Dynamically Selected Ensemble for Data Stream Classification

Lucca Portes Cavalheiro, Alceu de Souza Britto Jr., Jean Paul Barddal
Graduate Program in Informatics (PPGIa)
Pontifícia Universidade Católica do Paraná
Curitiba, Brazil
{lucca, alceu, jean.barddal}@ppgia.pucpr.br

Laurent Heutte
LITIS
Université de Rouen Normandie
Rouen, France
laurent.heutte@univ-rouen.fr

2021 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-3900-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/IJCNN52387.2021.9533702

***Abstract*—Mining data streams is a hot topic in the machine learning (ML) community. In addition to learning and updating accurate models over time, these techniques must respect constraints that are not necessarily as strong in batch mode, such as time processing and memory consumption efficiency. A successful family of techniques in batch ML is dynamic classifier selection (DCS). However, these are roughly overlooked in data stream mining. In this paper, we propose a novel dynamic classifier selection framework for data streams called Double Dynamic Classifier Selection (DDCS). We compare DDCS against state-of-art methods for mining data streams in both synthetic and real-world datasets. Results depict that DDCS not only outperforms the state-of-art ensemble methods for data stream classification in terms of accuracy but is also significantly more efficient in terms of processing time and memory consumption.**

I. Introduction

With the ever-growing amount of data available in governments and corporations, machine learning has become more pervasive every day. Over the years, several approaches for learning descriptive and predictive models from humongous amounts of data have been developed in both batch and streaming fashions. In this paper, we target streaming methods, as they are tailored for learning and updating such models as new data become available with high accuracy, while at the expense of small memory and processing time [1].

Even though machine learning for data streams suffers from the same problems observed in batch scenarios, e.g., class imbalance, label availability, and high dimensionality, it does present intrinsic problems of its own, such as concept drift. Concept drifts occur when the data distribution changes. Consequently, machine learning models should be able to detect and adapt to them swiftly, so that accuracy rates are not jeopardized [2].

A common approach for achieving high accuracy rates in streaming classification is the use of ensembles, which consists of learning, updating, and combining multiple classifiers. In streaming scenarios, ensembles benefit from being versatile and easy to couple with drift detectors, which culminate in robust solutions for streaming data classification [3].

Overall, most of the existing ensemble-based methods for data streams assume that either: (i) all classifiers are equally important for prediction [4], or (ii) classifiers have their predictions weighted according to their past performance [5], [6]. Nonetheless, as ensemble members are diverse, these are expected to be experts in different parts of the feature space. Thus, dynamic classification selection (DCS) targets the determination of which of these classifiers are the most suited for predicting each query instance individually given its characteristics [7].

In this paper, we propose a novel ensemble for data stream classification that performs dynamic selection of classifiers in concept drifting data streams. We show that our proposal overcomes state-of-the-art ensembles with statistical confidence in terms of accuracy while also being significantly faster and light-weighted in terms of memory consumption. The implementation and experiments described are available at https://github.com/dcspaper/dcspaperCode.

This paper is divided as follows. Section II overviews data stream classification and concept drift. Section III discusses related works on ensemble learning and dynamic selection of classifiers in streaming scenarios. Section IV introduces our proposal, which is a dynamic ensemble with dynamic selection of classifiers for streaming scenarios. Section V reports the experimental results obtained. Finally, Section VI concludes this paper and reports envisioned future works.

II. Data Stream Classification

Traditional machine learning algorithms handle data in a *offline* manner. In other words, data are gathered together, labeled according to their proper classification, and used to train models. Next, these models are used to classify new unseen data instances. Nonetheless, most of the scenarios in which machine learning is applied to are based on data streams, as new data are continuously made available over time, thus requiring predictive models to be updated accordingly. More specifically, real-world applications that interact with data streams are required to process data quickly to both provide responses timely and to avoid data to be buffered, thus avoiding memory issues [2]. These scenarios become even more cumbersome if the data distribution changes over time, which is a phenomenon named 'concept drift' that renders predictive models obsolete (see Section II-A). Consequently, researchers and practitioners have been gathering efforts to

Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES)

978-0-7381-3366-9/21/$31.00 ©2021 IEEE

tailor techniques that are able to update predictive models as new training data become available, the so-called *data stream mining* methods.

