Data Mining and Knowledge Discovery (2020) 34:50–74
https://doi.org/10.1007/s10618-019-00656-w

A drift detection method based on dynamic classifier selection

Felipe Pinagé[1,2] · Eulanda M. dos Santos[1] · João Gama[3]

Received: 14 March 2017 / Accepted: 1 October 2019 / Published online: 11 October 2019

Abstract
Machine learning algorithms can be applied to several practical problems, such as spam, fraud and intrusion detection, and customer preferences, among others. In most of these problems, data come in streams, which mean that data distribution may change over time, leading to concept drift. The literature is abundant on providing supervised methods based on error monitoring for explicit drift detection. However, these methods may become infeasible in some real-world applications—where there is no fully labeled data available, and may depend on a significant decrease in accuracy to be able to detect drifts. There are also methods based on blind approaches, where the decision model is updated constantly. However, this may lead to unnecessary system updates. In order to overcome these drawbacks, we propose in this paper a semi-supervised drift detector that uses an ensemble of classifiers based on self-training online learning and dynamic classifier selection. For each unknown sample, a dynamic selection strategy is used to choose among the ensemble's component members, the classifier most likely to be the correct one for classifying it. The prediction assigned by the chosen classifier is used to compute an estimate of the error produced by the ensemble members. The proposed method monitors such a pseudo-error in order to detect drifts and to update the decision model only after drift detection. The achievement of this method is relevant in that it allows drift detection and reaction and is applicable in several practical problems. The experiments conducted indicate that the proposed method attains high performance and detection rates, while reducing the amount of labeled data used to detect drift.

Keywords Concept drift · Drift detection · Ensemble classifiers · Self-training · Data streams

Responsible editor: Charu Aggarwal.

Extended author information available on the last page of the article

1 Introduction

In the literature related to classification systems in data streams, several problems present the following characteristic: given a set of known concepts, data distribution drifts from one concept to another. This is the so-called concept drift problem, which can be defined as $P_j(x, \omega) \neq P_k(x, \omega)$, where x represents a data instance, ω represents a class, and the change occurs from time t_j to time t_k, where $(t_j < t_k)$. As a consequence, an optimal prediction function for $P_j(x, \omega)$ will be no longer optimal for $P_k(x, \omega)$ (Ang et al. 2013).

When a concept drift occurs, the decision model must adapt itself in order to assure high classification performance. These adaptive methods can be applied to several practical problems, such as intrusion detection (Spinosa et al. 2008), spam filtering (Kmieciak and Stefanowski 2011), web click stream (Huang 2008), and fraud detection (Wang et al. 2003). In most of these problems, we do not have previous knowledge of the incoming data labels. Moreover, due to the massive quantity of incoming data, labeling the whole data is time-consuming and requires human intervention. In addition, true labels of newly streaming data instances are usually not immediately available.

Despite these problems, the most common detection methods in the literature are based on accuracy monitoring, which may lead to decreasing the system performance, besides requiring fully labeled data. An alternative to explicit detection by accuracy monitoring-based methods is blind adaptation. In this case, the system is periodically updated with no verification of changing occurrence. There may be drawbacks to blind adaptation as well, since this approach may lead to unnecessary system updates and to a high computational cost. These interesting observations motivated us to propose a method focused on avoiding error monitoring and periodic updates in order to be widely used in practical problems.

Gama and Castillo (2004) claim that a significant classification error rate increase suggests a drift in the class distribution. In order to detect this type of drift, our method is inspired by drift detectors based on error monitoring such as the *Drift Detection Method* (DDM) (Gama and Castillo 2004). These methods detect drifts based on the prequential error rate motivated by the *probably approximately correct* (PAC) learning model (Mitchell 1997). The prequential error (Dawid and Vovk 1999) is the average error calculated in an online way after the prediction of each incoming example. PAC relies on the assumption that, if the distribution of the examples is stationary, the prequential error of the learning algorithm will decrease as the number of examples increases. Thus, an increase of the prequential error suggests a concept drift, leading the current model to be outdated.

Since we intend to deal with unlabeled data, we propose a strategy to simulate the classification error to be monitored. For this, we create an ensemble of classifiers to make online predictions. The literature concerning data stream problems points out that classifier ensembles may handle concept drift more efficiently than single classifiers (Pinage et al. 2016). Tsymbal et al. (2008) investigated three different types of dynamic integration of classifiers. Their results indicated that dynamic selection techniques presented the best performance when dealing with concept drifts. In our method, for each incoming example, we assume the ensemble prediction as its *true* *label*. In order to allow the ensemble to be most likely correct for classifying each new sample individually, we employ dynamic classifier selection, which is defined as a strategy that assumes each ensemble member as an expert in some regions of competence (De Almeida et al. 2016). In dynamic selection, a region of competence is defined for each unknown instance individually and the most competent classifier for that region is selected to assign the label to the unknown instance.

Our method is divided into three modules: (1) ensemble generation; (2) dynamic classifier selection; and (3) drift detection. The first module is focused on generating an online ensemble of classifiers. The second module is intended to select the most competent ensemble member to classify each incoming example. Finally, the third module is designed to detect drift. Assuming the prediction provided by module 2 as the true label, a drift detector is applied so as to monitor a pseudo-error. The prediction provided by module 2 is also used to update every ensemble member incrementally. Hence, it generates a process of ensemble self-training. Then, when a number n of members detects a drift, all members are updated.

Experiments are conducted using artificial and real datasets focusing on handling abrupt and gradual drifts. The ensemble of classifiers is created using online bagging, while Hoeffding Trees are the base classifiers. *Dynamic Classifier Selection with Local Accuracy* (DCS-LA) (Woods et al. 1997), and *Dynamic Classifier Selection based on Multiple Classifier Behavior* (DS-MCB) (Giacinto and Roli 2001) are the selection strategies used at the dynamic selection module. To select the expert member, these methods need a labeled validation dataset to evaluate the performance of each member on a local region defined by the similarity between the current example and the samples contained in the validation dataset.

Our experiments show that our method is able to detect concept drift and to update the system only after detection, while keeping low prequential error rates, even when small amounts of labeled data are used. In addition, our method is flexible so that it can be adjusted to different dynamic selection methods and to different drift detectors based on statistical process control.

The main contribution of this paper is the estimate of prequential pseudo-error that is used to update ensemble members incrementally and to detect drifts. In this way, we may handle practical problems with no fully labeled dataset available, even using supervised detection methods.

This paper is organized as follows. Related work is detailed in Sect. 2. The proposed method is explained in Sect. 3. Experiments and results are discussed in Sect. 4. Finally, in Sect. 5, conclusions and future work are provided.

2 Related work

The common characteristic of methods proposed to handle concept drift in data streams available in the literature is *online learning*. Minku and Yao (2012) present the following definition for online learning algorithms: methods that process each training example once, without storing it or reprocessing it. The decision model makes a prediction when an example becomes available, allowing the system to learn from the example and to update the learning model. These online methods may detect drifts explicitly or implicitly, also called active or passive approaches, respectively. In this paper, the active-based methods are assumed to be drift detectors, while the passive-based methods are called blind methods, which is another possible categorization.

Drift detectors employ statistical tests to monitor whether or not the class distribution is stable over time and to reset the decision model when a concept drift is detected. DDM (Gama and Castillo 2004), *Early Drift Detection Method* (EDDM) (Baena-Garcıa et al. 2006) and *Detection Method Using Statistical Testing* (STEPD) (Nishida and Yamauchi 2007) are common drift detectors. They all monitor the prequential error performance in order to detect drifts. The decreasing of performance measurement is divided into two significant levels: warning and drift levels. These two levels automatically define two thresholds. Beyond the warning level, a new decision model is created and updated using the most recent examples. Thus, it is assumed that a concept drift takes place when the prequential error reaches the drift level. In this case, a new decision model replaces the current one. It is important to mention that these drift detectors receive the incoming data in a stream and are based on single classifiers, which are reset only after concept drift detections. In addition, these three methods are based on supervised online learning.

