Classifying evolving data streams with partially labeled data

Hanen Borchani*, Pedro Larrañaga and Concha Bielza

Recently, several approaches have been proposed to deal with the increasingly challenging task of mining concept-drifting data streams. However, most are based on supervised classification algorithms assuming that true labels are immediately and entirely available in the data streams. Unfortunately, such an assumption is often violated in real-world applications given that it is expensive or because it takes a long time to obtain all true labels. To deal with this problem, we propose in this paper a new semi-supervised approach for handling concept-drifting data streams containing both labeled and unlabeled instances. First, contrary to existing approaches, we monitor three possible kinds of drift: feature, conditional or dual drift. Drift detection is based on a hypothesis test comparing Kullback-Leibler divergence between old and recent data, whose distribution under the null hypothesis of coming from the same distribution is approximated via a bootstrap method. Then, if any drift occurs, a new classifier is learned from the recent data using the EM algorithm; otherwise, the current classifier is left unchanged. Our approach is so general that it can be applied to different classification models. Experimental studies, using the naive Bayes classifier and logistic regression, on both synthetic and real-world data sets demonstrate that our approach performs well.

Keywords: Data streams, concept drift, change detection, semi-supervised learning

1. Introduction

With the rapid growth of information technology, infinite flows of records are collected daily. These flows, defined as data streams, pose many challenges to computing systems due to limited time and memory resources. Furthermore, they are characterized by their concept-drifting aspect [10,27]. Concept drift means that the learned concepts and/or the underlying data distribution are not stable and may change over time. As a result, the model in use becomes out-of-date and has to be updated.

The field of mining concept-drifting data streams has received increasing attention and has been intensively researched in recent years. Several approaches have been proposed [3,5,13,19,25,28] and applied to a wide range of real-world applications including network monitoring, telecommunications data management, market-basket analysis, information filtering, fraud and intrusion detection, etc.

However, most of these approaches are based on supervised classification algorithms assuming the availability of labeled data for accurate learning. Generally, they continuously monitor classification performance and detect a concept drift if there is a significant fall over time. Unfortunately, the assumption of entirely labeled data streams availability is often violated in real-world problems, as labels may be scarce and not readily available.

For instance, for the malware detection problem, only a few true labels (i.e. malware or goodware) may be available immediately after the classification process, and therefore we may have to wait for a quite long time until all the instances are labeled. This leads a traditional stream classification algorithm to choose between updating the classifier with just a few labeled data, which usually results in a poor classifier, or waiting longer to get all labeled data. This can also affect the quality of the classifier since most of the data will be outdated.

Semi-supervised learning methods have proved to be useful in such cases since they combine both labeled and unlabeled data to enhance the performance of classification algorithms [34]. However, they mainly assume that data is generated according to some stationary distribution, which is not true when learning from evolving data streams, where changes may occur over time.

In this paper, we propose a new semi-supervised learning approach for concept-drifting data streams. Our aim is to take advantage of unlabeled data to detect possible concept drifts and, if necessary, update the classifier over time even if only a few labeled data are available.

To this end, inspired by earlier work by Dasu et al. [6], we use the Kullback-Leibler (KL) divergence [20] to measure distribution differences between data stream batches. Then, based on a bootstrapping method [8], we determine whether or not the KL measures are statistically significant, i.e. whether or not a drift occurs. However, our approach differs from Dasu's work on three key points. First, we do not only detect whether or not a drift occurs, but we further distinguish and monitor three possible kinds of drift: feature, conditional or dual drift. Second, we do not assume the available data streams are entirely labeled. Indeed, we detect possible drifts using both labeled and unlabeled instances. Moreover, we propose a general approach for learning from all these instances. In fact, when any of the three possible kinds of drift is detected, a new classifier is learned using the expectation-maximization (EM) algorithm [7]. EM has been widely used in semi-supervised learning where it has been found to improve classification accuracy, especially when there is a small number of labeled data [24]. Otherwise, i.e. when no drift is detected, the current classifier is left unchanged.

Note that our approach is so general that it can be applied with different classification learning algorithms. In this paper, we consider two classifiers, namely naive Bayes and logistic regression. We perform experiments on rotating hyperplane and mushroom data sets using different percentages of labeled instances. Moreover, we evaluate our approach using a real-world malware detection data set, where we deal with the additional problem of imbalanced data streams and make use of two recently proposed approaches for mining skewed data streams, namely clustering-sampling [31] and SERA [5]. The results show that our approach performs well even using limited amounts of labeled data.

The remainder of this paper is organized as follows. Section 2 defines the concept drift problem and three types of drift. It then goes on to briefly review existing approaches for learning from concept-drifting data streams. Section 3 introduces our new approach. Section 4 presents the experimental study. Finally, Section 5 rounds the paper off with some conclusions.

2. Concept drift

2.1. Problem definition

In dynamic environments, the characteristic properties of data streams are often not stable but change over time. This is known as the concept drift problem [32]. According to Tsymbal [27], there are two possible types of concept drift: *real concept drift*, defined as a change of the target concept that the classifier is trying to predict, and *virtual concept drift*, defined as a change of the underlying data distribution.

From a probabilistic point of view, concept drift can be defined as the change in the joint probability distribution $P(\mathrm{x}, c)$, where c is the class label of a feature vector x. $P(\mathrm{x}, c)$ is the product of the class posterior distribution $P(c \mid \mathrm{x})$ and the feature distribution $P(\mathrm{x})$. According to [14], there are three possible sources of concept drifts:

- *Conditional change*: In this case, a change occurs in $P(c \mid \mathrm{x})$, that is, the class labels change given the feature vectors. For instance, a conditional change may occur in an information filtering domain consisting of classifying a stream of documents, denoted by x, as relevant or irrelevant, denoted by c, if the relevance of some documents changes over time, that is, their class labels change from relevant to irrelevant or vice versa. With respect to Tsymbal's concept drift categorization, a conditional change corresponds to a real concept drift.
- *Feature change*: In this case, a change occurs in $P(\mathrm{x})$. Intuitively, some previously infrequent feature vectors may become more frequent or vice versa. For instance, the relative frequency of some documents in information filtering domain changes over time regardless of their relevance, which may remain constant over a long period of time. With respect to Tsymbal's concept drift categorization, a feature change represents a virtual concept drift.
- *Dual change*: In this case, changes occur in both $P(\mathrm{x})$ and $P(c \mid \mathrm{x})$. According to the information filtering example, changes in both the relative frequency and the relevance of some documents are observed, i.e. a virtual and a real concept drift both occur together.

Moreover, Zhang et al. [33] proposed an additional categorization also based on the decomposition of $P(\mathrm{x}, c)$ into two parts, as $P(\mathrm{x}, c) = P(\mathrm{x}) \cdot P(c \mid \mathrm{x})$. In fact, they defined *rigorous concept drifting* for changes in both $P(\mathrm{x})$ and $P(c \mid \mathrm{x})$, and *loose concept drifting* for changes in $P(\mathrm{x})$ only.

To the best of our knowledge, in spite of these categorizations, all existing approaches dealing with the concept drift problem either update the current classifier without using any detection method, or detect only whether or not there is drift, i.e. without specifying which type of concept drift occurs.

