Mass-Based Short Term Selection of Classifiers in Data Streams

Daniel Nowak Assis  
Polytechnic School  
Pontifícia Universidade Católica do Paraná  
Curitiba, Brazil  
daniel.nowak@pucpr.edu.br

Fabrício Emembreck, Jean Paul Barddal  
Graduate Program in Informatics (PPGIA)  
Pontifícia Universidade Católica do Paraná  
Curitiba, Brazil  
{fabricio, jean.barddal}@ppgia.pucpr.br

Abstract—Dynamic classifier selection (DCS) regards well-known machine learning techniques in the batch setting that leverage ensemble performance. Most of the methods use similarity-based methods as a proxy, culminating in high computation costs and becoming unfeasible in many streaming scenarios. In this paper, we propose a DCS method able to cope with the high-speed streaming setting, which is based on the performance of base learners in the most recent instances. The impact of our method is evaluated with different ensembles for data streams. We also propose modifications to an Online Boosting method, which has its performance improved with DCS. Our method increases the accuracy and kappa statistic of state-of-the-art ensembles with low overhead of time processing and memory.

I. INTRODUCTION

Online Machine Learning focuses on extracting knowledge from a high amount of data generated in many scenarios, such as bank transactions, spam filters, sensors, social media, and others. In the stream setting, data arrives continuously, one at a time, possibly supporting infinite arrival. Unlike standard Machine Learning, known as batch-setting, the predictive models take into account limitations in memory because it is unfeasible to store a high amount of instances, and processing-time, since an instance must be processed faster than the arrival of another instance, otherwise data must be discarded [1].

Additionally, the data distribution might change throughout time, an effect known as concept drift [12]. This change can affect the performance of predictive models, which must adapt in case of detection.

Ensemble-based methods are state-of-the-art algorithms for the data stream classification problem. These models combine the prediction of predictive models, having superior performance compared to monolithic models [2].

In general, ensembles in the stream setting literature are versions of batch setting algorithms, such as Bagging [3], Boosting [4] [5], and Random-Subspace-based [6] [7] [8]. Even if ensembles are more efficient than monolithic models, issues such as diversity maintenance, accuracy maintenance, and resource usage are still open questions in stream and batch settings. For instance, it is known that not every ensemble member might positively impact the final vote given by the ensemble, and selecting a subset of the ensemble to vote can achieve higher predictive performance. This process is known as the selection of classifiers [9].

Many works in literature propose novel dynamic classifier selection methods in the batch setting. In [9], the authors show that dynamically selecting classifiers to vote each instance is better than always selecting the same subset of classifiers for voting (static selection) and divided dynamic selection of classifiers into two approaches: (i) individual-based, where only one classifier of the ensemble is selected to classify an instance, and (ii) group-based, in which one or more classifiers of the ensemble are selected to vote an instance.

In the stream setting, as in the batch setting, most selection methods use kNN to determine the competence region of learners of an ensemble [9], [23]. Consequently, kNN causes a high overhead of processing time, being not practical in various streaming situations.

In this paper, we propose a Dynamic Selection of Classifiers (DCS) method to cope with streaming scenarios based on mass functions and Short-Term Assessment. Our proposal exploits the fact that classifiers can have a good performance in a concept (most of the ensembles in literature weights the vote of a classifier based on its accuracy), but does not perform well in the short term, having a negative impact in the final ensemble vote. We use sliding windows to store the performance of members of the ensemble in the most recent instances and apply dynamic selection of classifiers strategies. Experiments show that our method can achieve improvement in the performance of state-of-the-art data stream mining methods such as ARF [19], SRP [20], and BOLE [10].

The contributions of this work are summarized below:

- Usage of short sliding windows with different metrics to select classifiers in online ensembles.
- Empirical demonstration that selection of classifiers based on the sum of mass functions has the best results overall.
- Modifications to the BOLE algorithm [10], having its performance potentiated with DCS and becoming a strong contender in state-of-the-art methods.

