.

Considering the results of Table 7, we can highlight the following:

- The value 0.5 causes the selected training set to be larger, maintaining a greater number of instances at the borders and within the classes. This situation negatively affects the predictive capabilities of the models (higher MAE values), as well as the non-monotonicity index (higher NMI). Keeping more samples also results in the number of Non-Comparable pairs growing.
- As the threshold value is increased, the number of instances selected is reduced (see Eq. (16)). In the case of threshold with a value of 0.7, we can see how the size is reduced (column Size in Table 7), as well as the number of pairs that are not comparable. With respect to 0.5, better MAE values are obtained and monotonicity index (NMI) in three of the classifiers. The difference appears in the MID classifier, whose predictive behavior is better as the input set increases. The improvement in most of the classifiers indicates that the selection of training sets is effective.

Table 5
Predictive MAE results for the 30 data sets.

Data sets	Relabeling	Original Data Set				MonTSS				Original Data Set			MonTSS		
	+ M*k*NN	M*k*NN	OLM	OSDL	MID	+ M*k*NN	+ OLM	+ OSDL	+ MID	NMI	Non Comparable	Size	NMI	Non-Comparable	Size
appendicitis	0.735	0.359	0.198	0.226	0.176	0.124	**0.105**	0.206	0.159	0.980	710.3	95.4	0.000	194.1	48.2
australian	0.271	0.230	0.306	0.264	**0.174**	0.265	0.349	0.203	0.175	0.152	92,424.6	621.0	0.000	85,242.1	594.8
balance	0.995	0.950	0.990	0.923	0.245	0.239	0.354	0.656	**0.235**	0.992	79,435.5	562.5	0.000	18,327.7	303.1
breast	0.354	0.326	0.318	0.455	0.325	**0.286**	**0.286**	0.336	0.300	0.477	12,342.5	249.3	0.006	6,243.6	180.3
cleveland	0.787	0.771	0.848	0.647	0.710	**0.626**	0.764	0.664	0.721	0.156	22,814.7	267.3	0.000	16,765.9	235.0
contraceptive	0.856	0.826	0.847	0.763	0.720	0.705	0.747	**0.684**	0.724	0.864	534,957.4	1325.7	0.104	256,392.9	920.0
era	2.135	2.481	2.150	**1.285**	1.363	1.746	2.169	1.422	1.830	1.000	291,577.0	900.0	0.269	183.5	24.1
esl	0.608	1.739	0.473	0.361	**0.342**	0.361	0.486	0.344	0.455	0.798	18,108.9	439.2	0.088	1,933.0	124.4
german	0.359	0.370	**0.289**	0.626	0.308	0.366	0.293	0.583	0.304	0.071	169,934.6	900.0	0.000	160,463.0	876.6
haberman	0.519	0.709	0.650	0.284	0.288	0.314	0.344	0.295	**0.279**	0.918	11,117.9	275.4	0.124	1,179.8	97.1
heart	0.285	0.293	0.333	0.374	0.244	0.285	0.293	0.363	**0.226**	0.034	14,373.2	243.0	0.000	11,663.1	224.9
lev	1.072	1.226	0.668	**0.392**	0.399	0.541	0.774	0.455	0.525	0.991	213,431.0	900.0	0.124	10,517.9	226.5
mammographic	0.386	0.466	0.193	0.371	0.201	0.173	0.183	**0.164**	0.170	0.851	54,129.3	747.0	0.213	5,548.7	188.3
pima	0.263	0.267	0.289	0.344	0.273	**0.260**	0.271	0.344	0.290	0.258	98,662.6	691.2	0.008	72,386.5	582.3
spectfheart	0.299	0.295	0.802	0.217	0.239	0.284	0.794	**0.206**	0.237	0.024	9,442.8	240.3	0.000	9,444.5	240.3
swd	0.447	1.084	0.763	**0.437**	0.468	0.519	0.648	0.457	0.487	0.975	234,339.2	900.0	0.060	35,797.1	398.3
wdbc	0.652	0.687	0.661	0.643	0.065	**0.035**	0.587	0.575	0.051	0.813	57,925.5	512.1	0.000	60,413.9	512.1
wine	0.202	0.247	0.954	0.949	0.084	**0.045**	0.893	0.618	0.095	0.220	8,417.3	160.2	0.000	8,191.4	160.2
winequality-red	0.726	0.560	1.674	2.529	0.454	**0.377**	1.410	1.537	0.449	0.313	663,745.0	1439.1	0.003	613,713.1	1392.6
wisconsin	0.036	**0.035**	0.113	0.041	0.054	0.047	0.159	0.050	0.051	0.023	15,597.8	614.7	0.000	7,091.4	305.4
auto-mpg4cl	1.252	0.964	1.082	0.696	0.248	0.252	0.462	0.365	**0.245**	0.795	46,346.5	352.8	0.000	10,636.4	208.5
baseball4cl	0.670	0.625	0.587	0.603	**0.480**	0.590	0.771	0.513	**0.480**	0.316	29,290.0	303.3	0.000	20,336.4	241.9
bostonhousing4cl	0.546	0.595	1.409	1.014	0.389	**0.352**	0.595	0.694	0.387	0.128	77,750.9	455.4	0.007	51,657.6	405.4
dee4cl	0.513	0.468	0.784	1.493	0.422	**0.312**	0.682	1.428	0.387	0.284	38,527.3	328.5	0.000	20,911.0	268.1
diabetes4cl	**0.508**	1.033	0.542	1.025	1.075	0.583	0.542	1.058	0.608	0.811	212.2	38.7	0.000	71.8	20.3
ele1_4cl	1.660	1.203	1.515	1.107	1.081	1.122	1.140	1.186	**1.056**	0.991	26,996.2	445.5	0.470	6,346.3	171.2
ele2_4cl	0.067	0.108	0.073	0.905	**0.044**	0.107	0.118	0.911	0.106	0.105	51,730.7	950.4	0.000	6,836.6	336.8
mortgage4cl	**0.043**	**0.043**	1.056	1.433	0.049	0.166	0.407	0.989	0.105	0.000	332,281.2	944.1	0.000	35,731.5	632.1
stock4cl	0.564	0.583	0.993	1.008	0.417	**0.397**	1.221	1.444	0.417	0.327	269,251.2	855.0	0.006	249,201.5	823.9
treasury4cl	**0.043**	**0.043**	1.085	1.443	0.073	0.184	0.446	0.984	0.114	0.003	332,476.0	944.1	0.000	28,968.8	596.3
Average	0.595	0.652	0.754	0.761	0.380	0.388	0.609	0.657	0.388	0.489	126,944.977	590.040	**0.049**	**60,413.037**	**377.967**