In this paper, we propose an efficient approach for quantifying and detecting the three possible types of drift: feature, conditional or dual using both labeled and unlabeled data. Details are presented in Section 3.2.

2.2. *Related work*

Different approaches have been proposed to handle concept-drifting data streams. As pointed out in [12], these approaches can be classified into *blind approaches* that adapt the classifier at regular intervals without considering whether changes have really occurred, and *informed approaches* that are used in conjunction with a detection method and only adapt the classifier after a change is detected.

Examples of blind approaches include weighted examples [16] and fixed size time windows [32]. Weighted examples assigns lower weights to old instances according to their age and/or utility in order to focus more on recent instances incorporating the new concepts. Fixed size time windows consider over time a fixed number of instances over time: In this case, the choice of an appropriate window size should trade off fast adaptation in phases with concept drifts against good generalization in stable phases without concept drifts.

Ensemble methods can also be considered as blind approaches. In fact, the general technique applied by these methods is that the data stream is divided into sequential blocks of fixed size, and each of these blocks is used to train a classifier. The ensemble is continuously refined by adding a new classifier, removing the oldest or the weakest classifier, increasing or decreasing the classifier weights using some criteria usually based on current data block performance [4,19,28,30,31].

The adaptive size time window is an example of informed methods [32]. In fact, the window size is adjusted dynamically to the current concept drift: As a general rule, if a drift is detected the window size decreases to exclude the out-of-date instances; otherwise the window size increases to include the more recent instances [21].

Clearly, informed methods are more interesting since they are a more efficient way of coping with concept drifts and avoid the uncontrolled updating of the current classifier. The main issue is how to detect concept drifts. Most of the existing research monitors at least one performance indicator over time [3,12,18,25,32]. Classification accuracy is the most used indicator, i.e. if there is a consistent drop in the accuracy, a drift is signaled. Other performance indicators, such as error rate, recall and precision, have also been used.

An alternative approach detecting drift is to monitor the data distribution in two different windows [13, 15,29]. It is assumed that as long as the distribution of old instances is similar to the distribution of recent ones, no concept drift occurred. A distribution difference, on the other hand, indicates a concept drift. In particular, Dasu et al. [6] and Sebastião and Gama [26] used the Kullback-Leibler divergence to measure the distance between the probability distributions of two different windows to detect possible changes, and proved its generality, efficiency and resilience to false alarms.

However, note that all previously presented works assume that true labels are entirely available in data streams. To the best of our knowledge, only two relevant previous works have addressed the problem of scarceness of labeled instances in concept drifting data streams.

The first, proposed by Klinkenberg [17], is based on transductive support vector machines and it maintains two separate adaptive windows on labeled and unlabeled data in order to monitor, respectively, the probabilities $P(c \mid \mathrm{x})$ captured by labeled data and $P(\mathrm{x})$ underlying both labeled and unlabeled data. This was justified by the fact that $P(c \mid \mathrm{x})$ and $P(\mathrm{x})$ may change at different rates. However, although theoretically well-founded, this method has never been evaluated.

The second work was recently proposed by Masud et al. [22]. It is based on an ensemble approach where each model in the ensemble is built as micro-clusters using a semi-supervised clustering technique. In fact, the learning step of each model starts by choosing k_c points from the labeled data of class C to initialize k_c centroids. Then, the EM algorithm is applied by iterating the following two steps until convergence: The E-step assigns each unlabeled data point $\mathbf{x}$ to a cluster such that its contribution to a cluster-impurity function is minimized, and the M-step recomputes each cluster centroid by averaging all the points in that cluster. Finally, a summary of the statistics of the instances belonging to each built cluster is saved as a micro-cluster. These micro-clusters serve as a classification model.

To cope with stream evolution, Masud et al. [22] keep an ensemble of L models. Whenever a new model is built from a new data chunk, they update the ensemble by choosing the best L models from $L+1$ models (previous L models and the new model), based on their individual accuracies on the labeled instances of the new data chunk. Besides, they refine the existing models in the ensemble whenever a new class of data evolves in the stream. Note finally that this approach is blind since it does not incorporate any drift detection method.

3. Background on EM algorithm

Let D denote the data stream that arrives over time in batches. Let D^s denotes the batch at step s. D^s consists of the union of two disjoint subsets D_u^s and D_l^s. D_u^s denotes a set of N_u^s unlabeled instances

$(\mathbf{x})$, whereas D_l^s denotes a set of N_l^s labeled instances $(\mathbf{x}, c)$, s.t. $\mathbf{x}$ represents an n-dimensional feature vector $(x_1, \ldots, x_n)$ and $c \in \Omega_C = \{c_1, c_2, \ldots, c_{|C|}\}$ represents the corresponding class value for labeled instances. $N^s = N_u^s + N_l^s$ denotes the total size of D^s.

Learning a classifier from the D^s data corresponds to maximizing the likelihood of D^s given the parameters $\boldsymbol{\theta}^s$. Assuming that instances are independent, this likelihood is the product of all (labeled and unlabeled) instance probabilities expressed as follows [24]:

$$\begin{aligned} P(D^s \mid \boldsymbol{\theta}^s) = & \prod_{i=1}^{N_l^s} P(c_j \mid \mathbf{x}_i; \boldsymbol{\theta}^s) P(\mathbf{x}_i \mid \boldsymbol{\theta}^s) \\ & \cdot \prod_{i=1}^{N_u^s} \sum_{j=1}^{|C|} P(c_j \mid \mathbf{x}_i; \boldsymbol{\theta}^s) P(\mathbf{x}_i \mid \boldsymbol{\theta}^s) \cdot \end{aligned} \tag{1}$$

where the first term is derived from labeled instances, and the second one is based on unlabeled data where the sum expresses the fact that the unknown class value can be any of the existing values.

Then, considering $logP(D^s \mid \boldsymbol{\theta}^s) = LL(D^s \mid \boldsymbol{\theta}^s)$, we have:

$$\begin{aligned} LL(D^s \mid \boldsymbol{\theta}^s) = & \sum_{i=1}^{N_l^s} log(P(c_j \mid \mathbf{x}_i; \boldsymbol{\theta}^s) P(\mathbf{x}_i \mid \boldsymbol{\theta}^s)) \\ & + \sum_{i=1}^{N_u^s} log \sum_{j=1}^{|C|} P(c_j \mid \mathbf{x}_i; \boldsymbol{\theta}^s) P(\mathbf{x}_i \mid \boldsymbol{\theta}^s) \cdot \end{aligned} \tag{2}$$

Notice that this equation contains a log of sums for the unlabeled data, which makes a maximization by partial derivatives with respect to $\boldsymbol{\theta}^s$ analytically intractable.