This paper is divided as follows. Section II presents the formal definition of data stream classification, concept drift, and dynamic selection of classifiers. Section III discusses related works on ensemble learning and dynamic selection of classifiers. Section IV introduces our method and mod-
ifications to BOLE, which has its performance potentiated by our proposal. Section V reports the experimental results obtained. Finally, Section VI concludes this paper and states future works.

II. DATA STREAM CLASSIFICATION

This section defines data stream classification, concept drift, and dynamic selection of classifiers problems.

A. Classification Problem

A data stream is a set \( S = \{(X, Y)\} \), with \( X = \{\vec{x}_1, \ldots, \vec{x}_n\} \), a set of vectors of features, \( Y = \{y_1, \ldots, y_n\} \) a set of labels and \( n \) potentially tends to infinite. The aim is to build a predictive model that ideally represents the image of the classes (represented by distinct labels), given a vector of features \( f: \vec{x}_n \rightarrow y_n \).

B. Concept Drift

A concept \( C \) is a set of class probabilities and a density function of conditional probabilities [11], defined as follows:

\[
C = \bigcup_{\vec{x}_n \in X, y_n \in Y} (P[y_n], P[\vec{x}_n | y_n]) \tag{1}
\]

In 2 time stamps \( t_i \) and \( t_j \) in a stream \( S \), if \( t_j > t_i \) and \( C_{t_i} \neq C_{t_j} \), a concept drift occured. We reference the reader to the following works for more details about concept drift. [12], [13].

C. Dynamic Selection of Classifiers

Given a pool of classifiers \( H = \{h_1, \ldots, h_m\} \), a set of base learners to vote a label \( y_m \), a constrain \( c \) to discriminate learners with positive impact for the vote, \( H' \), the set of hypotheses with a selection of classifiers is defined by:

\[
H' = \bigcup_{h_i \in H} \begin{cases} 
  h_i, & \text{if } c_i \\
  \emptyset_i, & \text{otherwise}
\end{cases} \tag{2}
\]

The strategy to obtain the final hypothesis may differ for every ensemble and considers \( H' \) instead of \( H \) to determine the labels.

III. RELATED WORKS

This section introduces related works on (i) ensemble-based methods for streaming data and (ii) dynamic selection of classifiers.

A. Ensemble methods

Many works of literature introduce new ensemble methods to cope with streaming scenarios. One of the first works on ensemble learning for data stream mining is the online version [14] of Bagging [3] and Boosting [4] [5]. In Online Bagging, the authors simulate sampling with instances replacement by training with an instance \( k \) times, being \( k \) a variable that follows a Poisson\((\lambda = 1)\) distribution.

The probability of a base learner training with an instance once or more times is \( P[k > 0] = 1 - P[k = 0] = 1 - \frac{1}{e^{\lambda t}} = 1 - \frac{1}{e^{1}} \approx 63\% \). In [15], the authors propose the Leveraging Bagging algorithm and simulate sampling with Poisson\((\lambda = 6)\). This way, a base learner has \( \approx 99\% \) chance of training with an instance once or more times. This makes the members of the ensemble more specialized but causes an additional computation cost. Each ensemble member has an ADWIN detector [16]. If a concept drift is detected, the worst member of the ensemble (in terms of the estimated ADWIN error) is reset.

In [19], the authors propose the Adaptive Random Forest algorithm, a version of the Random Forests [6] to the stream setting, combining Poisson\((\lambda = 6)\), a subset of random features per node, so the trees evaluate a split, an ADWIN detector such as in [15], background learners to deal with evolutionary (with concept drift) streams and vote weighted by accuracy. In [20], the authors propose the Streaming Random Patches (SRP) algorithm, a version of the Random Patches [7] [8], using the same mechanisms as in ARF, but instead of considering a random subset of features per node, all of the split attempts are evaluated with a random subset of features defined at the creation of the base learner.

To adapt Boosting to streaming settings, the authors in [14] propose a classifier is trained with an instance \( k \) times, being \( k \) a variable that follows a Poisson\((\lambda = h_{n-1})\) distribution, receiving weights for the train based in the performance of another base classifier from the previous layer.

In [10], the authors present the Boosting-like Online Learning Ensemble (BOLE) algorithm. The authors propose changes to the Online Boosting [14] algorithm.

