Knowl Inf Syst (2014) 40:489–507
DOI 10.1007/s10115-013-0654-6

REGULAR PAPER

Recurrent concepts in data streams classification

João Gama · Petr Kosina

Received: 28 October 2011 / Revised: 5 February 2013 / Accepted: 27 April 2013 /
Published online: 31 May 2013

Abstract This work addresses the problem of mining data streams generated in dynamic environments where the distribution underlying the observations may change over time. We present a system that monitors the evolution of the learning process. The system is able to self-diagnose degradations of this process, using change detection mechanisms, and self-repair the decision models. The system uses meta-learning techniques that characterize the domain of applicability of previously learned models. The meta-learner can detect recurrence of contexts, using unlabeled examples, and take pro-active actions by activating previously learned models. The experimental evaluation on three text mining problems demonstrates the main advantages of the proposed system: it provides information about the recurrence of concepts and rapidly adapts decision models when drift occurs.

Keywords Data streams · Concept drift · Meta-learning · Recurrent concepts

1 Introduction

Several authors stress that the relevance of information and data depends on the context [8, 10, 20]. In this work, we study stream mining problems where contexts change over time and might reoccur. We consider online learning systems that continuously maintain a decision model from high-speed data streams generated by dynamic environments. The unknown dynamics of the processes generating streaming data requires that learning algorithms monitor the learning process and self-diagnose changes in the context of learning. We propose a

J. Gama (✉) · P. Kosina
LIAAD-INESC TEC, Porto, Portugal
e-mail: jgama@fep.up.pt

J. Gama
FEP-University of Porto, Porto, Portugal

P. Kosina
FI Masaryk University, Brno, Czech Republic

generic framework that identifies contexts using drift detection, characterizes contexts using meta-learning techniques, and selects the most appropriate base model for the incoming data using unlabeled examples. From an AI perspective, the proposed algorithm is self-stabilizing [3] in the sense that, starting from an arbitrary state, it converges to a legitimate state and remains in a legitimate set of states thereafter. In this paper, learning is no more a one-shot run but a process that evolves over time. We monitor the learning process and reason about how it evolves. The advantage of identifying reappearing context is that previously learned model can give performance boost for the predictions after drift where new classifier would be under-trained. It is expected that over time the performance of a new model and a reused model will converge to the similar performances.

For illustrative purposes, consider a sensor network. Sensors are geographically distributed and produce high-speed distributed data streams. They measure some quantity of interest, for example, the electricity demand in a particular geographic region. Companies are interested in predicting, for each sensor, the demand in the corresponding geographic region, for different time horizons. Assume that at time t our predictive model makes a prediction $\hat{y}_{t+k}$ for time $t + k$, where k is the desired horizon forecast. Later on, at time $t + k$, the sensor measures the quantity of interest y_{t+k}. With a delay k, we can estimate the loss of our prediction $L(\hat{y}_{t+k}, y_{t+k})$. Moreover, electricity consumption depends on the weather conditions and evolves over time. Consumption patterns change (e.g. from winter to summer), and, due to seasonality, might reoccur. This is a particular example that demonstrates some of the challenges we are faced nowadays: the decisions must be taken in real time, although the feedback is available later on.

This is the general framework we address in this work. We propose a framework that detects changes in the processing generating data by monitoring the learning process using drift detection techniques. When a change is detected, the learned model is stored in a *sleep* mode for possible reuse later. We use meta-learning techniques [16] to decide when to reuse one of the *sleeping* decision models. The main contribution of the proposed method is that it is able to use information from unlabeled examples to select and reuse a previously learned model. The proposed system is able to provide explicitly information about the recurrence of patterns.

The paper is organized as follows. The next section presents related work in change detection, recurrence of concepts, and context-sensitive learning. Section 3 presents a general overview of the proposed framework. Section 4 discusses results of the experimental evaluation using real-world problems. The last section presents conclusions and future research lines.

2 Related work

The stream mining community has already introduced many different approaches to deal with the phenomenon of concept drift. Suppose a supervised learning problem, where the learning algorithm observe sequences of pairs $(\vec{x}_i, y_i)$ where $y_i \in \{C_1, C_2, \ldots, C_k\}$. At each time-stamp t, the learning algorithm outputs a class prediction $\hat{y}_t$ for the given feature vector $\vec{x}_t$. Assuming that examples are independent and generated at random by a stationary distribution $\mathcal{D}$, some model class algorithms (e.g. decision trees, neural networks) can approximate $\mathcal{D}$ with arbitrary precision (bounded by the *Bayes error*) whenever the number of examples increases to infinite.