Table 6
Wilcoxon test to analyze the MAE results on classifiers with and without our preprocessing proposal.

Algorithm	Ranking	Adjusted p-value	vs. Algorithm
MonTSS+M*k*NN	R^+ 382.0 R^- 53.0	**0.000**	Relabeling+M*k*NN
MonTSS+M*k*NN	R^+ 427.0 R^- 38.0	**0.000**	M*k*NN
MonTSS+OLM	R^+ 335.5 R^- 100.0	**0.010**	OLM
MonTSS+OSDL	R^+ 327.5 R^- 107.5	**0.016**	OSDL
MonTSS+MID	R^+ 231.0 R^- 234.0	1.000	MID

Table 7
Threshold analysis for the expression (16).

	MonTSS						
Threshold	+M*k*NN	+ OLM	+OSDL	+MID	NMI	Non-Comparable	Size
0.5	0.484	0.647	0.691	**0.384**	0.204	83,150.633	472,146
0.7	0.394	0.609	0.660	0.387	0.138	78,224.924	402.115
0.9	**0.388**	**0.605**	**0.657**	0.388	**0.049**	60,413.037	377.967
0.95	0.413	0.612	0.701	0.388	0.053	**55,664.228**	**349.788**

- The threshold value of 0.9 improves even the results offered by using 0.7, both for MAE and NMI. Similarly, the only discordant classifier is MID, although the difference in MAE is minimal.
- In the case of 0.95, the size of the subset selected is the smallest, so the number of non-comparable pairs is the smallest as well. This major reduction has a significant impact on the number of instances located at the class boundaries, with class separability being its main mission. As separability is compromised, prediction (MAE) and monotonicity (NMI) are adversely affected.

It can be concluded that threshold 0.9 allows us to optimize the predictive behavior of most classifiers (except MID, with results close to optimal), and at the same time improve their monotonicity.

6. Concluding remarks

This paper aims to present a proposal of training set selection for monotonic classification called MonTSS. It selects the most representative instances focusing on the class distribution and monotonicity conditions of those instances. The experimental evaluation highlights some remarkable characteristics of the proposal.

MonTSS is able to select the most representative instances independently of the classifier to be applied later. This leads monotonic classifiers to always offer equal or better results than without preprocessing. Furthermore, data related metrics are notably improved, fully satisfying the monotonicity restrictions without affecting or modifying the nature of the original data. At the same time, it reduces the number of non-comparable pairs of instances and the size of the training data sets before the learning stage starts.