Instead of performing linear boosting, i.e., a classifier influences the training weights of the classifier that was created after him at the creation of the ensemble, there are restrictions regarding which classifier will receive the training weight.

First, all classifiers are sorted by predictions rate (line 4, Alg. 1), being a best-case scenario insertion sort \((O(n))\) because the higher number of processed instances lower is the variation in correct predictions rate will be (for more details, we refer the reader to the implementation by the authors in the MOA framework [17]).

Initially, the classifier with the worst prediction rates will start training when an instance \( I \) arrives. If \( I \) is correctly classified, it is assumed that base learners with higher correct prediction rates also have a great chance of correctly classifying the instance (an error is unlikely), and the best classifier not yet trained in the ensemble will receive the weight for training (line 5-8, Alg. 1). Otherwise, the worst classifier not yet trained will receive weights for voting (line 9-11, Alg. 1). It is worth noting that the value of \( \lambda \) decreases in case an instance is correctly classified (line 19, Alg. 1) and increases if there is a misclassification (line 23, Alg. 1). Unlikely errors have less impact in \( \lambda \) as more instances are processed since the classifiers with the best prediction rates will be trained subsequently. Base learners with the worst prediction rates that are the ones likely to make mistakes, receive the training weights more towards the end of the training process.

In BOLE, each member of the ensemble has a DDM
Some of the most relevant works in this area are: the processed instance, known as the region of competence. Weighting classifiers votes based on the Nearest-Neighbours. Most methods in the batch setting make usage of kNN, selected for voting can change at each processed instance. That deals with the combination of classifiers’ votes of ensembles. Dynamic Selection of Classifiers

Dynamic selection of classifiers is a family of algorithms that deals with the stream setting, for instance, each base learner must store the result of the instances buffered by the kNN (unfeasible because of memory issues), or each base learner must classify again the k-nearest instances of the region of competence, which is unfeasible because of time processing issues. Besides, there is also the calculation cost of the nearest neighbors, that is \( O(k \times n \times d) \) with brute force and \( O(n \times \log n \times k) \) with KD-Tree, where \( k \) is the number of neighbors, \( n \) is the number of instances buffered, and \( d \) is the dimension of the instances.

Algorithm 1 BOLE Training

**Input:** ensemble size \( M \), ensemble \( h \), instance \( I \), number of processed instances \( N \)

1: \( \text{minPos} \leftarrow 1; \text{maxPos} \leftarrow M \)
2: correct \leftarrow false
3: \( \lambda \leftarrow 1 \)
4: sort \( h \) by \( \frac{\lambda_m^\text{sc}}{\lambda_m^\text{sw} + \lambda} \) in ascending order;
5: for \( m \leftarrow 1 \) to \( M \) do
6: if correct then
7: \( \text{pos} \leftarrow \text{maxPos} \)
8: \( \text{maxPos} \leftarrow \text{maxPos} - 1 \)
9: else
10: \( \text{pos} \leftarrow \text{minPos} \)
11: \( \text{minPos} \leftarrow \text{minPos} + 1 \)
12: end if
13: \( K \leftarrow \text{Poisson}(\lambda) \)
14: for \( k \leftarrow 1 \) to \( K \) do
15: \( h_{\text{pos}} \leftarrow \text{Train}(h_{\text{pos}}, I) \)
16: end for
17: if \( h_{\text{pos}} \) has correctly classified \( I \) then
18: \( \lambda_m^\text{sc} \leftarrow \lambda_m^\text{sc} + \lambda \)
19: \( \lambda \leftarrow \lambda(N/N + 1) \)
20: correct \leftarrow true
21: else
22: \( \lambda_m^\text{sw} \leftarrow \lambda_m^\text{sw} + \lambda \)
23: \( \lambda \leftarrow \lambda(N/N + 1) \)
24: correct \leftarrow false
25: end if
26: end for
27: return \( h \)

Most of the methods in the DCS literature calculate the nearest neighbors of each instance, being impractical in many streaming scenarios due to the high computation cost.