A. Concept Drift

Concept drift is, by far, the most tackled issue of data stream learning. By definition, data streams are potentially unbounded. Consequently, the idea that these are made available over time is often followed. As a result, the processes that generate data may change over time, thus leading to changes in the data distribution. Focusing on classification, a concept drift is said to happen when the decision boundary between classes changes [2]. A theoretical explanation of concept drift is related to probability distributions. More specifically, we assume instances in the $\vec{x}^t$, y^t format, where $x \in X$ is a d-dimensional feature vector, $y \in Y$ is the corresponding label, and t is the arrival timestamp. Following [2], a concept drift is said to occur when the probability distribution between labels and features changes between timestamps t_i and t_j, i.e., $P_{t_i}(y|X) \neq P_{t_j}(y|X), i \neq j$. Another important characteristic of concept drifts regards its length. A drift is said to be abrupt when the concept completely changes between two subsequent timestamps t and $(t+1)$, or gradual if the posterior concept takes longer to stabilize.

Furthermore, drifts can be detected using several strategies, ranging from performance monitoring to algorithms that perform data distribution comparisons [8]. One of the most used detectors is the Drift Detection Method (DDM) [9], which monitors the error rate of the model: if it suddenly rises and exceeds certain thresholds, a drift is signaled. Another method, ADWIN [10], uses the same rationale, however keeping this monitoring limited to a window with variable size with rigorous statistical guarantees. Finally, it is noteworthy to highlight that drift detectors are often coupled with classifiers in ensembles, including those that are described in the following section.

III. Related Works

This section introduces related works to our approach that involve (i) ensemble learning for data streams, (ii) dynamic classifier selection, or both.

A. Ensembles for data stream classification

Ensembles have been widely used in batch machine learning to improve the accuracy of classifiers in complex problems. The rationale is that diverse members working together provide a better representation of the problem than a single classifier [11]. Given the success obtained by ensembles in batch scenarios, adaptations were made to apply such techniques in the data stream context.

One of the most classical ensemble algorithms for data stream mining is the OzaBag [4], which is an adaptation of the batch ensemble called Bootstrap Aggregating or Bagging [12]. In OzaBag, instances are resampled for training each member of the ensemble k times, such that k is drawn from a Poisson($\lambda = 1$) distribution. Therefore, the probability of an instance being used for training each classifier is approximately two thirds, as $P[k > 0] = 1 - P[k = 0] = 1 - \frac{e^{-1}}{k!} = 1 - \frac{e^{-1}}{0!} \approx 63\%$. Despite its simplicity, OzaBag and its internal mechanism for ensuring diversity have inspired more data streaming ensembles [1], [5], [6], and it remains competitive in multiple scenarios given the trade-off between accuracy, processing time, and memory consumption.

One of the current state-of-art ensemble algorithms for data stream mining is the Adaptive Random Forest (ARF) [5]. As OzaBag, ARF is an adaptation of the batch Random Forest algorithm [13]. ARF builds an ensemble of Hoeffding decision trees [14] using resampling with Poisson(λ=6) and by choosing a random subset of features that each tree considers when analyzing which feature to split on. Both of these traits allow ARF to induce diversity amongst its classifiers, as the number of times each instance is used for training and the features used for learning in each tree are likely to be different. Focusing on drift detection, ARF combines the use of drift detectors and background learners to optimize drift recovery speed. Even though ARF can be coupled with any drift detector, authors in [5] used ADWIN [10] as the off-the-shelf detector. More specifically, ARF couples two drift detectors with each tree, such that one has low confidence and the other, high confidence. When the low confidence detector is triggered, it flags a warning, which initializes the construction of a new tree in the background. If the high confidence detector is triggered, the tree pre-built in the background replaces the tree where the drift was flagged. ARF presents robust statistical measures for many data stream problems, yet, its internal mechanisms are computationally expensive, which renders ARF prohibitive in several applications [15].

Many more ensemble algorithms for data stream mining were proposed in the literature, often focusing on specific classification and data stream characteristics. For instance, we highlight the recent Kappa Updated Ensemble (KUE) [6], which as the names states, is based on the Kappa metric. Kappa measures how well the classifiers are performing w.r.t. pure statistical probability. This is particularly useful when dealing with imbalanced datasets, as in such scenarios, an evaluation with accuracy is not representative of the actual quality of the learned concept. In opposition to ARF, KUE works by treating the stream as chunks of data. With the arrival of a data chunk, the ensemble is updated following the OzaBag resampling process and random subspaces [16]. To keep the ensemble updated, at each new chunk, KUE removes ensemble members with Kappa rates below new classifiers that are trained solely on the chunk that arrived. During prediction, KUE selects only the members with Kappa greater than 0. Even though this selection process exists, it is not considered DCS, as it does not account for characteristics of each query instance individually. Finally, the output prediction is a Kappa-weighted majority vote obtained from the selected classifiers.