In terms of blind methods, they update their knowledge base by adding, removing or updating classifiers periodically, even when a drift does not occur. Many blind solutions use an ensemble of classifiers, such as *Streaming Ensemble Algorithm* (SEA) (Street and Kim 2001), *Dynamic Weighted Majority* (DWM) (Kolter and Maloof 2007), and Learn++NSE for *NonStationary Environment* (Muhlbaier and Polikar 2007). Even though performance monitoring is not conducted for detecting drift, these methods need to monitor accuracy to adjust the weights of the classifiers in order to update them.

For instance, the Learn++NSE method trains a new classifier for every incoming chunk of data. Thus, a performance monitoring mechanism is conducted using new and old data. Then, the average error is combined by majority voting to determine a voting weight to each classifier. In this way, the poorest performing classifier on the current concept is discarded. Another example is the SEA method, which trains a new classifier for every incoming chunk of data and increases or decreases the quality of its classifiers based on accuracy. This method removes the oldest member in a fixed-size ensemble.

Besides blind methods, ensemble of classifiers are also used in active approaches. An interesting example of this category is called *Diversity for Dealing with Drifts* (DDD), proposed by Minku and Yao (2012). DDD updates the system only after drift detection. This is a drift reaction method which uses ensemble diversity to adapt to different types of drifts. This method processes each example at a time and maintains ensembles with different diversity levels so as to deal with concept drift. DDD divides the ensemble into two subsets of high/low diversity classifiers. Before drift detection, the high/low diversity ensembles are generated and DDD uses a drift detector to monitor the accuracy of the low diversity ensemble. Then, after drift detection, the low/high diversity ensembles previously generated are assigned as old high/low diversity ensembles and the method reactivates the before-drift status to create new low/high diversity ensembles. DDD uses knowledge from the old concept (old high diversity ensemble) to learn the new concept.

All these methods are supervised and handle drifts based on accuracy either at the detection phase or at the reaction phase. It is interesting to observe that accuracy monitoring and blind update may be the main disadvantage of these methods, since the accuracy monitoring implies that the drift detector must depend on decreasing the system performance and on fully labeled data, which is not often suited for practical problems. In terms of blind approaches, they may lead to unnecessary updates, increasing the computational cost as a consequence.

In order to deal with unlabeled data, some unsupervised drift detectors were proposed, such as the *Drift Novelty Detection* method (which we call DND) (Fanizzi et al. 2008) or *Dissimilarity-based Drift Detection Method* (DbDDM) (Pinage and dos Santos 2015). DND and DbDDM are both processed in online mode based on dissimilarity measures to assign the current example to previous generated clusters. The difference between DND and DbDDM relies on the detection module: (1) DND calculates a maximum distance between examples to establish a decision boundary for each cluster. The union of the boundaries of all clusters is called the global decision boundary. The new unknown incoming examples that fall outside this global decision boundary are assumed as drifts or novelties; (2) DbDDM compares a dissimilarity prediction to a classifier prediction (assumed as true label) to monitor the pseudo prequential error to detect drifts by statistical tests.

Even though unsupervised drift detectors are assumed to be the solution for dealing with fully unlabeled data, these methods must assume some structure to the underlying distribution of data, must store clusters in a short-term memory and their decisions depend on similarity/dissimilarity measures. All these aspects may compromise system performances in online practical problems. In order to avoid storing examples and to provide more confident decision models, semi-supervised methods may be an interesting alternative, since they usually allow working with a small amount of labeled data and a large amount of unlabeled data.

Semi-supervised methods have been proposed in several machine learning research domains, such as deep learning (Pezeshki et al. 2016). It is, therefore, not surprising that this idea was already investigated in the literature of concept drift. Wu et al. (2012) proposed SUN (Semi-supervised classification algorithm for data streams with concept drifts and UNlabeled data). Basically, SUN divides the streaming data into two sets: training and testing datasets. First, at the training phase, the method builds a growing decision tree incrementally and generates concept clusters in the leaves using labeled information. The unlabeled data are labeled according to the majority-class of their nearest cluster. Then, SUN suggests a concept drift based on the deviation measured in terms of distance, radius, etc, between old and new concept cluster. Secondly, at the test phase, samples are evaluated using the current decision tree. However, SUN searches for concept drifts only during the training phase, where samples contained in the training set are assigned to true labels or to pseudo-labels. During the test phase, SUN assumes that all concepts described in the data stream are already known. Moreover, in terms of drift detection, the results attained by SUN are worse than its baseline, since SUN produces more false detections, missing detections, and larger delays of drift reactions.

Another semi-supervised strategy for dealing with concept drift in the context of data streams was proposed by Kantardzic et al. (2010). Here, however, instead of using single classifiers, the authors employed an ensemble of classifiers. Their online method calculates similarity measures to select suspicious examples, which are assumed to belong to the new concept. Suspicious examples are samples that must be labeled in order to improve the accuracy of the current classifier. The incoming streaming examples are clustered in the *current distribution* and the suspicious examples are put together to form the *new regions*. This method does not build a new classifier for the ensemble periodically and it also does not detect the moment when drifts occur. The system update works as follows: when the number of examples assigned to a cluster *new region* reaches the predefined minimum number of examples, the ensemble requests a human expert to label them. After such a labeling process, the ensemble builds a new member classifier and removes the oldest one.

One recent semi-supervised method based on ensemble classifiers is called SAND (Semi-Supervised Adaptive Novel Class Detection and Classification over Data Stream) proposed by Haque et al. (2016). SAND maintains a window *W* to monitor estimates of classifier confidence on recent data instances. When classifier confidence decreases significantly, such a behavior suggests a concept drift. Focusing on updating the ensemble, SAND uses the most recent chunk of data and selects some instances (by classifier confidence) to be labeled and to be included in a dataset for training a new model. This new model replaces the oldest one. In this way, SAND deals with concept drift by continuing to update the ensemble with the most recent concept.

The semi-supervised methods mentioned in this section aim to update the systems so as to deal with evolving data streams. However, they do not present high performance on detecting drifts at the right moments when they occur. The literature indicates that the best strategy for detecting true drifts relies on error monitoring, commonly used by supervised drift detectors. In addition, both semi-supervised methods based on ensemble classifiers make their predictions by majority voting fusion function. Classifier fusion assumes error independence among the ensemble's component members. This means that the classifier members are supposed to misclassify different patterns. In this way, the combination of classifier members' decisions would improve the final classification performance. However, when the condition of independence is not verified, there is no guarantee that the combination of classifiers will outperform single classifiers (Kuncheva et al. 2002).

In order to avoid the assumption of independence of classifier members, methods for classifier selection have been used as an alternative to classifier fusion. Classifier selection is traditionally defined as a strategy that assumes each ensemble member as an expert in some regions of competence. The selection is called dynamic or static as to whether the regions of competence are defined during the test or the training phase respectively. DCS-LA is a popular method for dynamic classifier selection based on local accuracy estimates. The local regions are defined by k-nearest neighbors (k-NN) over a validation dataset, which is created in an offline mode. The member classifier with the highest local accuracy is selected as the expert to classify the current incoming example.

Dynamic classifier selection strategies have also been used to deal with concept drift. The framework proposed by De Almeida et al. (2016), called DYNSE (Dynamic Selection Based Drift Handler), uses dynamic classifier ensemble selection to choose an expert sub-ensemble to classify an unlabeled incoming instance. DYNSE intends to deal with concept drift by building a new classifier using every most recent incoming batch of data, which is also used to replace the validation dataset in each update. Even though dynamic classifier selection is used, DYNSE is in the category of blind methods and is based on supervised learning, which are drawbacks that we intend to avoid in our methods. Moreover, DYNSE employs a dynamic ensemble selection strategy (KNORA-E: K-Nearest Oracles Eliminate) instead of dynamic classifier selection, such as DCS-LA.

Therefore, in this paper, we propose a method considering four important criteria: (1) it is intuitive that the best moment to update a system to new concepts is right after a concept drift; (2) ensemble methods may improve classification performance (Altınçay 2007; Oza and Russell 2001; Ruta and Gabrys 2007) when compared to single classifiers; (3) diverse classifier ensembles may be the key for the success of classifier ensembles' performance (Minku and Yao 2012); and (4) dynamic classifier selection may help to detect drift with no information about classifiers' errors. The method proposed is based on dynamic classifier selection focused on explicit drift detection for dealing with unlabeled data and ensemble self-training, as detailed in the next section.