In a scenario where KNORA is used in the stream setting, for instance, each base learner must store the result of the instances buffered by the kNN (unfeasible because of memory issues), or each base learner must classify again the k-nearest instances of the region of competence, which is unfeasible because of time processing issues. Besides, there is also the calculation cost of the nearest neighbors, that is \( O(k \times n \times d) \) with brute force and \( O(n \times \log n \times k) \) with KD-Tree, where \( k \) is the number of neighbors, \( n \) is the number of instances buffered, and \( d \) is the dimension of the instances.

Dynamic Selection Based Drift Handler [24] deals with the stream in chunks. At each chunk, a new classifier is created and added to the ensemble. Each chunk is a validation set, and in the prediction process, a batch DCS method is used. Preprocessed DCS I [25], and II [26] (PDCS I and II) are methods that focus on the imbalanced classification problem. Both methods deal with the stream as a chunk and have preprocessing techniques to treat imbalanced data applied to the chunk, being under or over-sampling. In PDCS I, for each chunk of data, an offline bagging ensemble is created. If the maximum value of ensembles (user-given) is reached, the ensemble with the lowest balanced accuracy (BAC) is removed. PDCS II can train in any classifier, and members of the ensemble are removed when a classifier has a BAC lower than a user-given threshold. In scenarios where there are no time processing limitations, this method can be a good fit.

In [23], the authors propose the Double Dynamic Classifier Selection, which has Online Bagging and online base learners such as Naive Bayes and Hoeffding Tree [1]. Even being significantly more efficient than the methods cited earlier, chunks of data and kNN are still used.

IV. Mass-Based Short Term Selection

To avoid manipulating chunks of data, since the chunk size is hard to determine, our proposed method explores the prequential evaluation, i.e., when an instance arrives, a classifier predicts the instance and receives the label for training. These steps are repeated until no more instances arrive. Also, in the prequential evaluation, it is possible to know the number of correct predictions of a base learner at any time.

Our proposed technique selects classifiers by observing only the performance in the short term, being the region of competence in the last \( n \) instances evaluated. As cited earlier, most of the online ensemble methods weight votes by accuracy, and there is the possibility that a learner has a good performance in the long term but a bad performance in the short term, negatively impacting the final vote of the ensemble.

The selection of classifiers strategies proposed in this paper is based on the number of correct predictions of each base learner in the last ten instances. Three selection strategies were tested. A classifier is selected for voting if its number of right
predictions is higher than (i) a fixed threshold, (ii) a fixed threshold and mean of right predictions among learners in the most recent instance, or (iii) higher than a fixed threshold and mode of right predictions among learners in the most recent instances. At least one classifier must fulfill the chosen requirement; otherwise, every learner of the ensemble votes. We opted to have a sliding window with size 10 for the following reasons:

- Memory: since each ensemble classifier needs a sliding window, large windows can compromise memory usage for large ensembles.
- Processing-time: small windows are preferable to extract metrics with a low cost of processing time.

The incremental update of sliding windows is presented in Alg. 2. The mass function of right predictions (mfrp) is calculated in lines 11–16. Since the region of competence is small, it is possible to test fixed thresholds to allow classifiers to vote. Besides, the probability of the mfrp being sparse is lower, increasing the number of voting classifiers and still inducing diversity. In larger regions of competence, in which the mfrp is probably sparse, metrics such as fixed thresholds need statistical analysis, such as quantiles, elevating the number of parameters, and mode becomes a not significant metric since the mfrp frequencies would be similar and would not represent how well the most of the learners are performing. In the batch setting, a small region of competence is suggested in [27], being the region of competence the 7-nearest neighbors to the test instance.

Formally, the discrete mass function of right predictions is defined as:

\[ f(n) = |H^*(n)| \]  

where \( H^*(n) \) is the set of base learners with \( n \) right predictions in the span of the sliding window.

Another strategy tested consists of defining the threshold as the mode of the sum of the mfrp in the last \( \delta \) instances. The sum of the mfrp in the last instances takes into account the performance (mfrp distribution) of the learners in a higher spam of instances, making the threshold not affected by abrupt changes. To achieve this, it is only necessary to set the mass control array maximum size (Alg. 2) to larger values without significant additional computation cost.