Consider that we can have access to the class labels of all the instances, represented using a matrix of binary indicator variables $\mathbf{z}$, where rows correspond to different instances and columns to different classes, so that an entry is $z_{ij} = 1$ iff c_j is the class of instance $\mathbf{x}_i$, and $z_{ij} = 0$ otherwise. Thus, Eq. (2) can be rewritten as follows without a log of sums, because only one term inside the sum would be non-zero:

$$LL(D^s \mid \boldsymbol{\theta}^s; \mathbf{z}) = \sum_{i=1}^{N^s} \sum_{j=1}^{|C|} z_{ij} \, log(P(c_j \mid \mathbf{x}_i; \boldsymbol{\theta}^s) P(\mathbf{x}_i \mid \boldsymbol{\theta}^s)) \cdot \tag{3}$$

We use the EM algorithm [7] to find the maximum $\hat{\boldsymbol{\theta}}^s$ of Eq. (3). Let $\hat{\mathbf{z}}_t$ and $\hat{\boldsymbol{\theta}}_t^s$ denote the estimates for $\mathbf{z}$ and $\boldsymbol{\theta}^s$ at iteration t. EM starts with an initial estimate of classifier parameters $\hat{\boldsymbol{\theta}}_1^s$ from only the labeled data in D_l^s. Then, it iterates over the E- and M-steps:

- The E-step uses the current classifier parameters to probabilistically assign labels to the unlabeled instances in D_u^s. Formally, it computes the expected value of

  $$\hat{\mathbf{z}}_{t+1} = E[\mathbf{z} \mid D^s; \hat{\boldsymbol{\theta}}_t^s] \cdot \tag{4}$$

  Clearly, for labeled data, z_{ij} is easily determined since classes are already known. For unlabeled data, z_{ij} should be estimated as follows:

  $$E[z_{ij} \mid D^s; \hat{\boldsymbol{\theta}}_t^s] = \begin{cases} 1 & \mathrm{if\ c_j = arg\ max_c P(c \mid \mathbf{x}_i; \hat{\boldsymbol{\theta}}_t^s)} \cdot \\ 0 & \mathrm{otherwise.} \end{cases} \tag{5}$$

- The M-step re-estimates the classifier for all the data in D^s, i.e. using all instances (the originally labeled and the newly labeled by the E-step). In fact, this step consists of computing new parameters $\hat{\boldsymbol{\theta}}^s_{t+1}$ using the current expected value of $\hat{\mathbf{z}}$. Formally, we have

$$\hat{\boldsymbol{\theta}}^s_{t+1} = arg\ max_{\boldsymbol{\theta}^s} LL(D^s \mid \boldsymbol{\theta}^s; \hat{\mathbf{z}}_{t+1})\cdot \tag{6}$$

These two steps are iterated until convergence as proved by Dempster et al. [7]. At convergence, EM finds $\hat{\boldsymbol{\theta}}^s$ that locally maximizes the log likelihood with respect to both labeled and unlabeled data.

4. New approach for mining concept-drifting data streams with a limited number of labeled instances

In this section, we will first introduce the two considered classifiers, namely naive Bayes and logistic regression, learnt from both labeled and unlabeled instances. Then, we will present the drift detection method.

4.1. Used classifiers

4.1.1. Naive bayes (NB)

Naive Bayes [23] is a generative classifier that optimizes the joint log likelihood of the data as previously detailed. Based on the assumption that the features are all conditionally independent of one another given the class variable C, parameters $\boldsymbol{\theta}^s$ denote the probability table of C, i.e. $P(C)$, as well as the conditional probability tables of each feature X_r given C, i.e. $P(X_r \mid C), r \in \{1, \ldots, n\}$.

To classify a given instance, the posterior probability of each possible class value c_j is computed, and then, the most probable class is selected. More formally,

$$c = arg\ max_{c_j} P(c_j) \prod_{r=1}^{n} P(x_r \mid c_j)\cdot \tag{7}$$

4.1.2. Logistic regression (LR)

Logistic regression [11] is a discriminative classifier that maximizes the conditional log likelihood instead of the log likelihood. Hence, in this case, instead of (3), EM algorithm maximizes the following formula:

$$LL(D^s \mid \boldsymbol{\theta}^s; \mathbf{z}) = \sum_{i=1}^{N^s} \sum_{j=1}^{|C|} z_{ij}\ logP(c_j \mid \mathbf{x}_i; \boldsymbol{\theta}^s)\cdot \tag{8}$$

where parameters $\boldsymbol{\theta}^s$ are represented by the vector $(\boldsymbol{\theta}^s_{j0}, \boldsymbol{\theta}^s_{j1}, \ldots, \boldsymbol{\theta}^s_{jn})^T$ for $j = 1, \ldots, |C|$.

To classify a given instance, the posterior probability of each possible class value c_j is computed as follows:

$$P(C = c_j \mid \mathbf{x}; \boldsymbol{\theta}^s) = \begin{cases} \frac{\exp(\boldsymbol{\theta}^s_{j0} + \sum_{r=1}^{n} \boldsymbol{\theta}^s_{jr} x_r)}{1 + \sum_{p=1}^{|C|-1} \exp(\boldsymbol{\theta}^s_{p0} + \sum_{r=1}^{n} \boldsymbol{\theta}^s_{pr} x_r)} & \forall \text{j} < |\text{C}|, \\ \\ \frac{1}{1 + \sum_{p=1}^{|C|-1} \exp(\boldsymbol{\theta}^s_{p0} + \sum_{r=1}^{n} \boldsymbol{\theta}^s_{pr} x_r)} & \text{for j} = |\text{C}|\cdot \end{cases} \tag{9}$$

Then, the c_j value with the maximum probability is assigned as a label.

4.2. *Detecting a concept drift*

Given a new batch of data D^{s+1}, the objective is to detect changes whenever they occur and adapt the current classifier if necessary. In general, it is assumed that as long as the joint probability distribution of D^{s+1} is similar to the distribution of D^s, no concept drift occurs. Otherwise, a concept drift should be indicated.

In order to detect possible changes, we use the KL divergence [20], also known as the relative entropy, to measure differences between the empirical distributions of D^{s+1} and D^s. Note that the KL divergence has two fundamental properties, namely, non-negativity, being 0 iff the two compared distributions are the same, and asymmetry. Moreover, a higher KL value indicates a higher dissimilarity between distributions, and so, a pronounced drift.

First, in order to monitor the conditional change, we proceed to measure the KL divergence kl_{cc} between the class posterior distributions of D^{s+1} and D^s using only their corresponding labeled instances. kl_{cc} is computed as a sum of KL divergences, each of which measuring the divergence between the conditional distributions of the class given feature instantiation, expressed as follows:

$$\begin{aligned} kl_{cc} &= \sum_{\mathbf{x}} KL(\hat{P}_{D_l^{s+1}}(C \mid \mathrm{x}) || \hat{P}_{D_l^s}(C \mid \mathrm{x})) \\ &= \sum_{\mathbf{x}} \sum_{j=1}^{|C|} \hat{P}_{D_l^{s+1}}(c_j \mid \mathrm{x}) log_2 \frac{\hat{P}_{D_l^{s+1}}(c_j \mid \mathrm{x})}{\hat{P}_{D_l^s}(c_j \mid \mathrm{x})}. \end{aligned} \tag{10}$$

In addition, to monitor the feature change, we measure the KL divergence kl_{fc} between the feature distributions of D^{s+1} and D^s using all the labeled and unlabeled instances except the class variable:

$$kl_{fc} = KL(\hat{P}_{D^{s+1}}(\mathrm{x}) || \hat{P}_{D^s}(\mathrm{x})) = \sum_{\mathbf{x}} \hat{P}_{D^{s+1}}(\mathrm{x}) log_2 \frac{\hat{P}_{D^{s+1}}(\mathrm{x})}{\hat{P}_{D^s}(\mathrm{x})}. \tag{11}$$

In order to determine whether or not the computed KL measures are statistically significant, we use the bootstrapping method [8] following previous work reported in [6]. Intuitively, this method allows us to determine, by repeated sampling with replacement from the data, whether or not a specific measurement on the data is significant.