3 Proposed method

Our method may be included in the category of drift detectors and semi-supervised methods, since it resets the decision model only when a concept drift is detected in an unsupervised manner and it uses an ensemble of classifiers based on traditional training, as well as a self-training online learning leading to dealing with both labeled and unlabeled data. Our method is divided into three modules: (1) ensemble generation; (2) dynamic classifier selection; and (3) drift detection, as illustrated in Fig. 1.

The first module is focused on generating an online ensemble of classifiers. Even though several techniques for online classifier ensemble generation have been proposed in the literature, in this work we use a modified version (Minku et al. 2010) of online bagging (Oza and Russell 2001), which includes ensemble diversity. In the original online bagging, each classifier member is trained by n copies of each incoming example, where n tends to $Poisson(1)$ distribution. The modified version includes a parameter λ for the $Poisson(\lambda)$ distribution, where higher/lower λ values lead to lower/higher diversity in the ensemble.

Since our method is online and, operates in order to detect drifts on unlabeled data, the second module is intended to select the most competent ensemble member to classify each incoming example. Thus, for each new (unknown) example x_i, one ensemble member is selected to assign a label ($Pred_i$) to it, such as incoming data in a stream. If a drift is detected, the dynamic classifier selection module is updated with a new labeled validation dataset. This is the supervised step of our method.

Finally, the third module is designed to detect drift. Assuming the prediction $Pred_i$ provided by the previous module as the true label of x_i, a drift detector is then applied for each ensemble member, so as to monitor individual pseudo-errors, i.e. $Pred_i$ is compared to the output provided by each classifier member. $Pred_i$ is also used to

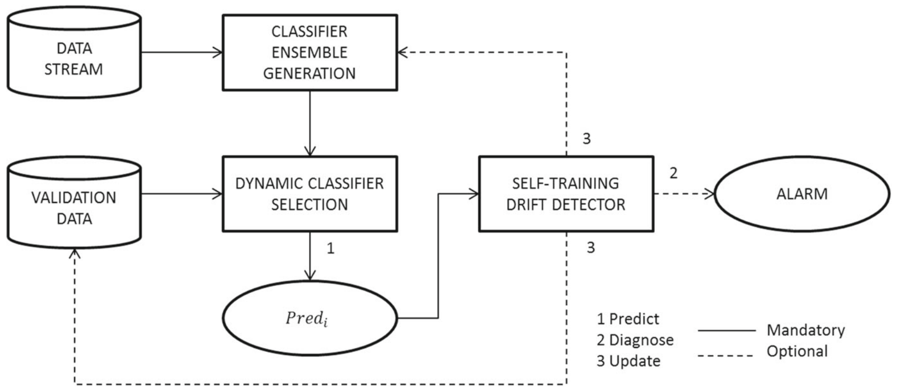

Fig. 1 Overview scheme of the proposed method

update every ensemble member incrementally as a self-training process. When one classifier detects a drift, all members and the validation dataset are updated.

In order to better describe this method, we divided it into three sections: ensemble creation, selection module, and detection module.

3.1 Ensemble creation

Motivated by the advantages of classifier ensembles highlighted in the literature concerning data stream problems, our method is intended to construct a diverse ensemble. In addition, the ensemble of classifiers must be designed to allow incremental learning by self-training. Hence, in the ensemble creation module, we employ a modified version of online bagging (Minku et al. 2010), summarized in Algorithm 1, as follows. The first m incoming examples are manually labeled to start the online bagging training process, i.e. the higher the number of labeled instances, the higher the classification performance during a stable concept. For each classifier member, each training example is presented K times (Algorithm 1, line 5), where K is defined by the $Poisson(\lambda)$ distribution (Algorithm 1, line 3). It is important to observe that low λ values lead to high diversity among ensemble members.

After such a supervised online bagging training with the first m incoming examples, online bagging keeps updating the ensemble incrementally, as it is expected. Here, however, the incoming examples after m are no longer manually labeled. Since the incoming data are unlabeled, the next module (Selection Module) provides a pseudo true label (predicted by the expert member) for every incoming example, i.e. this prediction is considered “correct” for comparison and incremental updating of each member classifier. Therefore, online bagging is adapted to work as a self-training method. It is important to mention that online bagging is initially supervised in order to generate an accurate ensemble. Then, it is incrementally updated by self-training.

```
Data: incoming example: $x$; ensemble: $c$; ensemble size: $T$
for each $x$ do
    for $j \leftarrow 1$ to $T$ do
        $K \leftarrow$ Poisson($\lambda$);
        while $K > 0$ do
            $c_j \leftarrow$ IncrementalUpdate($c_j, x$);
            $K \leftarrow K - 1$
        end
    end
end
```

Algorithm 1: Online ensemble creation.

The ensemble generated has a small fixed size of T diverse members. The base classifiers used in this paper are Hoeffding Trees. This choice is especially due to the fact that bagging is used to improve the performance of unstable algorithms, such as decision trees (Breiman 1996).

3.2 Selection module

It was mentioned in the introduction that classifier selection is defined as a strategy that assumes each ensemble member as an expert in some regions of competence. Thus, rather than combining all T classifiers generated by online bagging using a fusion function, dynamic selection chooses, in an online way, a winning classifier (assumed as an oracle) to assign the label to each x_i incoming sample. Among several different dynamic classifier selection methods reported in the literature, in this work, two methods are investigated: *Dynamic Classifier Selection with Local Accuracy* (DCS-LA) (Woods et al. 1997) and *Dynamic Classifier Selection based on Multiple Classifier Behavior* (DS-MCB) (Giacinto and Roli 2001). These methods are based on the assumption that the best-performing classifier over the k-nearest neighbors (local region) obtained from a validation dataset surrounding x_i is the most confident classifier to label it individually.

Hence, the second module of our method is designed to work with a validation dataset, which is created with a small number of labeled examples. It is important to mention that the initial validation dataset is generated in an offline mode and maintained until a drift detection. Only after, the current validation dataset is discarded and replaced by a new one generated in online mode using examples collected during the warning level by the drift detector in the next module.

DCS-LA is a popular dynamic classifier selection method with two different versions: *Overall Local Accuracy* (OLA) and *Local Class Accuracy* (LCA). The OLA is computed as the number of neighbors of x_i correctly classified by each classifier member c_j (Algorithm 2, line 2). Then, the member (c^*) with the highest OLA (Algorithm 2, line 6) is selected to classify x_i (Eq. 1).

In the LCA version, for each classifier member c_j, it is computed the number of neighbors of x_i for which c_j has correctly assigned class ω (Algorithm 2, line 2), but considering only those examples whose label are the same class predicted for x_i. In this way, the best member classifier for the current example (c_x^*) is the one with the highest LCA (Eq. 2).

$$c^*_x = \arg max_j(OLA_j) \tag{1}$$
$$c^*_x = \arg max_j(LCA_j)|\omega_t = \omega_x \tag{2}$$

DS-MCB is a behavior-based method which uses a similarity function to measure the degree of similarity (Sim) of the output of all member classifiers. First, it is computed the vector MCB_{x_i} of class labels assigned by all member classifiers to the incoming example x_i. Then, it is also computed the vector $MCB_{k,t}$ of class labels assigned by all member classifiers to each k_t in the *current local region* (kNN). The method computes the similarity between MCB_{x_i} and each $MCB_{k,t}$ to find a *new local region* (kNN') (Algorithm 2, line 2), i.e., the examples in kNN with the MCB most similar to MCB_{x_i} (Eq. 3). Finally, kNN' is used to select the most accurate classifier by overall local accuracy (OLA) (Algorithm 2, line 4), such as DCS-LA, to classify x_i.

$$kNN' = kNN' \cup k_t|Sim(MCB_{x_i}, MCB_{k,t}) > \theta \tag{3}$$

Finally, in this module, the prediction ($Pred_i$) assigned by the selected classifier is assumed as the true label of x_i and used in the next module (Algorithm 2, line 7), whatever the dynamic selection strategy used to select the most confident classifier to label x_i. Algorithm 2 summarizes module 2 of our method.

```
Data: incoming example: $x$; ensemble: $c$; ensemble size: $T$; labeled validation dataset: $S$; member
      performance: $pmc$
1 for each $x$ do
2     $kNN$ ← kNearestNeighbours($S$, $x$);
3     for $j$ ← 1 to $T$ do
4         $pmc_j$ ← evaluate($c_j$, $kNN$); %performance of each member
5     end
6     $expert$ ← selectClassifier($c$, $pmc$); %member with the best performance
7     $pseudoLabel$ ← predict($expert$, $x$); %this prediction is assumed as true label
8 end
```

Algorithm 2: Selection module algorithm

3.3 Detection module

In this third module, we apply a drift detector for each ensemble member. However, since most of the drift detectors are designed to cope only with labeled data, including the drift detectors investigated in this paper, these methods must be tailored to support unlabeled data. In order to accomplish this requirement, in our method we assume that $Pred_i$ is the x_i's true label. Thus, for each individual drift detector j, a pseudo error is measured (Algorithm 3, line 3) by comparing $Pred_i$ to the class label assigned by its classifier c_j to x_i. As a consequence, our assumption is that any drift detector based on statistical process control (Gama et al. 2014) available in the literature may be used to detect drift in the unsupervised detection module of our proposed method.