Acknowledgement

This work was supported by TIN2014-57251-P, by the Spanish "Ministerio de Economía y Competitividad" and by "Fondo Europeo de Desarrollo Regional" (FEDER) under Project TEC2015-69496-R and the Foundation BBVA project 75/2016 BigDaPTOOLS.

References

[1] J.S. Cardoso, J.F.P. da Costa, Learning to classify ordinal data: the data replication method, J. Mach. Learn. Res. 8 (2007) 1393–1429.
[2] M. Cruz-Ramírez, C. Hervás-Martínez, J. Sánchez-Monedero, P.A. Gutiérrez, Metrics to guide a multi-objective evolutionary algorithm for ordinal classification, Neurocomputing 135 (2014) 21–31.

[3] P.A. Gutiérrez, M. Pérez-Ortiz, J. Sánchez-Monedero, F. Fernandez-Navarro, C. Hervás-Martínez, Ordinal regression methods: survey and experimental study, IEEE Trans. Knowledge Data Eng. 28 (1) (2016) 127–146.
[4] W. Kotłowski, R. Słowiński, On nonparametric ordinal classification with monotonicity constraints, IEEE Trans. Knowledge Data Eng. 25 (11) (2013) 2576–2589.
[5] P.A. Gutiérrez, S. García, Current prospects on ordinal and monotonic classification, Prog. Artificial Intell. 5 (3) (2016) 171–179.
[6] H. Zhu, E.C. Tsang, X.-Z. Wang, R.A.R. Ashfaq, Monotonic classification extreme learning machine, Neurocomputing 225 (2017) 205–213.
[7] H. Daniels, M. Velikova, Monotone and partially monotone neural networks, IEEE Trans. Neural Netw. 21 (6) (2010) 906–917.
[8] C.-C. Chen, S.-T. Li, Credit rating with a monotonicity-constrained support vector machine model, Exp. Syst. Appl. 41 (16) (2014) 7235–7247.
[9] A. Ben-David, Monotonicity maintenance in information theoretic machine learning algorithms, Mach. Learn. 19 (1995) 29–43.
[10] R. Potharst, J. Bioch, Decision trees for ordinal classification, Intell. Data Anal. 4 (2000) 97–111.
[11] K. Cao-Van, B. De Baets, Growing decision trees in an ordinal setting, Int. J. Intell. Syst. 18 (2003) 733–750.
[12] W. Kotłowski, R. Słowiński, Rule learning with monotonicity constraints., in: ICML, vol. 382, 2009.
[13] C. Marsala, D. Petturiti, Rank discrimination measures for enforcing monotonicity in decision tree induction, Inform. Sci. 291 (2015) 143–171.
[14] S. Pei, Q. Hu, C. Chen, Multivariate decision trees with monotonicity constraints, Knowledge-Based Syst. 112 (2016) 14–25.
[15] J. Alcala-Fdez, R. Alcala, S. Gonzalez, Y. Nojima, S. Garcia, Evolutionary fuzzy rule-based methods for monotonic classification, IEEE Trans. Fuzzy Syst. (2017). http://dx.doi.org/10.1109/TFUZZ.2017.2718491 In press.
[16] H. Xu, W. Wang, Y. Qian, Fusing complete monotonic decision trees, IEEE Trans. Knowledge Data Eng. (2017) In press.
[17] A. Ben-David, L. Serling, Y. Pao, Learning and classification of monotonic ordinal concepts, Comput. Intell. 5 (1989) 45–49.
[18] S. Lievens, B. De Baets, K. Cao-Van, A probabilistic framework for the design of instance-based supervised ranking algorithms in an ordinal setting, Ann. Operat. Res. 163 (2008) 115–142.
[19] W. Duivesteijn, A. Feelders, Nearest neighbour classification with monotonicity constraints., in: ECML/PKDD (1), vol. 5211 of Lecture Notes in Computer Science, Springer, 2008, pp. 301–316.
[20] J. García, A. Albar, N. Aljohani, J.-R. Cano, S. García, Hyperrectangles selection for monotonic classification by using evolutionary algorithms, Int. J. Comput. Intell. Syst. 9 (1) (2016) 184–201.
[21] J. García, H.M. Fardoun, D.M. Alghazzawi, J.-R. Cano, S. García, Mongel: monotonic nested generalized exemplar learning, Pattern Anal. Appl. 20 (2) (2017) 441–452.
[22] B. Frénay, M. Verleysen, Classification in the presence of label noise: a survey, IEEE Trans. Neural Netw. Learn. Syst. 25 (5) (2014) 845–869.