Specifically, to decide whether or not the resulting kl_{cc} value is significant, we consider the null hypothesis

$$H_{0_{cc}} : P_{D_l^{s+1}}(C \mid \mathbf{X}) = P_{D_l^s}(C \mid \mathbf{X}),$$

denoting that no conditional change has occurred. So, our objective is to determine the probability of observing the value kl_{cc} if $H_{0_{cc}}$ is true.

To this end, given the empirical distribution $\hat{P}_{D_l^s}(C \mid \mathrm{x})$, we sample k data sets denoted S_b, $b = 1, \ldots, k$, each of size $2N_l^s$. Then, we consider the first N_l^s instances S_{b1} as coming from the distribution $\hat{P}_{D_l^s}(C \mid \mathrm{x})$, and the remaining N_l^s instances $S_{b2} = S_b \setminus S_{b1}$ as coming from the other distribution $\hat{P}_{D_l^{s+1}}(C \mid \mathrm{x})$; and we compute the bootstrap estimates $\hat{kl}_{ccb} = \sum_{\mathbf{x}} KL(\hat{P}_{S_{b2}}(C \mid \mathrm{x}) || \hat{P}_{S_{b1}}(C \mid \mathrm{x}))$ between each two samples S_{b2} and S_{b1}, $b = 1, \ldots, k$. The obtained estimates form an empirical distribution from which we construct a critical region $[\hat{kl}_{cc}^{\alpha}, \infty)$, where $\hat{kl}_{cc}^{\alpha}$ represents the $(1-\alpha)$-percentile of the bootstrap estimates, and α is a desired significance level.

Finally, if we observe that kl_{cc} falls into the critical region, i.e. $kl_{cc} > \hat{kl}_{cc}^{\alpha}$, we conclude that it is statistically significant and invalidates $H_{0_{cc}}$. In other words, we conclude that a conditional change is detected.

Similarly, in order to decide whether or not the resulting kl_{fc} value is significant, we consider the null hypothesis

$$H_{0_{fc}} : P_{D^{s+1}}(\mathbf{x}) = P_{D^s}(\mathbf{x}),$$

and apply the same process to determine the critical region $[\hat{kl}_{fc}^{\alpha}; \infty)$ and decide about a feature change. Note that, if either a feature or conditional change is detected, we proceed to learn a new classifier. Otherwise, the current classifier is left unchanged.

To recapitulate, Algorithm 1 outlines the whole proposed approach. First, KL divergence and the bootstrapping method are used to monitor possible conditional and feature changes (steps 1 to 4). If any change is detected, a new classifier is learned using the expectation maximization algorithm (step 5.1): an initial estimate of classifier parameters $\hat{\boldsymbol{\theta}}_1^{s+1}$ is induced using only the labeled instances of the new data set D^{s+1} (step 5.2), then EM iterates over the E- and M-steps until convergence (step 5.3). In case that no change occurred, the classifier is left unchanged (step 6).

Algorithm 1

```
begin
    Input : $D^s, \boldsymbol{\theta}^s, D^{s+1}, k, \alpha$
    Output : $\boldsymbol{\theta}^{s+1}$
    1. Compute $kl_{cc}$
    2. Compute the bootstrap estimates $\hat{kl}_{ccb}$, $b = 1, ..., k$, and critical region $[\hat{kl}_{cc}^{\alpha}, \infty)$
    3. Compute $kl_{fc}$
    4. Compute the bootstrap estimates $\hat{kl}_{fcb}$, $b = 1, ..., k$, and critical region $[\hat{kl}_{fc}^{\alpha}, \infty)$
    5. if $kl_{cc} > \hat{kl}_{cc}^{\alpha}$ or $kl_{fc} > \hat{kl}_{fc}^{\alpha}$ then
        5.1 A change is detected, learn a new classifier from $D^{s+1}$
        5.2 $\hat{\boldsymbol{\theta}}_1^{s+1} \leftarrow$ initial parameters induced only from labeled data $D_l^{s+1}$
        5.3 while no convergence do
            E-step: compute the expected labels for all unlabeled instances using (4)
            M-step: update classifier parameters using (6) obtaining $\hat{\boldsymbol{\theta}}^{s+1}$
        5.4 $\boldsymbol{\theta}^{s+1} \longleftarrow \hat{\boldsymbol{\theta}}^{s+1}$
    6. else
        No change is detected: $\boldsymbol{\theta}^{s+1} \longleftarrow \boldsymbol{\theta}^s$
    7. Return $\boldsymbol{\theta}^{s+1}$
end
```

5. Experimental study

5.1. Used data sets

We test our approach on the following synthetic and real data sets.

5.1.1. Rotating hyperplane data set

The rotating hyperplane data set is considered as a benchmark synthetic data set and has been widely used to simulate the concept drift problem [10,14,28,30]. In fact, this synthetic data set allows us to carry out experiments with different types of drift, as well as different percentages of labeled data and, hence, to investigate the performance of our approach under controlled conditions.

A hyperplane in an n-dimensional space is denoted by $\sum_{i=1}^{n} w_i x_i = w_0$, where $\mathbf{w} = (w_1, \ldots, w_n)^T$ is the weight vector. Instances for which $\sum_{i=1}^{n} w_i x_i \geqslant w_0$ are labeled positive, and instances for which

$\sum_{i=1}^{n} w_i x_i < w_0$ are labeled negative. Weights w_i are initialized by random values in the range of $[0, 1]$, and w_0 values are determined so that $w_0 = \frac{1}{2}\sum_{i=1}^{n} w_i$.

We generated x_i from a Gaussian distribution with mean μ_i and variance σ_i^2. The feature change is simulated by changing the mean, i.e. μ_i is changed to $\mu_i s_i(1 + t)$, and the conditional change is simulated by the change of weights w_i to $w_i s_i(1 + t)$. Parameter $t \in [0, 1]$ represents the magnitude of the changes, and parameter $s_i \in \{-1, 1\}$ specifies the direction of the changes which could be reversed with a probability of 0.1. We generated a data stream of 10 dimensions ($n = 10$) with 80,000 instances, using different magnitudes of change t respectively set to 0.1, 0.2, 0.5, 1 for each 20,000 instances. Then, we split the whole data stream into sets of blocks of size 2000, and from each block we considered equal training and testing subsets of size 1000, such that every training set is followed by a testing set.

5.1.2. Mushroom data set

The mushroom data set, from the UCI repository [2], is regarded as having virtual concept drift (i.e. feature changes) but no real concept drift (i.e. conditional changes) [19]. The mushroom data set contains 22 variables and 8124 instances. We split it into 6 blocks, and used 1000 instances from each block for training and 354 instances for testing.