In this paper, we have tailored two supervised drift detectors to work as unsupervised methods: DDM and EDDM, as follows. A pseudo prequential error rate is monitored for each classifier member using $Pred_i$. For instance, given an ensemble c composed of 10 classifiers c_j, where $j = 1 \ldots 10$, there are 10 pseudo prequential error rates to be monitored simultaneously and individually, i.e. 10 local drift detection processes are conducted. Thus, the set of pairs $Pred_i, x_i$, which rely on the local warning level (denoted by S_j) is stored to further update its respective alternative classifier member (Algorithm 3, line 13).

As mentioned before, both DDM and EDDM are divided into warning and drift levels. In terms of DDM, for each member classifier, warning and drift levels are reached if conditions (4) or (5) are satisfied, respectively. The p value represents the pseudo error rate of each classifier member c_j, while s denotes its standard deviation. The registers p_{min} and s_{min} are set during the training phase, and are updated if, after each incoming example (i), the current register $p_i + s_i$ is lower than $p_{min} + s_{min}$.

$$p_i + s_i < p_{min} + 2s_{min}. \tag{4}$$

$$p_i + s_i < p_{min} + 3s_{min}. \tag{5}$$

On the other hand, when using EDDM's statistical process control, our method calculates the distance (p') between two consecutive pseudo errors (Algorithm 3, line 3) and their standard deviation (s'), and stores the maximum values of p' and s' to register the point where the distance between two errors is maximum ($p'_{max} + 2 * s'_{max}$). According to Baena-Garcıa et al. (2006), the warning level is reached when the Eq. (6) is lower than α (here set to 0.95), and the drift level is reached when the same Eq. (6) is lower than β (here set to 0.9).

$$\left(p'_i + 2s'_i\right)/\left(p'_{max} + 2s'_{max}\right). \tag{6}$$

In this detection module, when a number of n member classifiers reach their drift level, the whole system is updated (Algortithm 3, line 8) and all parameters (such as prequential error and standard deviation) are reset (Algorithm 3, line 6), i.e., each member classifier is updated using its own subset S_j (Algorithm 3, line 14). However, it is important to mention that only the subset S_j of the classifier member which triggered the drift detection is labeled (Algorithm 3, line 5) and used as a new validation dataset in order to reduce the labeling processing time. For the remaining classifiers c_j, the label of each x_i contained in its S_j is assumed to be its respective $Pred_i$.

In addition, all members are also updated incrementally based on $Pred_i$ (Algorithm 3, line 18) due to online bagging self-training, as discussed in the beginning of this section. We can observe all the steps executed by the detection module in Algorithm 3.

3.4 Algorithm complexity analysis

Our proposed method presents two configurable modules, since it is possible to change the dynamic selector (selection module) and the statistical test for process control (detection module). In this way, in order to analyze the whole algorithm complexity, we assume the simplest Big-O notation for each module. Taking into account that our

```
Data: incoming example: $x$; current ensemble: $c$; alternative ensemble: $R$; ensemble size: $T$;
      validation dataset: $S$
for each $x$ do
    for $j \leftarrow 1$ to $T$ do
        $p_j \leftarrow$ evaluate($c_j, x$);
        if $p_j$ reaches DriftLevel then
            labeling($S_j$); %new validation dataset for selection module
            $p_j \leftarrow 1$;
            for $j \leftarrow 1$ to $T$ do
                replace($c_j, R_j$);
            end
            break; %go to the next example
        end
        else if $p_j$ reaches WarningLevel then
            $S_j \leftarrow$ add($x$);
            IncrementalUpdate($R_j, x$)
        end
        else
            reset($S_j$);
            IncrementalUpdate($c_j, x$);
        end
    end
end
```

Algorithm 3: Detection module algorithm

method is online, each incoming example is processed only once, which provides a notation $O(n)$, where n is the size of the stream (total number of incoming examples).

However, the method is based on ensemble classifiers and each incoming example has to be processed by each member. It means that each algorithm module provides a notation $O(T)$, where T is the size of the ensemble (number of classifiers). Finally, the three algorithm modules are independent of each other and they run only once for each incoming example. Therefore, we may conclude for the whole algorithm a final notation $O(nT)$.

4 Experiments

The objective of the experiments conducted in this paper is to measure the performance of our semi-supervised drift detection method, as well as to compare it to two baselines: SAND (semi-supervised) and DDM (supervised). In order to use the best version of our method to be compared to the baselines, we investigate different versions by trying different combinations of dynamic selection methods and drift detectors.

As mentioned before, we have used online bagging as an ensemble creation strategy and Hoeffding Trees as base learning algorithms. The drift detectors were implemented in Matlab 7.10 over Windows 7 in a machine based on Intel Core i7 3520M@2,90 GHz processor using WEKA implementations of the learning algorithms. First, however, we will present details related to the databases investigated in our experiments.

4.1 Databases

The main aspect taken into account to choose the databases for the experiments is the type of concept drifts we intend to deal with. Accordingly, we observed that most of the methods available in the literature handle abrupt and gradual drifts, while some methods handle recurrent concepts. In our experiments, we also intend to handle these types of drifts. In this way, we have used some artificial datasets investigated by Gama and Castillo (2004), Baena-Garcıa et al. (2006), Nishida and Yamauchi (2007), and Minku and Yao (2012). The artificial databases are described below:

SINE1 (Abrupt concept drift, noise-free examples): Classification is positive if a point lies below the curve given by $y = sin(x)$, otherwise it is negative. After concept drift, the classification is reversed.

LINE (Abrupt concept drift, noise-free examples): This dataset presents two relevant attributes, both assuming values uniformly distributed in [0,1]. The classification function is given by $y = -a_0 + a_1 x_1$, where a_0 can assume different values to define different concepts. In this dataset, there are two concepts with high severity.

CIRCLE (Gradual concept drift, noise-free examples): This dataset also presents two relevant attributes, which have values uniformly distributed in [0,1]. Examples are labeled according to a circular function, i.e. an example is labeled positive if it is inside the circle, otherwise its label is negative. The gradual drift occurs due to displacing the center of the circle and growing its radius. This dataset presents four contexts defined by four circles:

- center: [0.2,0.5], [0.4,0.5], [0.6,0.5], [0.8,0.5]
- radius: 0.15, 0.2, 0.25, 0.3

SINE1G (Very slow gradual concept drift, noise-free examples): This dataset presents the same classification function as SINE1, but there is a transition time between old and new concepts. The old concept disappears gradually and the probability of selecting an example from the new concept becomes higher after the transition time.

With the exception of SINE1, which presents changes only in terms of $P(x, \omega)$, the remaining artificial datasets present changes in both $P(x, \omega)$ and $P(x)$. Moreover, all these artificial datasets have two classes, are balanced and each batch of 1000 examples represents a concept, except for SINE1G, which is composed of 2000 examples for each concept and 1000 examples between the transition from one concept to another.

Besides these artificial datasets, we have used real datasets in our experiments. It is important to mention that, in the context of real data, we do not know when the drifts occur and which type of drifts occur. The real datasets are described below:

ELEC2 According to Gama and Castillo (2004), this dataset is composed of 45312 instances dated from 7 May 1996 to 5 December 1998. Each example of the database refers to a period of 30 minutes and has 5 fields: day of week, time stamp, NSW (New South Wales) electricity demand, Vic electricity demand, the scheduled electricity transfer between states and the class label. The latter identifies the change of the price related to a moving average of the last 24 h.