We refer to the application of short sliding windows and mfrp in DCS as Mass-based Short Term Selection (MSTS). The mfrp update runs in \( O(1) \), while the threshold calculation runs in \( O(sw_{max}) \), necessary to calculate the metrics used in this work.

A. Dealing with concept drift

For both ARF and SRP, short sliding windows were created for background learners. In case a drift is detected, and the background learner substitutes the main base learner, the sliding window is also substituted by the background learner sliding window.

In the original BOLE implementation by the authors in the MOA [17] framework, the values of \( \lambda_m^{sc} \) and \( \lambda_m^{sw} \) are not changed in case a concept drift is detected. This means that the values of \( \lambda_m^{sc} \) and \( \lambda_m^{sw} \) are influenced by the learners that are actively classifying instances. This happens because resetting the values of \( \lambda_m^{sc} \) and \( \lambda_m^{sw} \) can lead to the creation of a large number of too-weak base learners, thus imbalancing the ensemble. Aiming not to drastically change the subset of classifiers that vote, we opted to make the background learner inherit the sliding window of the main base learner in case a drift is detected. The background learner will have the same potential to vote since background learners will receive similar weights compared to the previous learner after the drift detection.

B. Modification in BOLE

We present a modification to the variation factor of \( \lambda \) (lines 19 and 26, Alg. 1) in the training phase. Instead of scaling the values of \( \lambda_m^{sc} \) and \( \lambda_m^{sw} \) to half the number of observations by the ensemble (\( \frac{N}{2} \)) (lines 19 and 23, Alg. 1) [14], we scale the values to the total number of observations (\( N \)). The value of \( \lambda_m^{sc} \) will decrease and \( \lambda_m^{sw} \) will increase, as desired [14], and the cumulative values of \( \lambda \) will be larger, making more specialized learners. However, scaling \( \lambda_m^{sc} \) and \( \lambda_m^{sw} \) with larger values make the variation factors of \( \lambda \) in right predictions higher (\( f_m^{sc} < 2 \) [14]), training with unnecessary weights in case of right predictions. We suspect this is the reason for having a decrease in accuracy in some datasets, and since \( \lambda_m^{sc} \) and \( \lambda_m^{sw} \) affect learners in the long term, our DCS method that takes into account the performance of base learners in the short term overcomes this problem. This is discussed in section V.

**Algorithm 2 Performance Mass Update**

**Input:** \( sw \): Sliding window of a classifier, \( sw_{max} \): sliding window maximum size (user-given), \( mc \): mass control array, \( mc_{max} \): mass control maximum size (user-given), \( m \): mass array (size = \( sw_{max} + 1 \)), \( I \): instance

1. \( \text{for classifier} \in \text{ Ensemble} \) \( \text{do} \)
2. \( \text{if classifier.GetVote}(I) = I.\text{value}() \) then
3. \( \quad \text{increment classifier.right_predictions by 1} \)
4. \( \quad \text{classifier.sw.insert}(1) \)
5. \( \text{else} \)
6. \( \quad \text{classifier.sw.insert}(0) \)
7. \( \text{end if} \)
8. \( \text{if classifier.sw.size()} > sw_{max} \) then
9. \( \quad \text{classifier.sw.remove_first()} \)
10. \( \text{end if} \)
11. \( \text{mc.insert(classifier.right_predictions)} \)
12. \( \text{increment m[classifier.right_predictions]} \) by 1
13. \( \text{if mc.size()} > mc_{max} \) then
14. \( \quad \text{decrement m[mc[0]]} \) by 1
15. \( \text{mc.remove_first()} \)
16. \( \text{end if} \)
17. \( \text{end for} \)
18. \( \text{calculate_threshold()} \)
V. EXPERIMENTS

In this section, we discuss the application of our DCS method in state-of-the-art ensembles. First, we introduce the experimental protocol adopted, followed by the results obtained and discussion.

We also made a repository\(^1\) that contains the code of our method and additional results, given the high number of experiments.

A. Experimental protocol

All the experiments were done with a prequential evaluation, in which instances are presented one by one, first for testing and later for training. Since all the methods in the DCS literature do not cope with the prequential evaluation, it is not possible to compare methods of the literature with our method. Therefore, we compared ensemble methods with and without our DCS proposal.