In this work, we used the OzaBag and ARF for comparison. OzaBag was chosen because it is the base for many other methods and depicts solid results in the trade-off between

accuracy, processing time, and memory consumption. Furthermore, ARF was selected as it is one of the current state-of-art ensembles for dealing with concept drift, which is tackled in our experiments. KUE is not considered because (i) it is focused on the imbalanced problem, which is not in the scope of our work, and (ii) the results obtained by it in large-scale analyses in [6] do not depict improvements when compared to ARF. Finally, the interested reader in thorough and comprehensive analyses on ensemble learning for data streams is referred to [3], [15].

B. Dynamic Classifier Selection

Combining the predictions of all the ensemble members is a crucial step for the ensemble to work. Most ensemble methods use the majority vote rule to output a prediction, which means that all classifiers have the same weight in defining the final prediction [3]. Weighted voting can also be used so that the performance of the classifiers are used to guide the importance of each classifier during prediction, which is the case of ARF [5], and KUE [6].

There is, however, another approach for combining the predictions obtained by classifiers in an ensemble, which is called dynamic classifier selection (DCS). DCS selects a subset of the classifiers in an ensemble to predict a query instance according to its characteristics. The rationale is that in a diverse ensemble, members are experts in different regions of the feature space, and thus, selecting them adequately improves the overall performance of the ensemble [7].

DCS is widely studied in the traditional (batch mode) machine learning area, and most of the methods have a similar structure. The process begins by dividing the training portion of the dataset into actual training and validation sets, such that the first part is used to induce the classification models, and the second is used to guide the selection process during prediction. During the prediction of a query instance, the most similar instances in the validation set are gathered using the K-Nearest Neighbors [17], which is referred to as the region of competence. As the region of competence consists of close instances to the test instance, its behavior on the base classifiers can be seen as a representation of the behavior on the test instance. Then, this behavior is analyzed to select the member(s) of the ensemble. Each method has its own way of performing this analysis. We can cite as some of the currently most relevant works in this area:

- Overall Local Accuracy - OLA [18]: Selects the classifier with the highest accuracy on the region of competence.
- Local Class Accuracy - LCA [18]: Selects the classifier with the highest recall on the region of competence.
- K-Nearest-Oracles Eliminate [19]: Selects all the classifiers with the highest accuracy on the region of competence.
- K-Nearest-Oracles Union [19]: Selects all the classifiers that correctly predicted at least one instance in the region of competence; the more instances correctly predicted, the more weight the classifier has in the final prediction.
- META-DES [20]: Treats the DCS process as a meta-learning problem; it extracts metrics from the region of competence to train a new classifier aimed at selecting the most competent members.

For more information regarding Dynamic Classifier Selection in static environments, the reader is referred to [7], [21].

In streaming environments, DCS methods are scarce. Since the training and testing phase are not clear cut as in the batch scenarios, DCS methods for data stream need to be adapted to encompass aspects such as the validation set. A relevant work is the Dynamic Selection Based Drift Handler (DYNSE) [22], which focuses on concept drifts. The method treats the data stream in chunks of data of size j. After the arrival of a data chunk, a new classifier is trained and added to the ensemble. In the original paper, the authors do not state that the maximum size of the ensemble must be bounded to n as it becomes an ever-growing ensemble. Nonetheless, in its implementation, when n is reached, the oldest classifier is removed from the ensemble. Furthermore, the last arrived chunk is always set as the validation set, so in the prediction step, batch DCS methods are used.

Focusing on the class imbalance imbalanced problem, two similar methods were proposed using data preprocessing. Since they are not given a name, we will hereby call them Preprocessed DCS I [23] and II [24] (PDCS I and II). Both are similar to DYNSE in the sense that they also process the data stream in chunks and new classifiers to the ensemble upon each chunk arrival. PDCS I works with an ensemble of ensembles so that at each chunk, a new ensemble is trained using offline bagging, yet, the data sample is stratified. If the ensemble maximum size n is reached, the ensemble with the lowest balanced accuracy (BAC) is removed from the pool. The validation set is also set as the last arrived batch; however, before updating it, a preprocessing technique focused on data imbalance, either under- or oversampling, is applied to the chunk. PDCS II, on the other hand, works with any classifier, and the preprocessing step is applied to the chunk before training. It also adds a step that removes any classifiers from the ensemble that have their BAC below a user-given threshold.