LUXEMBOURG It was constructed using European Social Survey 2002–2007. The task focuses on classifying a subject with respect to the internet usage, whether it is high or low. Each data sample is represented as a 20-dimensional feature vector. These features were selected as general demographic representation. The dataset is balanced 977 + 924 samples for each class. More details about the dataset creation and data source are found in Zliobaite (2011).

KDDCUP99 This is the dataset used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining. This dataset contains simulated intrusions in a military network traffic based on tcpdump data collected during 7 weeks. KDDCUP99 has approximately 4,900,000 examples which contain 41 features and are labeled as "bad"(intrusions or attacks) or "good"(normal) connections.

In the next section, we present experimental results obtained when setting up parameters such as: dynamic classifier selection strategy and drift detector.

4.2 Comparison of different versions of the proposed method

It is important to emphasize that the performance of the learning algorithm is not the aim of these experiments, but rather its capability in detecting and reacting to drifts quickly. Even so, the error rates presented for all experiments described in this section are real prequential errors, i.e., the final classification performance.

The objective of this comparison is to select the best version of our method, in terms of performance, to be compared to the baselines SAND and DDM. In addition, experiments were conducted for fine-tuning parameters such as kNN (set to 5), number of initial labeled examples m (set to 30), and λ (set to 0.2). All these parameters were chosen empirically after experiments by taking into account the best performance (accuracy and computational cost) attained in all datasets.

4.2.1 Experiments with artificial datasets

First, we analyze the results obtained on experiments using artificial datasets, which allow getting true, false and missing detections, as well as average detection delays, since it is well known the right moment when the drifts occur.

In Fig. 2, we can observe the prequential error attained when using the combinations DCS-LA + DDM (solid lines) and DCS-LA + EDDM (dotted lines). These plots illustrate datasets presenting abrupt drifts, such as SINE1 and LINE, and gradual drifts, such as CIRCLE and SINE1G.

Both versions using DCS-LA as dynamic selection strategy present results quite similar on abrupt drift datasets SINE1 and LINE in terms of detecting drifts at the right moment and keeping low prequential error. For the gradual drift datasets CIRCLE and SINE1G, it is more difficult to identify a pattern of behavior due to the fact that there are drift detections at the right moment, but detections with long delays and detections before drift are also verified. Even so, both versions keep low prequential error rates when new concepts appear. In all experiments, delays in the drift detections are verified, but we may point out that DCS-LA + EDDM detects drifts earlier than DCS-LA + DDM.

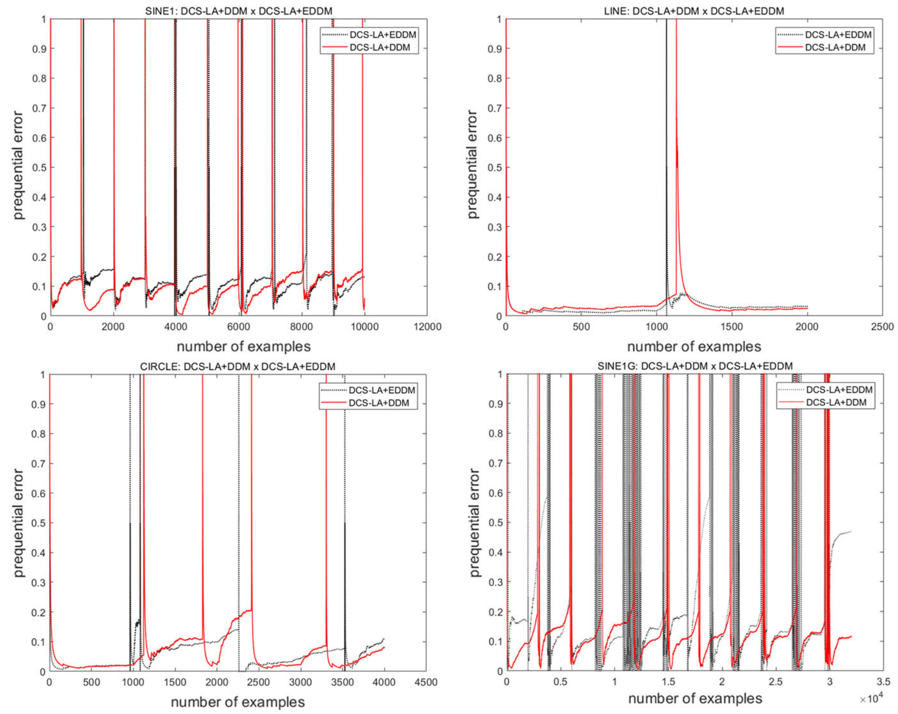

Fig. 2 Prequential error of versions DCS-LA + DDM (solid lines) and DCS-LA + EDDM (dotted lines) on artificial datasets. Top Left: SINE1; Top Right: LINE; Bottom Left: CIRCLE; Bottom Right: SINE1G

Figure 3 shows plots of the prequential error in artificial datasets obtained when using the dynamic selection method DS-MCB combined with DDM (solid lines) and EDDM (dotted lines). For the abrupt drift datasets (SINE1 and LINE), both versions with DS-MCB detected drift at the right moment and decreased the prequential error rate. However, DS-MCB + EDDM outperformed DS-MCB + DDM since it presents a smaller number of false detections on SINE1 and earlier reaction on LINE datasets. For CIRCLE, both versions missed detections. Finally, for SINE1G dataset, DS-MCB + EDDM did not detect the last drifts.

If we compare the two versions with DS-MCB to the versions with DCS-LA, some observations can be made from these results: (1) the proposed method using DCS-LA combined with both drift detectors was able to decrease the prequential error rate while learning the new concept before it ended, even with no access to the whole data labels; (2) the two versions using DS-MCB detect all the abrupt drifts but fail to detect some gradual drifts. Table 1 summarizes the comparison among all four versions of the proposed method in terms of accuracy, total number of true detections, false detections, and missing detections for each dataset. The items presented in Table 1 are defined as follows:

– accuracy: the percentage of examples correctly classified in the whole dataset;

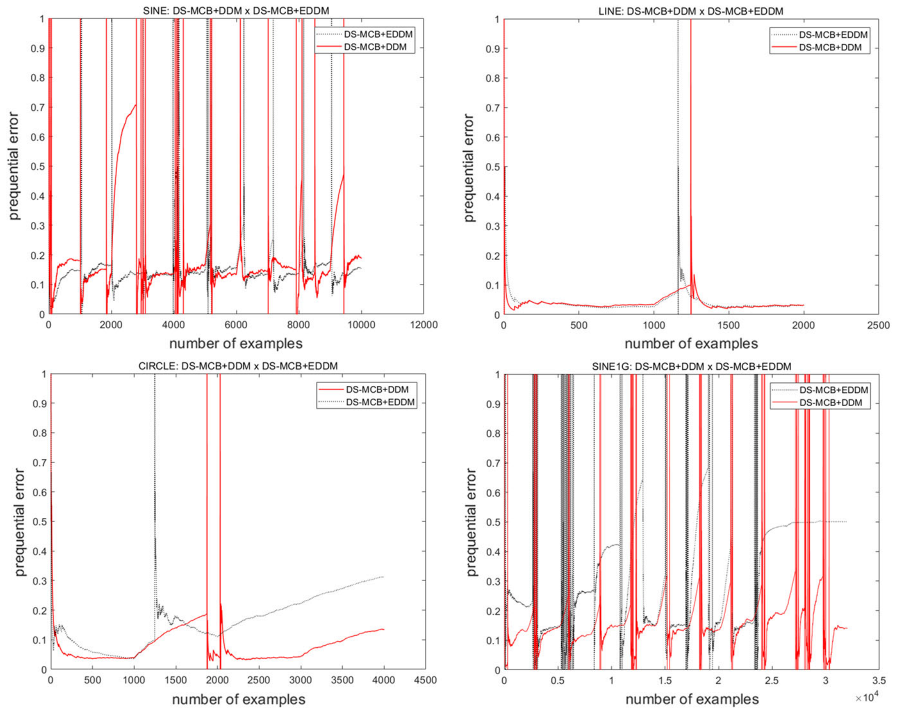

Fig. 3 Prequential error of versions DS-MCB + DDM (solid lines) and DS-MCB + EDDM (dotted lines) on artificial datasets. Top Left: SINE1; Top Right: LINE; Bottom Left: CIRCLE; Bottom Right: SINE1G

- average detection delay: the average number of examples between drift occurrence and true drift detection for all concept drifts. For SINE1G, detections during all transition periods are not considered delayed;
- true detections: drift detections at the right moment. For abrupt drifts, we consider only the first detection after drift occurrence; for gradual drifts, we consider all the detections occurring in the transition period from one concept to another;
- false detections: for abrupt drifts, we consider all detections indicated after a true detection in the same concept; for gradual drifts, we consider all detections performed after the transition period, for no transition periods, we employ the same rule used for abrupt drifts;
- missing detections: concept drifts not detected by the method (false negatives).