5.1.3. Malware detection data set

The malware detection data set represents the important problem of continuously classifying received files into malware (e.g. viruses, spyware, trojans, phishing, spam, etc.) or goodware to ensure that users are protected against malicious code. This data set has been provided by an IT security company and consists of 40,000 records. It contains 5398 features and a binary class taking either the malware or goodware value. Due to the confidentiality of the data, we omit the name of the company here, as well as the detailed description of the features.

Contrary to experiments with the previous data, we do not know whether or not changes occur in this real data set; and if so, we do not know when and which kind of changes occur. Moreover, we do not fix the percentage of labeled data in each block. Instead, we use all the available labeled data, the number may vary from one data block to another.

We also deal with two additional issues to process this malware detection data set. The first is *feature selection*, which aims to select a small subset of relevant features in order to avoid features dependency and redundancy and enhance classifier performance. In this paper, we use the conditional mutual information maximization criterion (CMIM) [9]. It iteratively picks features that maximize their mutual information with the class variable, conditionally upon the response of the already picked features. In this way, CMIM ensures weak dependency and no redundancy as it does not select a feature similar to any that have already been picked even if it is individually powerful.

In our case, feature selection is applied each time we learn a new classifier, i.e. each time we detect changes. Hence, a new and more informative subset of features is selected given new incoming data. In fact, some old selected features may be removed and new different features may be selected. This, consequently, allows us to build more efficient classifiers.

The second issue is *imbalanced data* since the number of malware instances is much higher than goodware instances. This leads to an important problem since the learned classifier may be biased towards the malware class, and therefore its predictive accuracy may be very poor on the goodware class. We apply two recent approaches to balance the class distribution:

- The clustering-sampling approach proposed by Wang et al. [31] makes use of the k-means clustering algorithm to select negative instances for representing the negative class (i.e. malware class in our

Table 1
Data set descriptions

Data set	Number of features	Number of instances	Number of blocks	Number of instances in a block
Rotating hyperplane	10	80000	40	2000
Mushroom	22	8124	6	1354
Malware detection	50	40000	10	4000

case). Firstly, the number of clusters nc is set to the size of positive instances (i.e. goodware instances). Then, the negative instances are clustered into nc clusters and the centroid of each cluster is used as as negative instance for representing the negative class.

- The selectively recursive approach (SERA) proposed by Chen and He [5] makes use of the previous data blocks knowledge to balance the current data block. In fact, it consistently collects the positive instances from the previous data blocks. Then, it applies the Mahalanobis distance to measure the similarity between each instance and the current positive instances, and includes a subset of the most similar previous positive instances of a size proportional to the size of the current negative set only. This is justified by the fact that only the previous positive instances not including the drifting concepts are actually helpful for the learning process.

The malware detection data set is divided into sets of blocks of size 4000, and from each block, the first 2000 instances are used for training while the remaining instances are used for testing. For feature selection, we select 50 of the 5398 features.

To summarize, the details of the three considered data sets are given in Table 1. Note finally that, for bootstrap parameters, we use the significance level $\alpha = 0.05$ and samples number $k = 500$ in all experiments. Our choice is based on Dasu et al.'s work [6] where they prove that the number of samples does not significantly affect the quality of the results and suggest that $k = 500$ is a reasonable number of samples. They also point out that lower α values make the null hypothesis harder to reject, leading to a lower change detectability. According to our experiments, $\alpha = 0.05$ works well and can be considered as an appropriate value.

5.2. *Experimental results*

5.2.1. *Results with rotating hyperplane data set*

Table 2 represents the results for the drift detection proposal. The first column represents the block numbers of the training sets. For instance, 1–2 denotes that the current data is the training set of the first block, while the new data corresponds to the training set of the second block. Then, in columns 2 and 3, we show the kl_{fc} and $\widehat{kl}^{\alpha}_{fc}$ values. These values are the same for all experiments irrespective of the different percentages of labeled data, since they only use the feature values. Finally, columns 4 to 9 report kl_{cc} and $\widehat{kl}^{\alpha}_{cc}$ respectively, for 2%, 5% and 10% of labeled data.

As expected, a feature change is only detected between blocks 10 and 11 where the magnitude of change t goes from 0.1 to 0.2, blocks 20 and 21 where t goes from 0.2 to 0.5, and blocks 30 and 31 where the t goes from 0.5 to 1. The larger the modification of t values, the higher the kl_{fc} values are, showing a more significant drift in the feature distributions between the data blocks.

The same applies to the conditional distributions monitored by kl_{cc} values for both 5% and 10% of labeled data, where higher kl_{cc} values are obtained for higher t values. However, in the case of 2% of labeled data, no conditional changes are detected. This can be explained by the fact that the true conditional distribution cannot be accurately approximated with very few labeled instances. In their