These methods can use both offline and online classifiers as they handle the data stream as chunks, and classifiers are only trained once. Consequently, the ensemble may not take advantage of online classifiers as its members are not updated as new data become available. Our proposal, described in the next section, takes a different direction as we focus on incremental classifiers and the approach is not limited to imbalanced data streams.

IV. Double Dynamic Classifier Selection

In this section, we introduce the Double Dynamic Classifier selection method. The rationale behind our proposal is that diverse classifiers are: (i) trained using different chunks of data and also using bagging and that (ii) they are appropriately selected according to their competence w.r.t. the query instance.

Algorithm 1: DDCS: Training step

input : `BaseClassifier`: base classifier to add to the ensemble, (X, Y): chunk to train, N: the maximum possible size of E, `use_bagging`: if set, train using online bagging, `init_all`: if set, initialize all the classifiers of the ensemble on the first run

1 $E \leftarrow get\_ensemble()$;
2 **if** $|E| = 0$ **then**
3 **if** *init_all* **then**
4 $E \leftarrow \{e_i |$ such that $1 \leq i \leq (N-1)$ and e_i is a `BaseClassifier` instance$\}$;
5 **end**
6 **end**
7 `new_classifier` $\leftarrow$ new `BaseClassifier()`;
8 **if** $|E| = N$ **then**
9 Remove from E the member e_i with lowest accuracy;
10 **end**
11 $E \leftarrow E \cup \{$`new_classifier`$\}$;
12 **if** *use_bagging* **then**
13 For $i = 1, \ldots, |E|$, train e_i using (X, Y) using Online Bagging;
14 **else**
15 For $i = 1, \ldots, |E|$, train e_i using (X, Y);
16 **end**
17 $V \leftarrow (X, Y)$;
18 `KNNSearcher` $\leftarrow$ `get_searcher()`;
19 `KNNSearcher.`$train(V)$;

Double Dynamic Classifier Selection (DDCS) structure is similar to DYNSE and PDCS (I and II). However, all ensemble members are updated when a new chunk arrives, and thus, the base learners in DDCS are required to be online learners, such as the Updatable Naive Bayes and Hoeffding Trees [14]. This represents an advantage over DYNSE and PDCS since DDCS is able to maintain members that have a more broad view of the concept, while having new members for each chunk. New members tend to be more specialized in recent concepts, while old members tend to make better predictions in stable concepts.

Algorithm 1 depicts DDCS training process. DDCS receives as input the base classifier of the ensemble, the chunk of instances and their labels (X, Y) to train on, the maximum ensemble size N, and the `use_bagging` and `init_all` hyper-parameters. At the beginning of the training process, i.e., the ensemble size is 0, if `init_all` is set, the ensemble is initialized with $(N-1)$ new base classifiers (lines 2 to 6), and otherwise, the ensemble will be filled at the pace of one classifier per chunk. Next, a new base classifier is created (line 7), which in the case of `init_all` being set, culminates in the ensemble reaching its maximum size N.

Lines 8 to 10 check if the maximum size of the ensemble is reached. If that is true, the member with the lowest accuracy is removed. The continuous replacement of poor performing classifiers induces poor performing classifiers, i.e., those affected by concept drifts, to be replaced by new models. The new classifier is then added to the ensemble (line 11). Lines 12 to 16 illustrate the step where all the members of the ensemble are trained. If `use_bagging` is set, the ensemble is updated using the same protocol described by OzaBag. Otherwise, the chunk will be trained on by each classifier as is. Next, lines 17 to 19 show the process of creating a new validation set and updating the algorithm responsible for finding the nearest neighbors.

Algorithm 2: Prediction Step of DDCS

input : x: instance to predict, `dcs`: DCS method to use, k: number of neighbors to gather

1 `KNNSearcher` $\leftarrow$ `get_searcher()`;
2 $E \leftarrow$ `get_ensemble()`;
3 `neighbors` $\leftarrow$ `KNNSearcher.find_neighbors`(x, k);
4 `selected_members` $\leftarrow$ `apply_dcs(neighbors, dcs,` E`)`;
5 $\hat{y} \leftarrow$ `predict(x, selected_members)`;
6 **return** $\hat{y}$;

Finally, the prediction step is depicted in Algorithm 2. This algorithm receives as input an instance to be predicted x, a DCS algorithm (`dcs`), and the number of neighbors (k) for similarity assessing in the DCS process. The KNN algorithm is used in line 3 to find the k-nearest neighbors in the validation set updated in the training phase. Next, these neighbors select the most competent members by applying the chosen DCS technique (line 4). Finally, the prediction is obtained from the selected members (line 5).

