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 Enembreck, 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 modifications 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 $X = \{\vec{x}_1, \dots, \vec{x}_n\}$, a set of vectors of features, $Y = \{y_1, \dots, y_n\}$ 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 class 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 $H = \{h_1, \dots, h_m\}$, 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:

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

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 $P[k > 0] = 1 - P[k = 0] = 1 - \frac{1}{e \cdot k!} = 1 - \frac{1}{e \cdot 0!} \approx 63\%$. 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 ($\mathcal{O}(n)$) 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 λ 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 λ 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

Algorithm 1 BOLE Training

Input: ensemble size M, ensemble h, instance I, number of processed instances N

$minPos \leftarrow 1$; $maxPos \leftarrow M$
$correct \leftarrow$ false
$\lambda \leftarrow 1$
sort h by $\frac{\lambda_m^{sc}}{\lambda_m^{sc}+\lambda_m^{sw}}$ in ascending order;
for $m \leftarrow 1$ to M **do**
  if $correct$ **then**
    $pos \leftarrow maxPos$
    $maxPos \leftarrow maxPos - 1$
  else
    $pos \leftarrow minPos$
    $minPos \leftarrow minPos + 1$
  end if
  $K \leftarrow Poisson(\lambda)$
  for $k \leftarrow 1$ to K **do**
    $h_{pos} \leftarrow$ Train(h_{pos}, I)
  end for
  if h_{pos} *has correctly classified* I **then**
    $\lambda_m^{sc} \leftarrow \lambda_m^{sc} + \lambda$
    $\lambda \leftarrow \lambda(\frac{N}{2 \cdot \lambda_m^{sc}})$
    $correct \leftarrow$ true
  else
    $\lambda_m^{sw} \leftarrow \lambda_m^{sw} + \lambda$
    $\lambda \leftarrow \lambda(\frac{N}{2 \cdot \lambda_m^{sw}})$
    $correct \leftarrow$ false
  end if
end for
return h

[18] concept drift detector. If a warning is detected, a new background learner is created and trained. If a drift is detected, the background learner substitutes the main base learner.

B. Dynamic Selection of Classifiers

Dynamic selection of classifiers is a family of algorithms that deals with the combination of classifiers' votes of ensembles, and many works have been done for the batch setting. The selection is dynamic because the subset of classifiers selected for voting can change at each processed instance. Most methods in the batch setting make usage of kNN, weighting classifiers votes based on the Nearest-Neighbours of the processed instance, known as the region of competence. Some of the most relevant works in this area are:

- K-Nearest-Oracles Eliminate (KNORA-E) [21]: Select the classifiers with the highest accuracy in the region of competence.
- K-Nearest-Oracles Union (KNORA-U) [21]: Select the classifiers that have at least one right prediction at the region of competence. The vote of the classifier is weighted according to the accuracy in the region of competence.
- META-DES [22]: A new classifier is created based on metrics of the region of competence to select new classifiers of the ensemble.

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 $\mathcal{O}(k \times n \times d)$ with brute force and $\mathcal{O}(n \times \log n \times k)$ 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 trained 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:

$$f(n) = |H^{\star}(n)| \tag{3}$$

where $H^{\star}(n)$ is the set of base learners with n right predictions in the spam of the sliding window.

Another strategy tested consists of defining the threshold as the mode of the sum of the mfrp in the last δ 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 $\mathcal{O}(1)$, while the threshold calculation runs in $\mathcal{O}(sw_{max})$, 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 λ_m^{sc} and λ_m^{sw} are