In Table 1, the best results for each dataset are indicated in bold. From this table, it is possible to observe that only versions including DS-MCB present missing detections on datasets with gradual drifts. This result may be due to the similarity function, which provides wrong new local regions in the transition period. On the other hand, the two versions involving DCS-LA are able to detect all drifts in all datasets. In the next section, we present experimental results to verify whether or not these results are observed considering real datasets.

Table 1 Classification accuracy (acc), average detection delay (delay), total number of true detections (TD), false detections (FD) and missing detections (MD) of different versions of our method in each artificial dataset

Method	Dataset	acc(%)	Delay	TD	FD	MD
DCS-LA + DDM	SINE1	**87.91**	27	**9**	2	**0**
	LINE	94.80	130	**1**	**0**	**0**
	CIRCLE	90.33	280	**3**	1	**0**
	SINE1G	**81.13**	**0**	35	**2**	**0**
DCS-LA + EDDM	SINE1	85.23	**25**	**9**	**1**	**0**
	LINE	**96.40**	**65**	**1**	**0**	**0**
	CIRCLE	**91.10**	287	**3**	1	**0**
	SINE1G	77.70	**0**	49	22	**0**
DS-MCB + DDM	SINE1	74,64	70	**9**	11	**0**
	LINE	92.60	246	**1**	**0**	**0**
	CIRCLE	84.42	**32**	1	1	2
	SINE1G	75.26	**0**	**33**	12	**0**
DS-MCB + EDDM	SINE1	80,14	65	**9**	4	**0**
	LINE	94.30	161	**1**	**0**	**0**
	CIRCLE	75.35	248	1	**0**	2
	SINE1G	63.43	**0**	36	10	2

4.2.2 Experiments with real datasets

In these datasets, we do not know where the changes occur and which type of changes exist. Therefore, we are not able to measure true detections, false detections, or missing detections. Thus, we only highlight the behavior of the prequential error and the total number of detections on each dataset. The parameter settings are the same as used in experiments on artificial datasets.

Figure 4 presents the prequential error attained by the two versions of our method composed by DCS-LA combined with DDM (solid lines) and EDDM (dotted lines) on real datasets. For ELEC2, both versions detected a high number of drifts, but they kept low prequential error rates. In LUXEMBOURG, each version identified drifts in different moments and both decreased the prequential error rates. In KDDCUP99, we plot the prequential error rate for the 100,000 first examples (10% of whole dataset) in order to provide a better visualization of the result; however, both versions with DCS-LA were able to keep low prequential error rates for the whole dataset presenting a high number of drift detections (Tables 2 and 4).

In Fig. 5, the prequential error rates reached by the two versions of our proposed method generated by combining DS-MCB to DDM (solid lines) and EDDM (dotted lines) on real datasets are shown. In ELEC2, the results are quite similar to the results obtained by the DCS-LA-based versions: several detections followed by low prequential error rates. In LUXEMBOURG, DS-MCB + DDM was able to decrease the prequential error rate after first detection, while DS-MCB + EDDM kept the prequen-

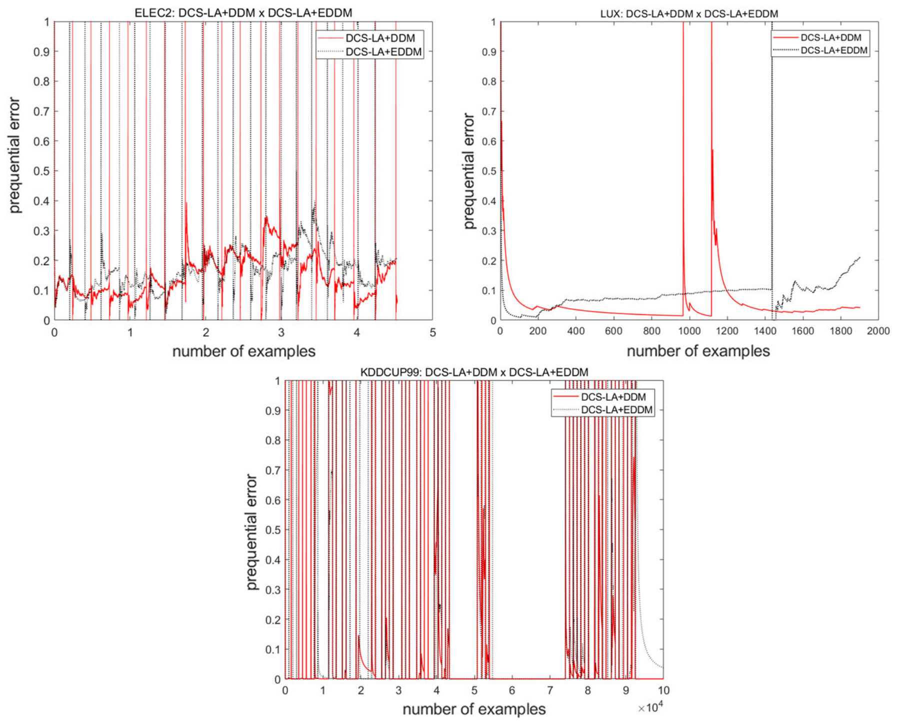

Fig. 4 Prequential error of versions DCS-LA + DDM (solid lines) and DCS-LA + EDDM (dotted lines) on real datasets. Top Left: ELEC2; Top Right: LUXEMBOURG; Bottom: KDDCUP99

Table 2 Classification accuracy (acc) and total number of detections of different versions of our method in each real dataset

Method	Dataset	acc (%)	Detections
DCS-LA + DDM	ELEC2	**84.16**	18
	LUXEMBOURG	**97.53**	2
	KDDCUP99	**95.66**	11,610
DCS-LA + EDDM	ELEC2	83.03	20
	LUXEMBOURG	87.01	1
	KDDCUP99	94.18	11,790
DS-MCB + DDM	ELEC2	81.12	17
	LUXEMBOURG	83.75	2
	KDDCUP99	94.31	11,706
DS-MCB + EDDM	ELEC2	81.45	21
	LUXEMBOURG	76.12	2
	KDDCUP99	90.29	9277

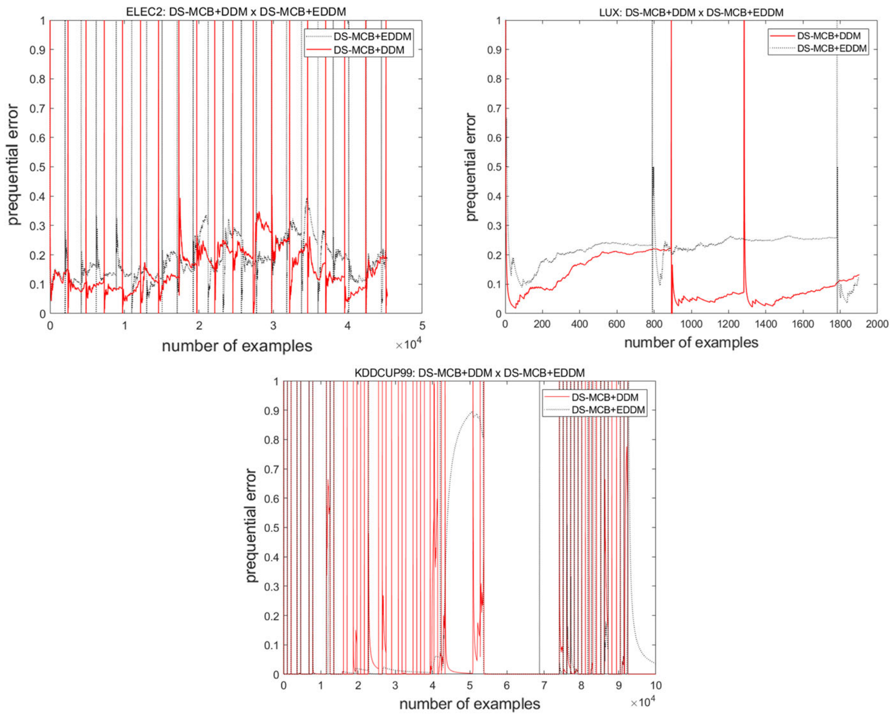

Fig. 5 Prequential error of versions DS-MCB + DDM (solid lines) and DS-MCB + EDDM (dotted lines) on real datasets. Top Left: ELEC2; Top Right: LUXEMBOURG; Bottom: KDDCUP99

tial error stable after detection. In KDDCUP99, both versions kept low prequential error rates, but it is possible to observe that DS-MCB + EDDM provided fewer detections than DS-MCB + DDM, and consequently lower accuracy rate.