Since performance is relevant when mining data streams, the K-Nearest Neighbors search for the DCS methods is not traditional. Rather, we use the KD-Tree algorithm [25], which subdivides the validation set into subspaces. Consequently, when querying neighbors, distances only need to be computed within the group that most likely contains its similars. This drastically reduces the computational complexity as the original brute force KNN has the complexity of $O(1)$ and $O(k \times n \times d)$ for training and testing, respectively, where k is the number of neighbors, n is the number of instances buffered, and d is the dimension of the instances. On the other hand, KD-Tree has training time complexity of $O(d \times n \times \log n)$ and $O(k \times \log n)$ for testing. In DDCS, it is noteworthy that the training phase is performed once per chunk, while the testing phase is performed for every instance of the chunk.

V. Analysis

In this section we compare the proposed methods against state-of-the-art ensembles for data streams. First, we bring forward the experimental protocol adopted, followed by the results obtained and discussion.

A. Experimental protocol

In the following experiments, all classifiers were tested in an interleaved chunks test-then-train process. Therefore, each stream is divided in chunks of one thousand instances, and each chunk is used first for testing and later for training. The only exception is the first chunk, which is used solely for training.

The proposed testbed also included synthetic and real-world datasets. The synthetic generators were: Agrawal (Agr) [26], Asset Negotiation (An) [27], [28], and SEA [29]. The real-world datasets encompassed Electricity (Elec) [30], Nomao [31], and Spam Corpus [32]. All of the synthetic streams contain 100,000 instances. Different realizations of the streams were created with zero, one, two, and three concept drifts in both abrupt and gradual fashions, and gradual drifts were set to happen over a window of 1,000 instances.

Our proposal (DDCS) was compared against the OzaBag [4], Adaptive Random Forest (ARF) [5], and the Dynamic Selection Based Drift Handler (DYNSE) [22] ensemble classifiers. ARF was set to possess 100 Hoeffding Trees, while DDCS and DYNSE were bounded to the same number and classifier type. Even though these may not represent optimal values for all classifiers in all the data streams, this parametrization resulted in robust average results in [5]. Regarding the selection process, the number of neighbors k for DDCS and DYNSE was set to 7, as suggested in [21].

In this analysis, different hyper-parameter values in DDCS for `use_bagging` and `init_all` were used. As for the dynamic selection algorithms, KNORA-E (KNE) and KNORA-U (KNU) [19] were chosen for the comparison. These algorithms were selected because of their smaller computational cost when compared to more recent approaches such as META-DES [20].

Since execution time and memory were also considered, an important hyper-parameter of ARF was optimized to provide a fair comparison, `max_features`. This parameter represents the size of the subset of features that ARF will take into account when splitting a node. It is important to optimize it because, in some cases, it can lead to changes in time of execution without losing its accuracy. The values used in this parameter ranged from 2 to 6, plus the square root of the total number of features. In the following results, we show two variants for ARF. The first regards the optimized ARF version towards accuracy (ARF_ACC), while the second focuses on processing time (ARF_TIME). Regarding DDCS, only the execution with both parameter `use_bagging` and `init_all` set to *false* are reported, as these were the most positive results. The remainder of the hyper-parameters were set as the default provided in the Massive Online Analysis (MOA) framework [33]. Experimentation encompassed the assessment of accuracy, execution time (CPU time), and memory use. The results reported are averages computed after 20 experiment runs.

After the results were gathered, we applied two statistical tests on the results, i.e., the Friedman and Nemenyi combination proposed in [8], and the pairwise Bayesian comparison brought forward in [34]. Finally, the source code behind our proposal, the script to reproduce the results, and the full list of the results are available at https://github.com/dcspaper/dcspaperCode.

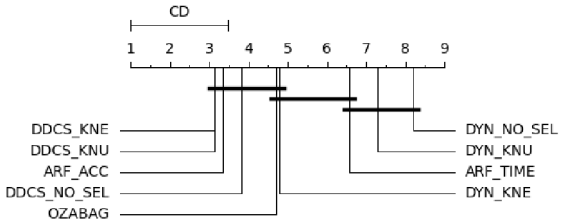

Fig. 1. Critical distance plot for accuracy results.