[23] A. Feelders, Monotone relabeling in ordinal classification., in: IEEE International Conference on Data Mining (ICDM), 2010, pp. 803–808.
[24] R. Potharst, A. Ben-David, M.C. van Wezel, Two algorithms for generating structured and unstructured monotone ordinal data sets, Eng. Appl. Artificial Intell. 22 (4–5) (2009) 491–496.
[25] M. Rademaker, B. De Baets, H. De Meyer, Optimal monotone relabelling of partially non-monotone ordinal data, Optimiz. Methods Softw. 27 (1) (2012) 17–31.
[26] A. Feelders, T. Kolkman, Exploiting monotonicity constraints to reduce label noise: An experimental evaluation, in: Neural Networks (IJCNN), 2016 International Joint Conference on, IEEE, 2016, pp. 2148–2155.
[27] S. García, J. Derrac, J.-R. Cano, F. Herrera, Prototype selection for nearest neighbor classification: taxonomy and empirical study, IEEE Trans. Pattern Anal. Mach. Intell. 34 (2) (2012) 417–435.
[28] D.A. Silva, L.C. Souza, G.H. Motta, An instance selection method for large datasets based on markov geometric diffusion, Data Knowledge Eng. 101 (2016) 24–41.
[29] J.-R. Cano, N.R. Aljohani, R.A. Abbasi, J.S. Alowidbi, S. García, Prototype selection to improve monotonic nearest neighbor, Eng. Appl. Artificial Intell. 60 (2017) 128–135.
[30] J.-R. Cano, F. Herrera, M. Lozano, Stratification for scaling up evolutionary prototype selection, Pattern Recog. Lett. 26 (7) (2005) 953–963.
[31] J.-R. Cano, S. García, F. Herrera, Subgroup discover in large size data sets preprocessed using stratified instance selection for increasing the presence of minority classes, Pattern Recogn. Lett. 29 (16) (2008) 2156–2164.
[32] K. Nikolaidis, T. Mu, J.Y. Goulermas, Prototype reduction based on direct weighted pruning, Pattern Recogn. Lett. 36 (2014) 22–28.
[33] S. García, J. Luengo, F. Herrera, Data preprocessing in data mining, Springer, 2015.
[34] L. Guo, S. Boukir, Fast data selection for svm training using ensemble margin, Pattern Recogn. Lett. 51 (2015) 112–119.
[35] M.-J. Kim, I. Han, The discovery of experts' decision rules from qualitative bankruptcy data using genetic algorithms, Exp. Syst. Appl. 25 (4) (2003) 637–646.
[36] Q. Hu, X. Che, L. Zhang, D. Zhang, M. Guo, D. Yu, Rank entropy-based decision trees for monotonic classification, IEEE Trans. Knowledge Data Eng. 24 (11) (2012) 2052–2064.
[37] S. Lievens, B. De Baets, Supervised ranking in the weka environment, Inform. Sci. 180 (24) (2010) 4763–4771.
[38] J. Quinlan, Induction of decision trees, Mach. Learn. 1 (1) (1986) 81–106.
[39] Y. Song, J. Liang, J. Lu, X. Zhao, An efficient instance selection algorithm for k nearest neighbor regression, Neurocomputing 251 (2017) 26–34.
[40] J.-R. Cano, F. Herrera, M. Lozano, On the combination of evolutionary algorithms and stratified strategies for training set selection in data mining, Appl. Soft Comput. 6 (3) (2006) 323–332.
[41] L. Nanni, A. Lumini, S. Brahnam, Weighted reward-punishment editing, Pattern Recogn. Lett. 75 (2016) 48–54.
[42] J. Alcalá, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, J. Multiple-Valued Logic Soft Comput. 17 (255–287) (2010) 11.
[43] K. Bache, M. Lichman, UCI machine learning repository (2013). URL ⟨http://archive.ics.uci.edu/ml⟩.
[44] L. Gaudette, N. Japkowicz, Evaluation methods for ordinal classification, Lecture Notes in Computer Science 5549 (2009) 207–210.
[45] N. Japkowicz, M. Shah, Evaluating Learning Algorithms. A classification Perspective, Cambridge University Press, 2014.
[46] I. Milstein, A. Ben-David, R. Potharst, Generating noisy monotone ordinal datasets, Artificial Intell. Res. 3 (1) (2014) 30–37.
[47] J.D. Gibbons, S. Chakraborti, Nonparametric Statistical Inference, Springer, 2011.
[48] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.