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:  for classifier ∈ Ensemble do
2:    if classifier.GetVote(I) = I.value() then
3:      increment classifier.right_predictions by 1
4:      classifier.sw.insert(1)
5:    else
6:      classifier.sw.insert(0)
7:    end if
8:    if classifier.sw.size() > sw_max then
9:      classifier.sw.remove_first()
10:   end if
11:   mc.insert(classifier.right_predictions)
12:   increment m[classifier.right_predictions] by 1
13:   if mc.size() > mc_max then
14:     decrement m[mc[0]] by 1
15:     mc.remove_first()
16:   end if
17: end for
18: calculate_threshold()
```

not changed in case a concept drift is detected. This means that the values of λ_m^{sc} and λ_m^{sw} are influenced by the learners that are actively classifying instances. This happens because resetting the values of λ_m^{sc} and λ_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 λ (lines 19 and 26, Alg. 1) in the training phase. Instead of scaling the values of λ_m^{sc} and λ_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 λ_m^{sc} will decrease and λ_m^{sw} will increase, as desired [14], and the cumulative values of λ will be larger, making more specialized learners. However, scaling λ_m^{sc} and λ_m^{sw} with larger values make the variation factors of λ in right predictions higher ($f_m^c < 2$ [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 λ_m^{sc} and λ_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.

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 δ 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:

$$F = \sum_{i=1}^{\delta} f_{N-i} \tag{4}$$

We tested values of δ 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 θ.

[1] https://sites.google.com/view/msts-paper

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, $y < 0.5 + 0.3 * \sin(3\pi x)$. 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

TABLE I

PREQUENTIAL ACCURACY OF NATIVE ALGORITHMS AND ALGORITHMS WITH MSTS WITH FIXED THRESHOLD

	ARF Native	SRP Native	BOLE$_{L1}$ Native	BOLE$_{L2}$ Native	MSTS-ARF Fixed	MSTS-SRP Fixed	MSTS-BOLE$_{L1}$ Fixed	MSTS-BOLE$_{L2}$ Fixed
Outdoor	64.325	68.475	69.800	65.125	67.125	72.650	73.575	70.925
Nomao	97.229	97.383	95.932	96.022	97.290	97.389	97.203	97.267
Elec	90.643	89.859	91.997	92.086	90.869	90.727	**92.496**	91.741
GMSC	93.585	93.509	92.750	92.700	93.584	93.507	93.083	93.123
Rialto	72.119	**80.010**	64.385	59.320	77.570	74.625	69.728	65.179
Airlines	66.720	**68.565**	61.211	61.169	66.750	62.686	61.427	61.297
Covtype	94.713	95.350	93.785	94.194	94.826	95.010	95.104	94.751
Poker Hand	88.780	89.798	**95.046**	94.230	89.823	91.487	94.223	92.115
KDD99	99.973	99.981	99.633	99.407	99.975	99.984	**99.991**	99.991
AGR	85.016	**92.463**	82.147	81.558	85.035	65.033	82.583	81.842
SEA	88.218	85.151	86.878	85.952	88.216	83.307	87.063	86.463
Mixed balanced	99.175	91.196	98.955	98.864	99.180	79.827	99.455	99.435
Mixed imbalanced	99.182	91.053	98.948	98.867	99.189	81.133	99.454	99.425
RBF	76.237	76.344	61.216	52.666	76.522	60.826	61.407	52.660
Avg. RANK$_{ALL}$	9.071	8.000	10.929	12.429	7.429	9.357	5.857	9.143
Avg. RANK$_{REAL}$	10.889	7.556	11.889	12.556	8.889	7.222	6.667	9.333
Avg. RANK$_{SYNT}$	5.800	8.800	9.200	12.200	4.800	13.200	**4.400**	8.800