Table 2 shows a comparison among the four versions of our method in terms of classification accuracy rates and total number of detections on all real datasets. The same behavior pointed out for the artificial datasets was observed for the real databases investigated: versions using DCS-LA outperformed the DS-MCB-based versions. In terms of accuracy rates, for ELEC2 dataset, DCS-LA + DDM was slightly superior to DCS-LA + EDDM, while the difference between DCS-LA + DDM and DS-MCB-based versions was higher. However, the performances attained by all four versions are quite similar, since they present accuracy rates ranging from 81.12 to 84.16%.

In contrast, these differences are more likely to be significant in experiments with LUXEMBOURG and KDDCUP99. Here, our results indicate that DCS-LA + DDM was able to cope with drifts presented in LUXEMBOURG and KDDCUP99 datasets, since it classified correctly more than 90% of examples. Especially noteworthy is the fact that this high accuracy rate was obtained in a semi-supervised context, with labels partially available. On the other hand, DS-MCB + EDDM produced the lowest accuracy rates in LUXEMBOURG (76.12%) and KDDCUP99 (90.29%). This result may be due to concepts with a small number of examples or to a sensitivity to outliers.

In Table 2, we indicate in bold the best accuracy for each dataset. We do not indicate the best result in terms of number of detections since we do not know when these detections are true or false.

In response to the question posed in the beginning of this section as to which is the best version of our proposed method, DCS-LA + DDM may be deemed to be the best strategy. First, based on the classification accuracy results, this version attained the highest rate in 5 datasets (2 artificial and the 3 real datasets) out of a total of 7 datasets investigated. In addition, DCS-LA + DDM does not present missing detections in any of the artificial datasets. In this way, we choose the version DCS-LA + DDM to be compared to the baselines, due to its best performances in high accuracy rates and true detections.

4.3 Comparison of the proposed method to baselines

In this series of experiments, we intend to compare the best version of our proposed method DCS-LA + DDM to two different baselines. The first baseline is the semi-supervised method SAND and the second baseline is the supervised method DDM, since we also intend to verify whether or not our method is able to deal with concept drifts even when there are no fully labeled datasets available. We choose DDM because it is part of the selected version of our method.

It is important to mention that there is no comparison in terms of prequential error due to the fact that SAND does not provide results in prequential error. Thus, the comparison summarized in Tables 3 and 4 is performed by taking into account other measures such as accuracy, delay, true and false detections, average runtime, etc.

Table 3 Classification accuracy (acc), average detection delay (delay), total number of true detections (TD), false detections (FD), missing detections (MD), the percentage of labeled examples (lbl) and average runtime (time) in each artificial dataset

Method	Dataset	acc (%)	Delay	TD	FD	MD	lbl (%)	Time (s)
DCS-LA + DDM	SINE1	87.91	27	**9**	2	**0**	1.6	219.08
	LINE	94.80	130	**1**	**0**	**0**	0.6	37.55
	CIRCLE	90.33	280	**3**	1	**0**	1.05	90.39
	SINE1G	83.00	**0**	35	2	**0**	0.5	570.04
SAND	SINE1	84.25	95	1	**0**	8	**0.03**	402.38
	LINE	94.90	83	**1**	**0**	**0**	**0.1**	39.01
	CIRCLE	91.40	**15**	**3**	3	**0**	**0.1**	168.59
	SINE1G	**83.12**	**0**	3	**0**	7	**0.02**	1207.88
DDM	SINE1	**96.28**	**11**	**9**	**0**	**0**	100	**53.68**
	LINE	**97.55**	**25**	**1**	**0**	**0**	100	**5.84**
	CIRCLE	**96.80**	32	**3**	**0**	**0**	100	**20.72**
	SINE1G	79.67	**0**	**13**	**0**	**0**	100	**165.94**

Table 4 Classification accuracy (acc), total number of detections, the percentage of labeled examples (lbl) and the average runtime (time) in each real dataset

Method	Dataset	acc (%)	Detections	lbl (%)	Time (s)
DCS-LA + DDM	ELEC2	84.16	18	0.4	452
	LUXEMBOURG	**97.53**	2	6.0	36
	KDDCUP99	95.97	11,679	25.06	50,292
SAND	ELEC2	**94.62**	44	**0.1**	998
	LUXEMBOURG	85.11	1	**1.0**	57
	KDDCUP99	74.23	8555	**1.5**	88, 919
DDM	ELEC2	83.06	128	100	**125**
	LUXEMBOURG	79.80	0	100	**6**
	KDDCUP99	**97.89**	2386	100	**10,754**

4.3.1 Experiments with artificial datasets

For these experiments, we have included the number of labeled examples and the average runtime as evaluation metrics. In addition, Table 3 also presents the classification accuracy, average detection delay, total number of true detections, false detections and missing detections of DCS-LA + DDM, SAND and DDM for each dataset. The best results for each dataset are indicated in bold. Some observations can be made from these results. First, it is possible to observe that DCS-LA + DDM was able to detect all drifts in all datasets, as DDM. In the SINE1G dataset, our method presents a large quantity of true detections in all experiments, but it is important to say that the detections were suggested in the transition period of 1000 examples. Even so, DCS-LA + DDM reached high accuracy rates, as well as both baselines.

The results achieved using DDM were expected, since it is a supervised drift detector. DDM presented the best performance on attaining the highest accuracy rates, with no false detections nor missing detections. However, in terms of accuracy rates, the rates reached by both semi-supervised methods were on average 6% lower than DDM's accuracy rates for SINE1, LINE and CIRCLE datasets, while both semi-supervised methods outperformed DDM in SINE1G dataset. Especially noteworthy is the fact that the semi-supervised methods dealt with only 5% of labeled data, on average, in order to attain these high accuracy rates.

In terms of number of labeled examples, our method selects those examples which rely on warning level, as indicated by statistical tests, while SAND selects those examples based on whether classifier confidence is lower than a threshold. In this way, SAND presents a more robust method to select a reduced number of examples among the most recent examples to be labeled. However, in datasets such as SINE1 and SINE1G, SAND presents more missing detections than true detections. Such a weak result may be due to the classification function of these datasets. Since the classification is reversed after concept drifts, in this case, the classifier confidence will remain high but the predictor will classify data incorrectly. In addition, SAND is strongly dependent on fine-tuning the threshold parameter. In our experiments, we employed $-log(0.05)$ as threshold value, as suggested by the authors (Haque et al. 2016). SAND missed 8 true detections in the SINE1 dataset and 7 true detections in the SINE1G dataset, while in LINE and CIRCLE datasets, SAND achieved lower detection delays.

4.3.2 Experiments with real datasets

As mentioned before, in these datasets we can only evaluate classification accuracy, total number of detections, the percentage of labeled examples used to classify the whole dataset and the average runtime. In order to compare our method to SAND and DDM, we choose again the version DCS-LA + DDM.