All the ensembles were set with 100 classifiers, and experiments were done with three state-of-the-art ensembles, namely ARF [19], SRP [20], and BOLE [10]. We excluded Leveraging Bagging [15] from experiments because of the high computation cost and inferior results compared to ARF, as shown in [19]. All the experiments were done in the MOA [17] framework. ARF and SRP were set with the parameters of their original papers. We denote BOLE\(_L1\) as BOLE with the proposed changes and BOLE\(_L2\) as the standard. For all ensembles, we used Hoeffding Trees [1] as base learners with Grace Period = 50, i.e., a split attempt occurs at every 50 instances.

We tested fixed thresholds between [3, 6] to evaluate the best value overall and applied metrics such as mean and mode higher than the fixed thresholds. To denote the sum of mfrp used with mode, we denote the parameter \(\delta\) as the number of instances considered to the sum. Given \(N\), the \(n\)-th observed sample, \(F\), the function considered for threshold calculation and \(f_n\), the mfrp of the \(n\)-th observed sample is given by Equation 4:

\[
F = \sum_{i=1}^{\delta} f_{N-i}
\]

We tested values of \(\delta\) for each algorithm from [1,4]. We refer to the algorithms without DCS (standard) as native. We did experiments in 14 datasets, nine real-world datasets, and five synthetic datasets. The synthetic datasets and parameters used are discussed as follows.

a) AGRAWAL [28]: This generator has six nominal features and three numerical features. Ten distinct functions map two classes. In this dataset, we simulate three abrupt datasets.

b) SEA [29]: This generator produces 3 numerical features \((f_1, f_2, f_3)\). If \(f_1 + f_2 \leq \theta\), the class has value 1, otherwise 0. In this dataset, we simulated three gradual drifts by changing the values of \(\theta\).

c) MIXED [18]: This generator has 2 boolean features \(v\) and \(w\), and 2 numerical features \(x\) and \(y\) between \([0,1]\). The examples are positive if two of the three conditions are satisfied: \(v, w, y < 0.5 + 0.3 \times \sin(3\pi x)\). Concept drift is simulated by inverting how classes are labeled. In this dataset, we use an imbalanced and balanced version and simulate three abrupt drifts.

d) RBF: This generator produces ten features and 5 class values. Data is generated based on the radial basis function (RBF). Centroids are generated in random positions and mapped with a standard deviation value, a weight, and a class label. In this dataset, incremental drifts are simulated by changing the centroids’ position at a continuous rate. The parameters used were 50 centroids at a speed change of 0.001.

The real-world datasets used were Outdoor, Nomao, Elec, GMSC, Rialto, Airlines, Covtype, Poker-Hand, and KDD99.

More details on the used datasets and their references can be found in the auxiliary repository.

B. Discussion

Tables I and II show the prequential accuracy. We opted to use fixed threshold = 5 and \(\delta = 3\) by having the best overall results per algorithm (see repository). MSTS leveraged Native ARF and Native BOLE, and MSTS-ARF-Mode presented the best-reported results. All versions of MSTS-BOLE\(_L1\) and MSTS ARF had better results than Native SRP, while Native BOLE\(_L1\) and Native ARF do not have results better than Native SRP. Accuracy gains were more noticeable in real-world datasets, especially in the datasets Elec, Covtype, Rialto, and Outdoor. The best-reported average gain in comparison with native algorithms was 1.38% for MSTS-ARF-Mode, 1.60% for MSTS-BOLE\(_L1\)-Mode and 1.58% for MSTS-BOLE\(_L2\)-Mode. In synthetic datasets, the highest gain reported was with BOLE\(_L1\) in the Mixed dataset, with a gain of 0.50%.

Even though Native BOLE\(_L1\) has better results than Native BOLE\(_L2\), in the datasets Nomao, Elec and Covtype, there was a loss in accuracy. We suspect this occurred because of the points discussed in section IV. However, MSTS BOLE\(_L1\) had better results than MSTS BOLE\(_L2\) in this datasets, overcoming this problem.