Bold values indicate the best results per data set

TABLE II

PREQUENTIAL ACCURACY OF ALGORITHMS WITH MSTS WITH {MEAN, MODE}

	MSTS-ARF Mean	MSTS-SRP Mean	MSTS-BOLE$_{L1}$ Mean	MSTS-BOLE$_{L2}$ Mean	MSTS-ARF Mode	MSTS-SRP Mode	MSTS-BOLE$_{L1}$ Mode	MSTS-BOLE$_{L2}$ Mode
Outdoor	65.850	72.375	74.050	71.600	66.025	72.800	**76.075**	73.450
Nomao	97.371	97.397	97.365	97.400	**97.441**	97.421	97.435	97.429
Elec	91.106	90.512	92.205	91.241	92.300	74.481	92.221	91.448
GMSC	93.586	93.508	93.157	93.217	**93.593**	93.391	93.093	93.152
Rialto	76.111	74.196	69.542	64.834	77.604	74.635	70.308	65.803
Airlines	66.786	61.896	61.403	61.276	66.797	57.657	61.454	61.338
Covtype	94.927	94.914	94.505	94.099	**95.719**	74.607	95.283	95.170
Poker Hand	90.020	90.856	93.618	90.982	91.062	69.112	93.126	90.712
KDD99	99.976	99.985	99.991	99.991	99.981	99.983	99.989	99.990
AGR	85.090	64.980	82.501	81.716	85.062	59.601	82.535	81.780
SEA	**88.228**	82.226	86.832	86.632	88.225	81.356	86.819	86.560
Mixed balanced	99.175	79.147	**99.473**	99.448	99.140	75.233	99.464	99.434
Mixed imbalanced	99.179	80.540	**99.465**	99.431	99.140	76.088	99.448	99.418
RBF	**76.528**	57.197	61.265	52.317	76.395	43.146	61.219	52.330
Avg. RANK$_{ALL}$	6.786	10.214	6.357	9.214	**4.857**	12.500	5.571	8.286
Avg. RANK$_{REAL}$	8.111	8.000	7.222	9.444	**4.444**	10.556	5.556	7.667
Avg. RANK$_{SYNT}$	**4.400**	14.200	4.800	8.800	5.600	16.000	5.600	9.400

Bold values indicate the best results per data set

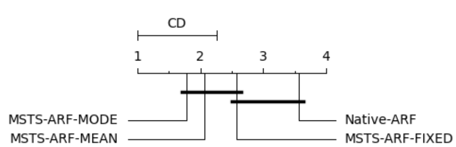

Fig. 1. Nemenyi test on ARF accuracy with 95% confidence.

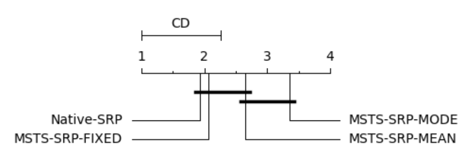

Fig. 2. Nemenyi test on SRP accuracy with 95% confidence.

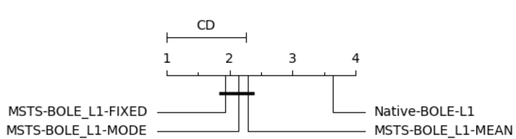

Fig. 3. Nemenyi test on BOLE$_{L1}$ accuracy with 95% confidence.

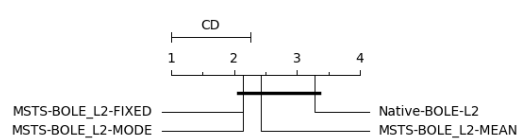

Fig. 4. Nemenyi test on BOLE$_{L2}$ accuracy with 95% confidence.

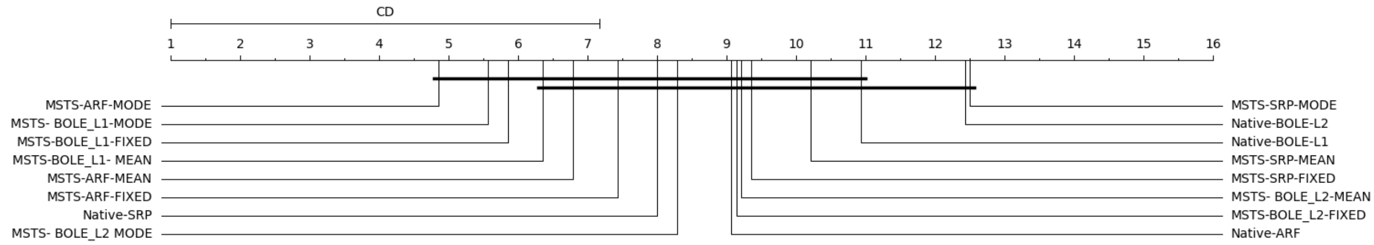

Fig. 5. Nemenyi test on accuracy with 95% confidence.

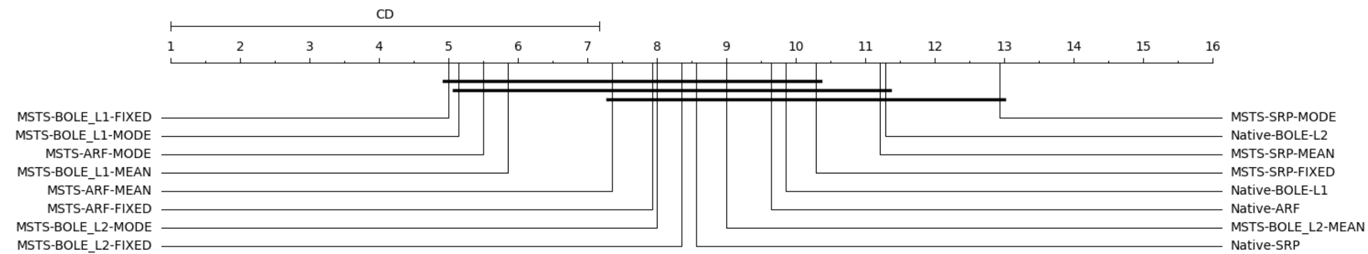

Fig. 6. Nemenyi test on kappa with 95% confidence.

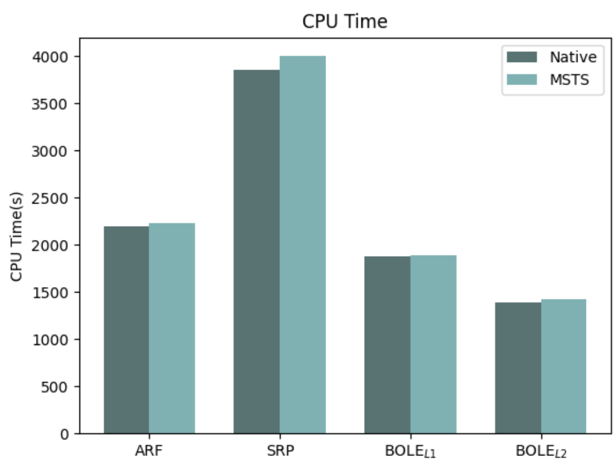

Fig. 7. Average CPU Time comparison

Nomoa and GMSC, MSTS improved the kappa of the algorithms, but not for GSMC in MSTS BOLE$_{L1}$. 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$_{L1}$-Fixed and 1.036% for MSTS BOLE$_{L2}$-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.

$$100 \times \left(\frac{M_{MSTS}}{M_{Native}} - 1 \right) \% \tag{5}$$

TABLE III
AVERAGE OVERHEAD (%) CPU-TIME

ARF	SRP	BOLE$_{L1}$	BOLE$_{L2}$
0,843	3,514	0,748	2,612

TABLE IV
AVERAGE OVERHEAD (%) RAM-HOURS

ARF	SRP	BOLE$_{L1}$	BOLE$_{L2}$
1,93	2,83	0,911	3,09

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$_{L1}$ compared to BOLE$_{L2}$ was 35.33%. However, BOLE$_{L1}$ 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$_{L1}$ are statistically different to their native counterparts, while MSTS SRP and BOLE$_{L2}$ are not, even if MSTS BOLE$_{L2}$ reported better results compared to Native BOLE$_{L2}$. 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$_{L2}$ is statistically different from the algorithms with the best results reported for both accuracy and kappa.

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. Acknolegdements