In Table 4, we can observe that DCS-LA + DDM presents the highest accuracy rate in the LUXEMBOURG dataset and probably 2 true detections. SAND takes advantage in accuracy on ELEC2, while our method reaches accuracy rates quite similar to DDM. However, in ELEC2, DCS-LA-DDM presents the smallest number of detections, which implies lower computational costs. The same scenario highlighted for the artificial datasets was observed on real datasets, i.e., our method and SAND reach high accuracy rates even using a limited number of labeled examples. In KDDCUP99, our method reaches an accuracy rate quite similar to DDM but using only 25.06% of labeled examples. In addition, for this last dataset, SAND reaches lower accuracy rates related to the other investigated methods. It is also important to mention that both semi-supervised methods outperformed DDM on real datasets, which was not an expected result, since DDM is supervised. The best results of accuracy, average runtime, and percentage of labeled examples for each dataset are indicated in bold.

In terms of runtime, as expected, DDM presents lowest average runtime in all datasets because this method does not take into account the labeling process since it is assumed that labeled data is available. SAND presents the highest average runtime. Despite a very small number of examples that pass by the labeling process, the window monitoring may increase the computational cost. Finally, our method also includes a labeling process for a few examples but using online learning, which may provide an average runtime significantly lower than SAND's.

5 Conclusion

This paper proposes a new semi-supervised method for dealing with concept drift focusing on practical problems that receive data in stream. As a consequence, they do not present fully labeled data. Besides, labeling massive quantities of data is time consuming. Regarding these important observations, our method intends to detect drifts avoiding strategies based on accuracy monitoring and on updating the system constantly.

The proposed method uses an ensemble of classifiers' predictions defined by dynamic classifier selection to assign labels to every incoming example. Based on these assigned labels, we apply a drift detector to every ensemble member in order to detect drifts. All the incremental updates are based on ensemble predictions in an ensemble self-training strategy. The labeled examples employed are only those used to compose a validation dataset, which is useful for the selection module, after drift detection.

Our semi-supervised method presents competitive results when compared to another semi-supervised method (SAND). However, it is important to mention that SAND provides drift detections by monitoring windows composed of the recent examples, and drift reactions by batch learning, while our method is online. In addition, when compared to a supervised method (DDM), our method attained performances quite similar or even better, such as the classification accuracy reached on all investigated real datasets. These results were not expected since DDM is a supervised drift detector.

Given that the proposed method is employed using unlabeled data and avoiding blind updates, the classification error rates and the system reaction delays attained by our method may be assumed to be very promising. However, as future work, we intend to study strategies aiming to decrease the prequential error and to react faster to drifts.

Acknowledgements The authors gratefully acknowledge the financial support granted by PNPD-CAPES (Coordination for the Improvement of Higher Education Personnel) and FAPEAM (Amazonas Research Foundation) for this research through process Number 009/2012 (RHTI-Doutorado).

References

Altınçay H (2007) Ensembling evidential k-nearest neighbor classifiers through multi-modal perturbation. Appl Soft Comput 7(3):1072–1083

Ang HH, Gopalkrishnan V, Zliobaite I, Pechenizkiy M, Hoi S (2013) Predictive handling of asynchronous concept drifts in distributed environments. IEEE Trans Knowl Data Eng 25(10):2343–2355

Baena-Garcıa M, del Campo-Ávila J, Fidalgo R, Bifet A, Gavalda R, Morales-Bueno R (2006) Early drift detection method. In: Fourth international workshop on knowledge discovery from data streams. vol 6, pp 77–86

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Dawid P, Vovk V (1999) Prequential probability: principles and properties. Bernoulli 5(1):125–162

De Almeida PL, Oliveira LS, Britto ADS, Sabourin R (2016) Handling concept drifts using dynamic selection of classifiers. In: 2016 IEEE 28th international conference on tools with artificial intelligence (ICTAI). IEEE, pp 989–995

Fanizzi N, dAmato C, Esposito F (2008) Conceptual clustering and its application to concept drift and novelty detection. In: European semantic web conference. Springer, pp 318–332

Gama J, Castillo G (2004) Learning with local drift detection. In: Advances in artificial intelligence. Springer, Berlin/Heidelberg, vol 3171, pp 286–295

Gama J, Zliobaite I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. ACM Comput Surv (CSUR) 46(4):44

Giacinto G, Roli F (2001) Dynamic classifier selection based on multiple classifier behaviour. Pattern Recognit 34(9):1879–1881

Haque A, Khan L, Baron M (2016) Sand: semi-supervised adaptive novel class detection and classification over data stream. In: THIRTIETH AAAI conference on artificial intelligence

Huang S (2008) An active learning method for mining time-changing data streams. In: 2008 Second international symposium on intelligent information technology application. IEEE, vol 2, pp 548–552

Kantardzic M, Ryu JW, Walgampaya C (2010) Building a new classifier in an ensemble using streaming unlabeled data. In: International conference on industrial, engineering and other applications of applied intelligent systems. Springer, pp 77–86

Kmieciak M, Stefanowski J (2011) Handling sudden concept drift in enron messages data stream. Control Cybern 40(3):667–695

Kolter Z, Maloof M (2007) Dynamic weighted majority: an ensemble method for drifting concepts. J Mach Learn Res 8(Dec):2755–2790

Kuncheva L, Skurichina M, Duin R (2002) An experimental study on diversity for bagging and boosting with linear classifiers. Inf Fusion 3(4):245–258
Minku L, Yao X (2012) Ddd: a new ensemble approach for dealing with concept drift. IEEE Trans Knowl Data Eng 24(4):619–633
Minku L, White A, Yao X (2010) The impact of diversity on online ensemble learning in the presence of concept drift. IEEE Trans Knowl Data Eng 22(5):730–742
Mitchell T (1997) Machine learning. McGraw-Hill Higher Education, New York
Muhlbaier M, Polikar R (2007) An ensemble approach for incremental learning in nonstationary environments. In: International workshop on multiple classifier systems. Springer, pp 490–500
Nishida K, Yamauchi K (2007) Detecting concept drift using statistical testing. In: International conference on discovery science. Springer, pp 264–269
Oza N, Russell S (2001) Experimental comparisons of online and batch versions of bagging and boosting. In: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 359–364
Pezeshki M, Fan L, Brakel P, Courville A, Bengio Y (2016) Deconstructing the ladder network architecture. In: International conference on machine learning. pp 2368–2376
Pinage FA, dos Santos EM (2015) A dissimilarity-based drift detection method. In: 2015 IEEE 27th international conference on tools with artificial intelligence (ICTAI). IEEE, pp 1069–1076
Pinage FA, dos Santos EM, da Gama JMP (2016) Classification systems in dynamic environments: an overview. Wiley Interdiscip Rev Data Min Knowl Discov 6(5):156–166
Ruta D, Gabrys B (2007) Neural network ensembles for time series prediction. In: 2007 International joint conference on neural networks. IEEE, pp 1204–1209
Spinosa E, de Leon de Carvalho AP, Gama J (2008) Cluster-based novel concept detection in data streams applied to intrusion detection in computer networks. In: Proceedings of the 2008 ACM symposium on applied computing. ACM, pp 976–980
Street N, Kim Y (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. ACM, pp 377–382
Tsymbal A, Pechenizkiy M, Cunningham P, Puuronen S (2008) Dynamic integration of classifiers for handling concept drift. Inf Fusion 9(1):56–68
Wang H, Fan W, Yu P, Han J (2003) Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 226–235
Woods K, Kegelmeyer WP, Bowyer K (1997) Combination of multiple classifiers using local accuracy estimates. IEEE Trans Pattern Anal Mach Intell 19(4):405–410
Wu X, Li P, Hu X (2012) Learning from concept drifting data streams with unlabeled data. Neurocomputing 92:145–155
Zliobaite I (2011) Combining similarity in time and space for training set formation under concept drift. Intell Data Anal 15(4):589–611

Affiliations

Felipe Pinagé[1,2] · Eulanda M. dos Santos[1] · João Gama[3]

✉ Felipe Pinagé
felipepinage@icomp.ufam.edu.br; fapinage@inf.ufpr.br

Eulanda M. dos Santos
emsantos@icomp.ufam.edu.br

João Gama
jgama@fep.up.pt

[1] Institute of Computing, Federal University of Amazonas, Manaus, AM, Brazil

[2] Department of Informatics, Federal University of Paraná, Curitiba, PR, Brazil

[3] Institute of Engineering and Computer Systems, University of Porto, Porto, Portugal