Table 2
Drift detection results for rotating hyperplane data set

	Feature change		Conditional change					
			2% labeled		5% labeled		10% labeled	
blocks	kl_{fc}	$\hat{kl}^{\alpha}_{fc}$	kl_{cc}	$\hat{kl}^{\alpha}_{cc}$	kl_{cc}	$\hat{kl}^{\alpha}_{cc}$	kl_{cc}	$\hat{kl}^{\alpha}_{cc}$
1–2	0.0962	0.1386	0.4807	3.1025	0.1168	0.5257	0.0222	0.4989
2–3	0.1353	0.1801	0.8112	3.9845	0.0862	1.7670	0.0514	0.6499
3–4	0.1214	0.1364	0.0637	5.7233	0.3874	0.1958	0.1013	0.7847
4–5	0.1245	0.1381	0.0158	6.1474	0.4248	1.4025	0.2139	0.5257
5–6	0.1069	0.1404	0.3166	2.6613	0.2985	1.4885	0.3100	0.7380
6–7	0.1008	0.1378	1.4039	6.9068	0.7706	1.3782	0.5962	0.8997
7–8	0.1013	0.1381	1.2359	5.6246	0.6927	1.3755	0.1665	0.3234
8–9	0.1304	0.1388	0.9124	6.2563	0.5369	1.6424	0.1990	0.2339
9–10	0.1017	0.1398	1.2339	4.2875	0.2925	1.3847	0.2180	0.6060
10–11	**0.1434**	**0.1405**	3.2875	6.4493	**1.3862**	**1.2369**	**0.9812**	**0.6499**
11–12	0.1249	0.1402	1.2026	6.2875	0.2534	1.7670	0.1355	0.5428
12–13	0.1154	0.1390	1.9967	7.3604	0.4209	1.5021	0.1544	0.3575
13–14	0.0882	0.1398	1.8104	8.2256	1.0638	1.7860	0.1419	0.7648
14–15	0.0956	0.1378	2.4464	6.4493	0.5212	1.3369	0.2552	0.6499
15–16	0.1344	0.1369	2.3008	6.3567	0.8237	1.6660	0.1580	0.6694
16–17	0.1373	0.1866	0.7418	6.2875	1.6932	1.8060	0.1862	0.3013
17–18	0.1374	0.1444	1.0548	5.1297	0.7915	1.9310	0.1273	0.6499
18–19	0.1316	0.1400	2.3245	6.5493	1.3169	1.8142	0.1355	0.5005
19–20	0.0932	0.1392	1.4075	6.3649	0.5472	1.3725	0.3919	0.7842
20–21	**0.1544**	**0.1408**	4.1283	6.4512	**1.3821**	**1.2364**	**1.0118**	**0.8162**
21–22	0.1378	0.1538	0.0637	6.4502	0.4669	1.7670	0.0456	0.5257
22–23	0.1141	0.1387	0.9158	5.8614	0.3925	1.6927	0.1153	0.2978
23–24	0.1296	0.1366	1.4257	6.5915	0.4248	1.2849	0.1030	0.1375
24–25	0.0851	0.1370	1.1079	6.1807	0.3472	0.5288	0.0168	0.2311
25–26	0.0849	0.1382	1.0042	6.4346	0.2812	1.2745	0.0953	0.2110
26–27	0.0653	0.1342	1.4721	6.3684	0.5004	1.3369	0.3651	0.5147
27–28	0.0880	0.1373	1.2339	6.6503	0.5257	1.1575	0.2120	0.6499
28–29	0.1139	0.1434	1.9099	6.5288	0.6927	1.7670	0.3275	0.5005
29–30	0.1383	0.1521	1.7233	6.8027	0.0818	1.7495	0.1094	0.4257
30–31	**0.1767**	**0.1466**	4.7233	6.4346	**1.9310**	**1.3660**	**1.3369**	**0.7648**
31–32	0.1011	0.1373	1.4792	6.2875	0.1613	0.2534	0.0375	0.3409
32–33	0.1332	0.1394	0.6748	6.7841	0.4838	1.7897	0.1978	0.4885
33–34	0.1360	0.1565	0.9099	5.1964	0.6927	1.0546	0.3248	0.7648
34–35	0.1171	0.1395	0.8475	3.5168	0.4354	0.4999	0.1279	0.5694
35–36	0.0951	0.1376	1.2339	6.4593	0.9211	1.3369	0.1947	0.6348
36–37	0.0957	0.1395	0.9213	4.8143	0.3234	1.7495	0.1456	0.5257
37–38	0.1056	0.1367	0.6014	6.3684	0.2648	0.6364	0.1898	0.2339
38–39	0.1264	0.1625	0.9078	6.3125	0.1763	0.4376	0.1504	0.4342
39–40	0.1378	0.1367	0.3478	5.6177	0.4517	1.6849	0.3973	0.5389

experiments studying the effect of window size on the performance of the change detection scheme, Dasu et al. [6] come to the same conclusion, i.e a larger window size gives better approximation of the true underlying distribution and results in a better detection of changes.

Furthermore, Fig. 1 presents accuracy curves for NB and LR. For each curve, the x-axis represents the block number, and the y-axis represents the classification accuracy. Obviously, the performance of both NB and LR is much better when higher percentages of labeled data are considered. Note also that in this data set LR always outperforms NB, which is mainly due to the small percentages of labeled data. In fact, as also pointed out in [1], the presence of only few labeled data may lead to poor estimates of the generative approach.

Drift detection results for mushroom data set

	Feature change		Conditional change					
			2% labeled		5% labeled		10% labeled	
blocks	kl_{fc}	$\hat{kl}_{fc}^{\alpha}$	kl_{cc}	$\hat{kl}_{cc}^{\alpha}$	kl_{cc}	$\hat{kl}_{cc}^{\alpha}$	kl_{cc}	$\hat{kl}_{cc}^{\alpha}$
1–2	**0.2251**	**0.0450**	0.0003	0.1430	0.0001	0.0035	0.0079	0.0176
2–3	0.0365	0.0755	0.0019	0.0156	0.0005	0.0030	0.0003	0.0009
3–4	0.1184	0.1536	0.0031	0.0049	0.0054	0.0057	0.0002	0.0181
4–5	**1.2973**	**0.2373**	0.0043	0.1347	0.0002	0.0063	0.0013	0.0030
5–6	0.0007	0.0814	0.0028	0.0105	0.0018	0.0092	0.0001	0.0053

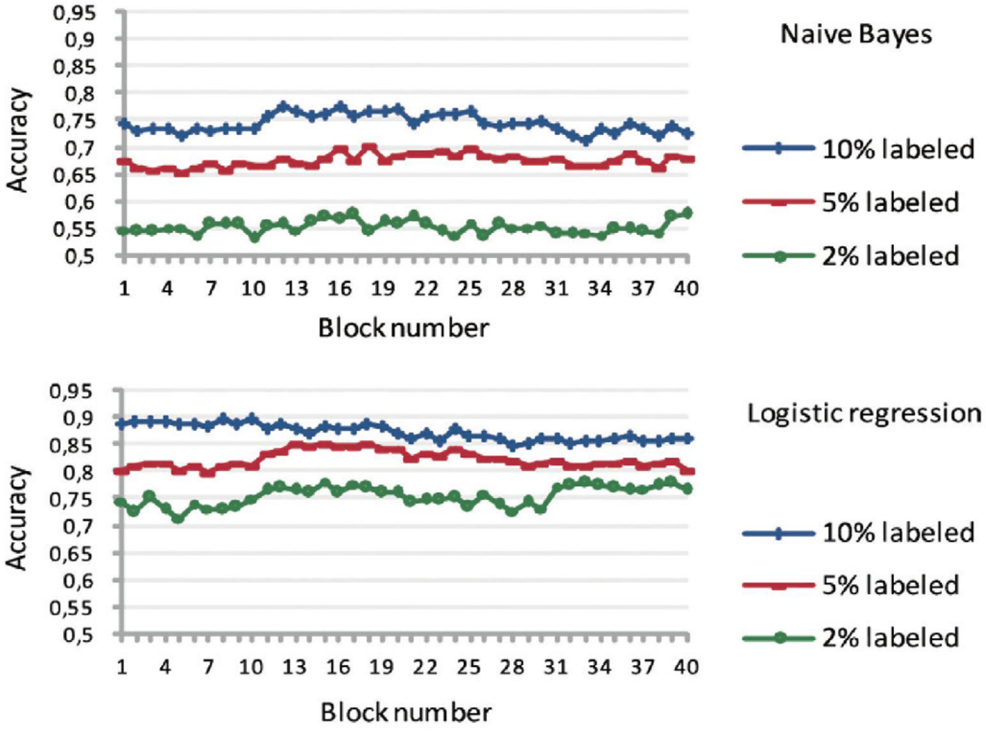

Fig. 1. Classification results of NB and LR for rotating hyperplane data set.

5.2.2. Results with mushroom data set

According to the results in Table 3, feature changes are only detected between blocks 1 and 2, and blocks 4 and 5. However, no conditional changes are detected for any of the percentages of labeled data as expected. This proves that our detection method is resilient to false alarms.

Moreover, according to Fig. 2, using more labeled data improves the predictive accuracies of both NB and LR. Nevertheless, the improvement is negligible for LR from 5% to 10% of labeled data and the corresponding curves are almost superimposed. Notice also that LR always outperforms NB and has a more stable behavior especially when more labeled data are used.