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

References

[1] Pedro Domingos and Geoff Hulten. 2000. Mining high-speed data streams. In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '00). Association for Computing Machinery, New York, NY, USA, 71–80. https://doi.org/10.1145/347090.347107

[2] Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. 2017. A Survey on Ensemble Learning for Data Stream Classification. ACM Comput. Surv. 50, 2, Article 23 (March 2018), 36 pages. https://doi.org/10.1145/3054925

[3] Leo Breiman, Bagging predictors. In Machine Learning 24, 123–140 (1996).

[4] Yoav Freund, "Boosting a weak learning algorithm by majority," Information and Computation, vol. 121, no. 2, pp. 256–285, 1995.

[5] Yoav. Freund and Robert E. Schapire, "Experiments with a new boosting algorithm," in International Conference on Machine Learning, vol. 96, 1996, pp. 148–156.

[6] Leo Breiman, Random Forests. Machine Learning 45, 5–32 (2001). https://doi.org/10.1023/A:1010933404324

[7] Gilles Louppe and Pierre Geurts, "Ensembles on random patches," in Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 2012, pp. 346–361.

[8] Panče Panov and Sašo Džeroski, "Combining bagging and random subspaces to create better ensembles," in International Symposium on Intelligent Data Analysis. Springer, 2007, pp. 118–129.

[9] Alceu S. Britto, Robert Sabourin, Luiz E.S. Oliveira, Dynamic selection of classifiers—A comprehensive review, In Pattern Recognition, Volume 47, Issue 11, 2014, Pages 3665-3680, ISSN 0031-3203, https://doi.org/10.1016/j.patcog.2014.05.003.

[10] Roberto S. M. d. Barros, Silas Garrido T. de Carvalho Santos and Paulo M. Gonçalves Júnior, "A Boosting-like Online Learning Ensemble," 2016 International Joint Conference on Neural Networks (IJCNN), 2016, pp. 1871-1878, doi: 10.1109/IJCNN.2016.7727427.

[11] Hai L. Nguyen, Yew K. Woon, Wee K. Ng, Li Wan, (2012). Heterogeneous Ensemble for Feature Drifts in Data Streams. In: Tan, PN., Chawla, S., Ho, C.K., Bailey, J. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2012. Lecture Notes in Computer Science(), vol 7302. Springer, Berlin, Heidelberg.

[12] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama and G. Zhang, "Learning under Concept Drift: A Review," in IEEE Transactions on Knowledge and Data Engineering, vol. 31, no. 12, pp. 2346-2363, 1 Dec. 2019, doi: 10.1109/TKDE.2018.2876857.

[13] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. ACM Comput. Surv. 46, 4, Article 44 (April 2014), 37 pages. https://doi.org/10.1145/2523813

[14] Nikunj C. Oza, Stuart J. Russell. Online Bagging and Boosting. Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics, PMLR R3:229-236, 2001.

[15] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, (2010). Leveraging Bagging for Evolving Data Streams. In Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2010. Lecture Notes in Computer Science(), vol 6321. Springer, Berlin, Heidelberg.

[16] Albert Bifet and Ricard Gavaldà, Learning from Time-Changing Data with Adaptive Windowing. In Proceedings of the 2007 SIAM International Conference on Data Mining (SDM). pp. 443-448, doi : 10.1137/1.9781611972771.42