B. Discussion

Table I shows the accuracy values obtained. Even though the algorithm that appeared most times as the winner (7) was ARF_ACC, its superiority was restrained to a single dataset, the SEA generator. Next, we see DDCS with KNORA-E appearing as the winner six times, which occurred in a multitude of datasets. In such scenarios, the mean accuracy gain provided by DDCS with KNORA-E achieved 3.04%. However, if we do the opposite analysis, the mean difference between ARF_ACC and the proposed methods was of 0.83%. Furthermore, the results depict that DYNSE (DYN) methods were outperformed by DDCS methods, except on SEA Generator and Spam Corpus.

It is also relevant to discuss the accuracy differences observed between ARF_ACC and ARF_TIME, which is significant. In average, ARF_ACC resulted in % improvements when compared to ARF_TIME, thus highlighting the impact of hyper-parameter tuning. It is also worth noting that DDCS without any selection method (DDCS_NO_SEL) obtained competitive results, appearing as the best-performing classifier thrice.

Figure 1 displays the Friedman + Nemenyi statistical tests results. DDCS with KNORA-E and KNORA-U are virtually tied in the first position. They are followed by ARF_ACC, and even though their difference is not significant according to the test, as previously stated, the gain provided by DDCS is much greater than that provided by ARF_ACC. DDCS with no selection method outperformed traditional OzaBag. The DYNSE methods were in the last positions, together with ARF_TIME, DYNSE with KNORA-E greatly outperformed its other executions.

Figure 2 shows the Bayesian analysis plot of DDCS with KNORA-E and KNORA-U against ARF_ACC. This analysis outputs the probability of the algorithm a being better than the algorithm b, considering a rope value, which is a region practical equivalence [34]. What can be inferred from the plot is that DDCS with KNORA-E and KNORA-U has a much greater probability of being better than ARF_ACC. With KNORA-E, for instance, its probability of outperforming

TABLE I
ACURRACY OF THE BASELINE METHODS AGAINST DDCS.

Dataset	DYN_KNE	DYN_KNU	DYN_NO_SEL	OZABAG	DDCS_KNE	DDCS_KNU	DDCS_NO_SEL	ARF_ACC	ARF_TIME
Agr-1X_Grad	83.53	73.77	78.95	89.97	90.89	90.95	**91.03**	84.76	69.51
Agr-1X_Abr	83.13	74.45	79.07	90.62	91.05	91.15	**91.76**	84.59	68.81
Agr-2X_Grad	80.19	71.29	72.88	86.25	**89.86**	89.20	89.36	81.96	70.32
Agr-2X_Abr	80.55	71.50	71.32	87.98	90.11	90.23	**90.35**	82.17	69.78
Agr-3X_Grad	80.74	73.38	73.30	83.88	**90.16**	88.81	88.51	84.15	72.74
Agr-3X_Abr	81.18	74.36	72.75	87.34	**90.51**	89.39	89.44	84.92	72.40
Agr-No_drift	86.67	78.56	85.79	**93.72**	92.60	93.03	93.31	88.51	68.56
An-1X_Grad	92.73	91.42	85.57	93.71	93.54	**93.78**	93.58	93.48	91.46
An-1X_Abr	92.74	91.14	85.59	**93.70**	93.43	93.48	93.51	93.27	91.47
An-2X_Grad	92.52	91.07	79.92	93.44	**93.66**	93.58	93.39	93.35	91.29
An-2X_Abr	92.47	90.56	79.70	**93.49**	93.41	93.46	93.32	93.29	91.27
An-3X_Grad	92.03	90.42	78.86	92.62	**93.58**	93.32	93.07	93.19	91.19
An-3X_Abr	91.71	90.20	78.64	92.75	92.84	**92.95**	92.92	92.81	90.88
An-No_drift	92.99	92.63	92.85	**94.00**	93.61	93.80	93.76	93.47	91.66
SEA-1X_Grad	88.44	87.48	87.23	87.47	88.22	87.95	87.55	**89.18**	88.53
SEA-1X_Abr	88.39	87.49	87.21	87.52	88.18	88.05	87.55	**89.15**	88.53
SEA-2X_Grad	88.25	87.08	86.52	86.93	88.06	87.76	87.16	**88.96**	88.04
SEA-2X_Abr	88.22	87.03	86.54	86.89	88.05	87.77	87.15	**88.97**	88.11
SEA-3X_Grad	88.10	86.86	86.09	86.15	87.96	87.74	87.12	**88.67**	88.02
SEA-3X_Abr	88.23	86.96	86.13	86.21	88.05	87.74	87.13	**88.72**	88.08
SEA-No_drift	88.53	88.14	87.93	88.02	88.39	88.34	87.84	**89.35**	88.77
Elec	76.69	76.91	78.71	76.25	78.11	79.09	**79.35**	78.90	61.71
Nomao	93.34	91.85	87.53	92.37	**93.73**	93.42	93.21	93.45	91.00
Spam Corpus	75.80	75.69	61.96	**76.00**	75.30	75.36	74.93	75.91	68.26
Avg. rank	4.79	7.29	8.21	4.71	**3.12**	**3.12**	3.83	3.33	6.58