5.2.3. Results with malware detection data set

Table 4 presents drift detection results for the malware detection data set. The first column reports as previously the block numbers, while the second column represents the percentage of labeled instances

Table 4
Drift detection results for malware detection data set

blocks	% labeled instances	kl_{fc}	$\hat{kl}^{\alpha}_{fc}$	kl_{cc}	$\hat{kl}^{\alpha}_{cc}$
1–2	70.00–80.45	0.1304	0.2049	0.0797	0.9180
2–3	80.45–69.30	0.1057	0.1553	0.1609	0.2062
3–4	69.30–67.10	**0.3892**	**0.1408**	**0.4330**	**0.2717**
4–5	67.10–78.35	0.1272	0.1939	0.2975	0.3473
5–6	78.35–71.25	0.1095	0.3717	0.0190	0.1652
6–7	71.25–74.85	0.0665	0.2760	0.5373	0.6614
7–8	74.85–76.05	0.0967	0.1434	0.5028	0.5484
8–9	76.05–83.80	**1.0006**	**0.2445**	**0.7558**	**0.6033**
9–10	83.80–78.85	0.1221	0.9668	0.3390	0.4493

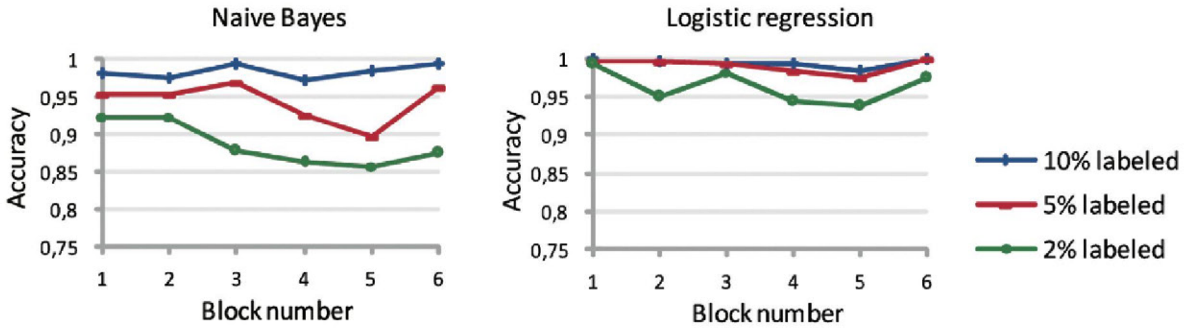

Fig. 2. Classification results of NB and LR for mushroom data set.

in each considered block. Then, columns 3 to 8 show respectively kl_{fc}, $\hat{kl}^{\alpha}_{fc}$, kl_{cc} and $\hat{kl}^{\alpha}_{cc}$ values. We observe that feature and conditional changes occur together and are detected between blocks 3 and 4, and again between blocks 8 and 9.

To evaluate classifier performance, we previously used only the overall classification accuracy. However, when dealing with imbalanced data sets, this metric is often insufficient, as it does not distinguish between the number of correctly classified instances of different classes.

Using balancing methods mainly aims to improve the classifier performance over the positive class, i.e. reduce the number of false positives. In order to appropriately monitor the behavior of NB and LR classifiers on the positive class in this case, then, we also calculate the precision, recall, F1 and G-mean metrics based on the confusion matrix analysis.

The results of the NB and LR classifiers for the first balancing approach SERA are described in Table 5. We observe that, in most cases, LR accuracies are slightly higher than for NB.

Furthermore, both NB and LR provide high precision values for all testing blocks, where all values are greater than 95%, and yield good results in terms of F1 and G-mean values, which is indicative of a good performance predicting the positive instances.

For the clustering-sampling balancing approach, as shown in Table 6, LR outperforms NB, except on the last two testing sets, where NB shows better accuracies, as well as better recall and F1 values.

Finally, note that the results of the two applied balancing approaches are comparable, with a slightly better performance of the clustering-sampling approach in terms of overall accuracy, recall and F1 metrics. In most cases, though, SERA provides slightly better precision with both NB and LR classifiers.

Table 5
Performance results of NB and LR for malware detection data set using SERA balancing approach

blocks	Algo.	Accuracy	Recall	Precision	F1	G-mean
1	NB	0.7063	0.6392	0.9792	0.7735	0.7795
	LR	0.7211	0.6483	0.9942	0.7848	0.7996
2	NB	0.7995	0.7701	0.9613	0.8551	0.8312
	LR	0.7212	0.6696	0.9537	0.7868	0.7729
3	NB	0.7540	0.7354	0.9838	0.8417	0.8148
	LR	0.7951	0.7788	0.9883	0.8711	0.8491
4	NB	0.7970	0.7375	0.9656	0.8363	0.8316
	LR	0.8328	0.7846	0.9722	0.8684	0.8620
5	NB	0.7668	0.7330	0.9818	0.8393	0.8270
	LR	0.7787	0.7409	0.9904	0.8477	0.8455
6	NB	0.7462	0.7289	0.9781	0.8353	0.7995
	LR	0.8051	0.7906	0.9858	0.8775	0.8503
7	NB	0.7488	0.7019	0.9890	0.8211	0.8227
	LR	0.7576	0.7111	0.9911	0.8281	0.8308
8	NB	0.8385	0.8334	0.9993	0.9089	0.9058
	LR	0.8006	0.7964	0.9966	0.8853	0.8568
9	NB	0.7625	0.7551	0.9746	0.8509	0.7907
	LR	0.7339	0.7182	0.9799	0.8289	0.7909
10	NB	0.8044	0.7878	1.0000	0.8813	0.8876
	LR	0.8011	0.7884	0.9947	0.8797	0.8658

Table 6
Performance results of NB and LR for malware detection data set using clustering-sampling balancing approach

blocks	Algo.	Accuracy	Recall	Precision	F1	G-mean
1	NB	0.6992	0.6279	0.9823	0.7661	0.7759
	LR	0.7087	0.6347	0.9906	0.7737	0.7879
2	NB	0.7546	0.7226	0.9447	0.8188	0.7884
	LR	0.8455	0.8344	0.9592	0.8925	0.8581
3	NB	0.7793	0.7696	0.9774	0.8611	0.8122
	LR	0.8381	0.8314	0.9840	0.9013	0.8609
4	NB	0.7508	0.7042	0.9231	0.7989	0.7787
	LR	0.8078	0.7745	0.9418	0.8500	0.8287
5	NB	0.7936	0.7645	0.9834	0.8602	0.8462
	LR	0.8590	0.8389	0.9899	0.9082	0.8964
6	NB	0.8014	0.7949	0.9756	0.8760	0.8221
	LR	0.8753	0.7837	0.9832	0.9252	0.8807
7	NB	0.8403	0.8240	0.9781	0.8945	0.8684
	LR	0.9359	0.9304	0.9909	0.9597	0.9456
8	NB	0.8795	0.8966	0.9769	0.9350	0.5918
	LR	0.9195	0.9238	0.9924	0.9569	0.8580
9	NB	0.8457	0.8735	0.9506	0.9104	0.7253
	LR	0.8264	0.8435	0.9582	0.8972	0.7559
10	NB	0.9038	0.9142	0.9802	0.9460	0.8453
	LR	0.8648	0.8623	0.9897	0.9217	0.8782

6. Conclusion

We deal with a more realistic and important problem in data stream mining, which most existing research has failed to address assuming data streams to be entirely labeled. In our research, using both labeled and unlabeled instances, we not only assert the presence or absence of drift but we also efficiently determine which kind of drift has occurred -feature, conditional or dual-using Kullback-Leibler divergence and a bootstrapping method. Then, if required, we update the classifier using the EM algorithm. Experimental results with naive Bayes and logistic regression show that our approach is effective for detecting different kinds of changes from data containing both labeled and unlabeled instances, as well as having a good classification performance.