The only case MSTS leveraged SRP results was with the fixed threshold in real-world datasets, which was a small gain in the ranking. We suspect that because SRP is less stable and its trees grow faster, as discussed in [30], this probably causes abrupt changes in mfrp distribution, being difficult to draw a separation between learners that will have a positive impact on voting. The worst-reported average loss in accuracy was 12.11% for MSTS-SRP-Mode.

To evaluate datasets that have an imbalanced class distribution, usually the most used metric in the streaming literature is kappa statistic [31]. In the repository, we show the results for kappa. As in accuracy, MSTS BOLE\(_L1\) and MSTS ARF had better results than Native SRP. However, MSTS BOLE\(_L1\) Fixed and Mode presented better results than MSTS-ARF. Surprisingly, MSTS BOLE\(_L2\) Fixed and Mode presented better kappa results than Native SRP. In the imbalanced datasets

\(^1\)https://sites.google.com/view/msts-paper
TABLE I
PREQUENTIALL ACCURACY OF NATIVE ALGORITHMS AND ALGORITHMS WITH MSTS WITH FIXED THRESHOLD

<table>
<thead>
<tr>
<th></th>
<th>ARF</th>
<th>SRP</th>
<th>BOLE(_{L1})</th>
<th>BOLE(_{L2})</th>
<th>MSTS-ARF</th>
<th>MSTS-SRP</th>
<th>MSTS-BOLE(_{L1})</th>
<th>MSTS-BOLE(_{L2})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Native</td>
<td></td>
<td>Native</td>
<td>Fixed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Outdoor</td>
<td>64.325</td>
<td>68.475</td>
<td>69.800</td>
<td>65.125</td>
<td>67.125</td>
<td>72.650</td>
<td>73.575</td>
<td>70.925</td>
</tr>
<tr>
<td>Nomao</td>
<td>97.229</td>
<td>97.383</td>
<td>95.932</td>
<td>96.022</td>
<td>97.290</td>
<td>97.389</td>
<td>97.203</td>
<td>97.267</td>
</tr>
<tr>
<td>Elec</td>
<td>90.643</td>
<td>89.859</td>
<td>91.997</td>
<td>92.086</td>
<td>90.869</td>
<td>90.727</td>
<td>92.496</td>
<td>91.741</td>
</tr>
<tr>
<td>GMSC</td>
<td>93.585</td>
<td>93.509</td>
<td>92.750</td>
<td>92.700</td>
<td>93.584</td>
<td>93.507</td>
<td>93.083</td>
<td>93.123</td>
</tr>
<tr>
<td>Rialto</td>
<td>72.119</td>
<td>80.010</td>
<td>64.385</td>
<td>59.320</td>
<td>77.570</td>
<td>74.625</td>
<td>69.728</td>
<td>65.179</td>
</tr>
<tr>
<td>Cotype</td>
<td>94.713</td>
<td>95.350</td>
<td>93.785</td>
<td>94.194</td>
<td>94.826</td>
<td>95.010</td>
<td>95.104</td>
<td>94.751</td>
</tr>
<tr>
<td>Poker Hand</td>
<td>88.780</td>
<td>89.798</td>
<td>95.046</td>
<td>94.230</td>
<td>91.487</td>
<td>94.223</td>
<td>92.115</td>
<td></td>
</tr>
</tbody>
</table>