ARF_ACC (bottom left) is 68%, and for KNORA-U, this value was 73%. The chance of the difference between them lying into the rope (top) is 31% and 24% for KNORA-E and KNORA-U respectively. The chance of ARF_ACC overcoming DDCS (bottom right) is close to zero.

When focusing on accuracy, DDCS presents robust results when compared to other methods. For instance, when compared to the state-of-art ARF_ACC, it outperformed DDCS in a single synthetic experiment (SEA) and a real-world dataset (Elec). In all other experiments, DDCS presented superior results. Comparing it to DYNSE variants, although DDCS was outperformed in the SEA generator, overall, unlike DDCS, DYNSE did not present competitive results against the state-of-art. When including memory consumption and time processing, this advantage is vastly increased. Table II shows the absolute and relative to ARF_ACC processing time and memory consumption of the algorithms tested. DDCS with KNORA-E and KNORA-U used only 6.00% of the time ARF_ACC took to run and consumed only around 1.20% of its memory. Consequently, when accounting for the trade-off between accuracy, processing time, and memory consumption, we conclude that DDCS depicts competitive accuracy results while being computationally light-weighted compared to ARF.

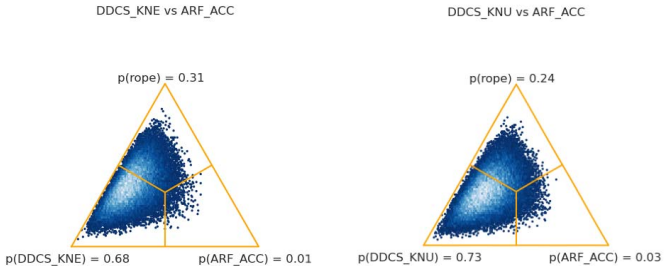

Fig. 2. Bayesian comparison of DDCS with KNORA-E (left) and KNORA-U (right) against ARF_ACC, with rope set to 0.5, and 50000 samples drawn.

VI. CONCLUSION

This work presented DDCS, a method for applying dynamic classifier selection in data stream mining environments. DDCS builds an ensemble by treating the stream as chunks of data and adding new classifiers when the chunks arrive. We have assessed our proposal in both synthetic and real-world scenarios, thus comparing it against the state-of-the-art Adaptive Random Forest (ARF) classifier.

Results show that our proposal is competitive against ARF in terms of accuracy. Regarding processing time, and memory usage, DDCS strongly outperformed ARF in the majority of datasets. More specifically, according to the Bayesian analysis, DDCS has around 70% chance of outperforming ARF. Regarding processing time and memory consumption, the numbers are even more impressive, as DDCS used only about 6.00% of the time and 1.00% of the memory used by ARF.

As future works, we plan to make DDCS available in `scikit-dyn2sel` "unpublished" [35], a tool for dynamic classifier selection in data streams; as well as testing our proposal in class imbalance and scarce label availability applications.

ACKNOWLEDGMENT

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

TABLE II
PROCESSING TIME AND MEMORY CONSUMPTION OF THE BASELINE METHODS AGAINST DDCS.