In the future, it would be interesting to investigate and compare the performance of other classifiers with our results. Furthermore, note that in this paper we assume that labeled and unlabeled data come from the same distribution. This usually leads to a better classification accuracy. An interesting future line of research would be to consider the scenario where labeled and unlabeled data possibly come from different distributions, inspect the impact of unlabeled data, and study the possibility of refining the change detection proposal.

Acknowledgements

This work has been supported by the Spanish Ministry of Science and Innovation under projects TIN2007-62626, TIN2008-06815-C02-02, Consolider Ingenio 2010-CSD2007-00018, and by the Cajal Blue Brain project.

[1] M.R. Amini and P. Gallinari, Semi-supervised logistic regression, in: *Fifteenth European Conference on Artificial Intelligence*, 2002, pp. 390–394.

[2] A. Asuncion and D.J. Newman, UCI Machine Learning Repository, University of California, Irvine, http://www.ics.uci.edu/∼mlearn/ MLRepository.html, 2007.

[3] A. Bifet and R. Gavalda, Learning from time-changing data with adaptive windowing, in: *Proceedings of the 7th SIAM International Conference on Data Mining*, 2007, pp. 29–40.

[4] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby and R. Gavald, New ensemble methods for evolving data streams, in: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009, pp. 139–148.

[5] S. Chen and H. He, SERA: Selectively recursive approach towards nonstationary imbalanced stream data mining, in: *Proceedings of the International Joint Conference of Neural Networks*, 2009, pp. 522–529.

[6] T. Dasu, S. Krishnan, S. Venkatasubramanian and K. Yi, An information-theoretic approach to detecting changes in multi-dimensional data streams, *in: Proceedings of the 38th Symposium on the Interface of Statistics, Computing Science, and Applications*, 2006, pp. 1–24.

[7] A.P. Dempster, N.M. Laird and D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, *Journal of the Royal Statistical Society* **39** (1977), 1–38.

[8] B. Efron and R. Tibshirani, Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy, *Statistical Science* **1**(1) (1986), 54–75.

[9] F. Fleuret, Fast binary feature selection with conditional mutual information, *Journal of Machine Learning Research* **5** (2004), 1531–1555.

[10] G. Hulten, L. Spencer and P. Domingos, Mining time changing data streams, in: *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining*, 2001, pp. 97–106.

[11] D.W. Hosmer and S. Lemeshow, *Applied Logistic Regression*, John Wiley and Sons, New York, 2nd edition, 2000.

[12] J. Gama and G. Castillo, Learning with local drift detection. *in: Proceedings of the 2nd International Conference on Advanced Data Mining and Applications*, 2006, pp. 42–55.

[13] J. Gama, R. Fernandes and R. Rocha, Decision trees for mining data streams, *Intelligent Data Analysis* **10**(1) (2006), 23–45.

[14] J. Gao, B. Ding, W. Fan, J. Han and P.S. Yu, Classifying data streams with skewed class distributions and concept drifts, *IEEE Internet Computing* **12**(6) (2008), 37–49.

[15] D. Kifer, S. Ben-David and J. Gehrke, Detecting change in data streams, in: *Proceedings of the 30th International Conference on Very Large Data Bases*, 2004, pp. 180–191.

[16] R. Klinkenberg, Learning drifting concepts: Example selection vs. example weighting, *Intelligent Data Analysis* **8**(3) (2004), 281–300.

[17] R. Klinkenberg, Using labeled and unlabeled data to learn drifting concepts, in: *Proceedings of International Joint Conference on Artificial Intelligence*, 2001, pp. 16–24.

[18] R. Klinkenberg and I. Renz, Adaptive information filtering: Learning in the presence of concept drifts, *Workshop Notes of the ICML/AAAI-98 Learning for Text Categorization*, 1998, pp. 33–40.

[19] J.Z. Kolter and M.A. Maloof, Dynamic weighted majority: An ensemble method for drifting concepts, *Journal of Machine Learning Research* **8** (2007), 2755–2790.

[20] S. Kullback and R.A. Leibler, On information and sufficiency, *The Annals of Mathematical Statistics* **22**(1) (1951), 79–86.

[21] L.I. Kuncheva and I. Žliobaite, On the window size for classification in changing environments, *Intelligent Data Analysis* **13**(6) (2009), 861–872.

[22] M.M. Masud, J. Gao, L. Khan, J. Han and B. Thuraisingham, A practical approach to classify evolving data streams: Training with limited amount of labeled data, *The 8th IEEE International Conference on Data Mining*, 2008, pp. 929–934.

[23] M. Minsky, Steps towards artificial intelligence, *Computers and Thought* (1961), 406–450.

[24] K. Nigam, A. McCallum, S. Thrun and T. Mitchell, Text classification from labeled and unlabeled documents using EM, *Machine Learning* **39**(2/3) (2000), 103–134.

[25] K. Nishida and K. Yamauchi, Detecting concept drift using statistical testing, in: *Proceedings of International Conference of Discovery Science*, 2007, pp. 264–269.

[26] R. Sebastião and J. Gama, Change detection in learning histograms from data streams, in: *Proceedings of the 13th Portuguese Conference on Artificial Intelligence EPIA*, 2007, pp. 112–123.

[27] A. Tsymbal, The problem of concept drift: Definitions and related work, *Technical Report TCD-CS-2004-15*, Department of Computer Science, Trinity College Dublin, Ireland, 2004.

[28] A. Tsymbal, M. Pechenizkiy, P. Cunningham and S. Puuronen, Dynamic integration of classifiers for handling concept drift, *Information Fusion* **9**(1) (2008), 56–68.

[29] P. Vorburg and A. Bernstein, Entropy-based concept shift detection, *in: Proceedings of the 6th International Conference on Data Mining*, 2006, pp. 1113–1118.

[30] H. Wang, W. Fan, P. Yu and J. Han, Mining concept drifting data streams using ensemble classifiers, in: *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 226–235.

[31] Y. Wang, Y. Zhang and Y. Wang, Mining data streams with skewed distribution by static classifier ensemble, in: *Proceedings of the 22nd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, 2009, pp. 65–71.

[32] G. Widmer and M. Kubat, Learning in the presence of concept drift and hidden contexts, *Machine Learning* **23**(1) (1996), 69–101.

[33] P. Zhang, X. Zhu and Y. Shi, Categorizing and mining concept drifting data streams, in: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008, pp. 812–820.

[34] X. Zhu, Semi-supervised learning literature survey, *Technical Report*, Computer Sciences, University of Wisconsin-Madison, 2008.