**Bold** values indicate the best results per data set.

TABLE II
PREQUENTIALL ACCURACY OF ALGORITHMS WITH MSTS WITH \{MEAN, MODE\}

<table>
<thead>
<tr>
<th></th>
<th>MSTS-ARF</th>
<th>MSTS-SRP</th>
<th>MSTS-BOLE(_{L1})</th>
<th>MSTS-BOLE(_{L2})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Mode</td>
<td>Mean</td>
<td>Mode</td>
</tr>
<tr>
<td>Outdoor</td>
<td>65.850</td>
<td>72.375</td>
<td>74.050</td>
<td>71.600</td>
</tr>
<tr>
<td>Nomao</td>
<td>97.371</td>
<td>97.397</td>
<td>97.365</td>
<td>91.241</td>
</tr>
<tr>
<td>Elec</td>
<td>91.106</td>
<td>90.512</td>
<td>92.205</td>
<td>91.360</td>
</tr>
<tr>
<td>GMSC</td>
<td>93.586</td>
<td>93.508</td>
<td>93.157</td>
<td>99.199</td>
</tr>
<tr>
<td>Rialto</td>
<td>76.111</td>
<td>74.196</td>
<td>69.542</td>
<td>61.276</td>
</tr>
<tr>
<td>Airlines</td>
<td>66.786</td>
<td>61.896</td>
<td>61.403</td>
<td>61.276</td>
</tr>
<tr>
<td>Cotype</td>
<td>94.927</td>
<td>94.914</td>
<td>94.505</td>
<td>94.999</td>
</tr>
<tr>
<td>Poker Hand</td>
<td>90.020</td>
<td>90.856</td>
<td>93.618</td>
<td>90.982</td>
</tr>
</tbody>
</table>

**Bold** values indicate the best results per data set.

---

Fig. 1. Nemenyi test on ARF accuracy with 95% confidence.

Fig. 2. Nemenyi test on SRP accuracy with 95% confidence.

Fig. 3. Nemenyi test on BOLE\(_{L1}\) accuracy with 95% confidence.

Fig. 4. Nemenyi test on BOLE\(_{L2}\) accuracy with 95% confidence.
Nomoa and GMSC, MSTS improved the kappa of the algorithms, but not for GSMC in MSTS BOLE\textsubscript{L1}. The best-reported average gain in comparison with native algorithms for all datasets was 1.317\% for MSTS-ARF-Mode, 1.218\% for MSTS BOLE\textsubscript{L1}-Fixed and 1.036\% for MSTS BOLE\textsubscript{L2}-Fixed.

Figure 7 compares all algorithms in terms of processing time. In the repository, we report all the results for CPU-Time and RAM-Hours. We calculate overhead with Equation 5.

\[
100 \times \left( \frac{M_{MSTS}}{M_{Native}} - 1 \right) \%
\]  

(5)

Tables III and IV report the mean overhead of MSTS compared to the native algorithms. Memory usage and CPU-Time overheads were small for all algorithms. The CPU-Time overhead of BOLE\textsubscript{L1} compared to BOLE\textsubscript{L2} was 35.33\%. However, BOLE\textsubscript{L1} still has CPU-Time lower than ARF.

Figures 1-4 show the Nemenyi test with all average rankings per algorithm in a number line, and the Critical Difference (CD), with a 95\% confidence level for all tests, was approximately 1.254. This means that each pair of algorithms with a ranking difference higher than 1.254 are statistically different. MSTS ARF and BOLE\textsubscript{L1} are statistically different to their native counterparts, while MSTS SRP and BOLE\textsubscript{L2} are not, even if MSTS BOLE\textsubscript{L2} reported better results compared to Native BOLE\textsubscript{L2}. In Figures 5 and 6, the Nemenyi test is presented with all the 16 algorithms versions together for accuracy and kappa, respectively, with a CD = 6.165. Not so many algorithms are statistically different with all compared together, but it is evident that BOLE\textsubscript{L2} is statistically different from the algorithms with the best results reported for both.
VI. CONCLUSIONS

In this paper, we proposed MSTS, a DCS method for data stream mining based on the performance of learners in the most recent instances. We also proposed a modification to BOLE that gets its results potentiated with DCS. The selection of classifiers strategies shown to leverage BOLE and ARF results, surpassing Native SRP results, with a low overhead of processing time and memory usage.

In future works we plan on creating a general method for any ensemble with any number of base learners, and test our method with different base learners, like Naive Bayes, Hoeffding Adaptive Tree [32] and Extremely Fast Decision Tree [33]. We also plan to analyze the SRP loss in accuracy, based on the points reported in [30].

VII. ACKNOWLEDGEMENTS

The first author dedicates this work to Sânziana Dobrovicescu and Mihai Codrea.

REFERENCES


[27] Rafael Cruz, Robert Sabourin, and George Cavalcanti. Dynamic classifier selection: Recent advances and perspectives. Information Fusion, 41, 05 2018