[17] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl. Moa: Massive online analysis, a framework for stream classification and clustering. volume 11 of Proceedings of Machine Learning Research, pages 44–50, Cumberland Lodge, Windsor, UK, 01–03 Sep 2010. PMLR

[18] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 3171, no. September, pp. 286– 295, 2004, https://doi.org/10.1007/978-3-540-28645-5_29.

[19] Heitor M. Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. Machine Learning, 106(9-10):1469–1495, June 2017.

[20] Heitor M. Gomes, Jesse Read, and Albert Bifet. Streaming random patches for evolving data stream classification. In 2019 IEEE International Conference on Data Mining (ICDM), pages 240–249. IEEE, 2019.

[21] Albert H. R. Ko, Robert Sabourin, and Alceu Souza Britto, Jr. From dynamic classifier selection to dynamic ensemble selection. Pattern Recogn., 41(5):1718–1731, May 2008.

[22] Rafael Cruz, Robert Sabourin, George Cavalcanti, and Tsang Ing Ren. Meta-des: A dynamic ensemble selection framework using metalearning. Pattern Recognition, 48, 05 2015.

[23] Lucca P. Cavalheiro, Alceu De Souza Britto, Jean P. Barddal and Laurent Heutte, "Dynamically Selected Ensemble for Data Stream Classification," 2021 International Joint Conference on Neural Networks (IJCNN), 2021, pp. 1-7, doi: 10.1109/IJCNN52387.2021.9533702.

[24] P. R. L. D. Almeida, L. S. Oliveira, A. D. S. Britto, and R. Sabourin. Handling concept drifts using dynamic selection of classifiers. In 2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI), pages 989–995, Nov 2016.

[25] Paweł Zyblewski, Robert Sabourin, and Michał Wozniak. Preprocessed dynamic classifier ensemble selection for highly imbalanced drifted data streams. Information Fusion, 66:138 – 154, 2021.

[26] Paweł Zyblewski, Robert Sabourin, and Michał Wozniak. Data preprocessing and dynamic ensemble selection for imbalanced data stream classification. In Machine Learning and Knowledge Discovery in Databases, pages 367–379. Springer, 2020.

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

[28] R. Agrawal, T. Imielinski and A. Swami, "Database mining: a performance perspective," in IEEE Transactions on Knowledge and Data Engineering, vol. 5, no. 6, pp. 914-925, Dec. 1993, doi: 10.1109/69.250074.

[29] W. Nick Street and YongSeog Kim. 2001. A streaming ensemble algorithm (SEA) for large-scale classification. In Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '01). Association for Computing Machinery, New York, NY, USA, 377–382.

[30] H.M. Gomes, J. Read, A. Bifet et al. Learning from evolving data streams through ensembles of random patches. Knowl Inf Syst 63, 1597–1625 (2021). https://doi.org/10.1007/s10115-021-01579-z

[31] D. Brzeziński, J. Stefanowski, (2018). Ensemble classifiers for imbalanced and evolving data streams. Data Mining in Time Series and Streaming Databases, 83(1), 44–68.

[32] Albert Bifet, Ricard Gavaldà. "Adaptive learning from evolving data streams." In International Symposium on Intelligent Data Analysis, pp. 249-260. Springer, Berlin, Heidelberg, 2009.

[33] C. Manapragada, G. Webb, and M. Salehi. Extremely Fast Decision Tree. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '18). ACM, New York, NY, USA, 1953-1962. DOI: https://doi.org/10.1145/3219819.3220005