Method	Time (s)	Time fraction w.r.t. ARF_ACC (%)	Ram-Hours (GB)	Memory fraction w.r.t. ARF_ACC (%)
DYN_NO_SEL	3.83	0.61%	5.17×10^{-7}	0.02%
DDCS_NO_SEL	17.22	2.73%	1.60×10^{-6}	0.07%
DYN_KNU	21.09	3.35%	1.42×10^{-5}	0.62%
DYN_KNE	22.50	3.57%	1.44×10^{-5}	0.63%
DDCS_KNU	38.03	6.04%	2.73×10^{-5}	1.20%
DDCS_KNE	42.18	6.70%	2.95×10^{-5}	1.30%
OZABAG	72.64	11.54%	5.72×10^{-5}	2.51%
ARF_TIME	162.04	25.75%	2.51×10^{-4}	11.04%
ARF_ACC	629.40	100.00%	2.28×10^{-3}	100.00%

REFERENCES

[1] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Leveraging bagging for evolving data streams. In *Machine Learning and Knowledge Discovery in Databases*, pages 135–150. Springer Berlin Heidelberg, 2010.

[2] Geoffrey I. Webb, Loong Kuan Lee, Bart Goethals, and François Petitjean. Analyzing concept drift and shift from sample data. *Data Min. Knowl. Discov.*, 32(5):1179–1199, September 2018.

[3] Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. A survey on ensemble learning for data stream classification. *ACM Comput. Surv.*, 50(2), March 2017.

[4] Nikunj C. Oza. Online bagging and boosting. In *SMC*, pages 2340–2345. IEEE, 2005.

[5] 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.

[6] Alberto Cano and Bartosz Krawczyk. Kappa updated ensemble for drifting data stream mining. *Machine Learning*, 109(1):175–218, October 2019.

[7] Alceu S. Britto, Robert Sabourin, and Luiz E.S. Oliveira. Dynamic selection of classifiers—a comprehensive review. *Pattern Recognition*, 47(11):3665 – 3680, 2014.

[8] Jaka Demšar and Zoran Bosnić. Detecting concept drift in data streams using model explanation. *Expert Systems with Applications*, 92:546 – 559, 2018.

[9] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In Ana L. C. Bazzan and Sofiane Labidi, editors, *Advances in Artificial Intelligence – SBIA 2004*, pages 286–295, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[10] Albert Bifet and Ricard Gavaldà. *Learning from Time-Changing Data with Adaptive Windowing*, pages 443–448.

[11] Thomas G. Dietterich. Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, page 1–15, Berlin, Heidelberg, 2000. Springer-Verlag.

[12] L. Breiman. Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California at Berkeley, 1996.

[13] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.

[14] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. pages 71–80. ACM Press, 2000.

[15] Bartosz Krawczyk, Leandro L. Minku, João Gama, Jerzy Stefanowski, and Michał Woźniak. Ensemble learning for data stream analysis: A survey. *Inf. Fusion*, 37:132 – 156, 2017.

[16] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[17] David W Aha, Dennis Kibler, and Marc K Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.

[18] K. Woods, W. P. Kegelmeyer, and K. Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):405–410, April 1997.

[19] 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.

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

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

[22] 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.

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

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

[25] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[26] R. Agrawal, T. Imielinski, and A. Swami. Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, Dec 1993.

[27] Fabrício Enembreck, Bráulio Ávila, Edson Scalabrin, and Jean-Paul Barthès. Learning drifting negotiations. *Applied Artificial Intelligence*, 21:861–881, 10 2007.

[28] Jean Paul Barddal, Heitor Murilo Gomes, Fabrício Enembreck, Bernhard Pfahringer, and Albert Bifet. On dynamic feature weighting for feature drifting data streams. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016*, volume 9852 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2016.

[29] W. Nick Street and YongSeog Kim. 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, page 377–382, New York, NY, USA, 2001. ACM.

[30] P. P. Rodrigues, J. Gama, and J. Pedroso. Hierarchical clustering of time-series data streams. *IEEE Trans. on Knowledge and Data Engineering*, 20(5):615–627, 2008.

[31] Laurent Candillier and Vincent Lemaire. Design and analysis of the nomao challenge active learning in the real-world. pages 1–8, 08 2013.

[32] Ioannis Katakis, Grigorios Tsoumakas, and I. Vlahavas. Dynamic feature space and incremental feature selection for the classiflcation of textual data streams. *Proceedings of ECML/PKDD-2006 Intl. Workshop on Knowledge Discovery from Data Streams*, 01 2006.

[33] 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.

[34] Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *Journal of Machine Learning Research*, 18(77):1–36, 2017.

[35] Lucca Portes Cavalheiro, Jean Paul Barddal, Alceu de Souza Britto Jr, and Laurent Heutte. scikit-dyn2sel – a dynamic selection framework for data streams (arxiv:2008.08920v1), 2020.