Suppose now the case where $\mathcal{D}$ is not stationary. The data stream consists of sequences of examples $e_i = (\vec{x}_i, y_i)$. Suppose further that from time to time, the distribution that generates the examples change. The data stream can be seen as sequences $\langle S_1, S_2, \ldots, S_k, \ldots\rangle$ where each element S_i is a set of examples generated by some stationary distribution $\mathcal{D}_i$. We designate as *context* each one of these sequences. In that case, and in the whole dataset, no learning algorithm can guarantee arbitrary precision. Nevertheless, if the number of observations within each sequence S_i is large enough, we could approximate a learning model to $\mathcal{D}_i$. The main problem is to detect change points whenever they occur. In real problems between two consecutive sequences S_i and S_{i+1}, there could be a transition phase where some examples of both distributions appear mixed. An example generated by a distribution $\mathcal{D}_{i+1}$ is noise for distribution $\mathcal{D}_i$. This is another difficulty faced by change detection algorithms. They must differentiate *noise* from *change*. The difference between noise and examples of another distribution is *persistence*: there should be a consistent set of examples of the new distribution. Algorithms for change detection must combine *robustness* to noise with *sensitivity* to concept change [5].

In this work, we are interested in identifying contexts, that is, parts of the stream where the process generating data is stationary. While blind adaptation methods, for example, sliding windows and example weights [13], are widely used to maintain a classifier consistent with the most recent data, they do not provide information about the unknown dynamics of the process generating data. Given the goals of the proposed system, the context identification, much more relevant methods are those that explicitly detect change points or small time-windows where the concept to learn has changed. In particular, we are interested in single classifier methods equipped with drift detection and forgetting mechanism [6], although the exploitation of ensemble methods [19] might be relevant.

The related work can be grouped into two main categories: context-sensitive learning and methods for recurrent concepts. The following sections review the main works in both categories.

2.1 Context-sensitive learning

It is known that the concept definitions which have been learned in one context can become invalid in a new context. In the context-sensitive learning, it is assumed that there exist contextual features, which might not be relevant for the discriminative task, but are useful for identifying the context of learning. When these features are not available, a concept drift is referred to as a change in hidden variables. Context-sensitive learning that deals with changes in hidden variables is relevant for the proposed system. Turney [20] provides a review of context-sensitive features and correspondent strategies to handle them. Features in learning tasks are distinguished as primary, contextual, and irrelevant. The context features can be known and explicitly represented or can be hidden. To cope with the influence of known context features, five strategies were presented: contextual normalization, expansion (feature space of primary features is expanded with contextual features), classifier selection (classifier trained on contextual features selects specialized classifier for primary features), classification adjustment (first, it uses classifier of primary features and then final decision is adjusted based on contextual features), and weighting (contextual features give weight to primary features). Also, the performance of the combinations of some of them is discussed. There are two methods aiming to recover the missing context, which are unsupervised clustering and the information about the temporal sequence of the instances. In the former, the main idea is that members of the same cluster are likely to share the same class and context. Therefore, after clustering on the primary features, we can label the clusters and introduce a new contextual feature. The latter approach is based on the idea that events occurring close together in time tend to share the context. A work that exploits the idea of using the time attribute was introduced by Harries et al. [10]. The work presents the Splice system, a meta-learning algorithm that implements a context-sensitive batch learning approach. Splice is designed to identify intervals with stable hidden context and to induce and refine local concepts associated with these hidden contexts. The main idea consists of using a time-stamp of the examples as an attribute for a batch classifier. In the first stage, examples are augmented with a time-stamp attribute, and a decision tree inducer learns a decision tree. If the decision tree finds splits on the time-stamp attribute, the partitions on that attribute suggest different contexts. In the second stage, C4.5 is applied to each partition to find interim (temporal) concepts. In the next stage, all examples and interim concepts are given a score based on the accuracy of the concepts in given fixed window (over the time-stamp). This is followed by clustering examples with highest score for the same concept. Another application of C4.5 then creates new interim concepts.

2.2 Recurrent concepts

An approach which employs meta-learning and contextual information was presented in Widmer [21], Widmer and Kubat [22]. First, two types of attributes are defined: predictive attributes and contextual attributes. The predictive attributes are correlated with the class values and are used in the predictive model. The contextual attributes are not correlated with the class values but are useful to predict the predictive attributes. The predictive characteristic is decided via statistical χ^2 test over the sufficient statistics kept for Bayesian learning.

A pioneer method for recurrent concepts was presented by Lazarescu [14]. The method uses multiple windows for tracking the concept drift. The size of the window can be adapted by predicting the rate of change. When drift is estimated, the repository of stored historical concepts is checked for recurrence. Concepts are described by averages of attributes (numeric), and similarity can be measured by any feature distance metric. Also, Yang et al. [23] present a method for reusing previously learned concepts. It uses a proactive approach which selects previously learned concepts that will most likely follow after the current concept according to a transition matrix. Learned concepts are treated like a Markov Chain with concepts as states. An ensemble of classifiers with recurrence is presented by Ramamurthy and Bhatnagar [17]. The authors propose that classifiers of new concepts are stored in global set and only the models of certain quality are part of the ensemble, which is then responsible for labeling the examples. Classifiers are tested, selected to ensemble or a new one is created with every incoming chunk of labeled examples. Katakis et al. [12] present the conceptual clustering and prediction (CCP) framework to handle streaming data and identify recurrent concepts. The data stream is divided into short batches of examples, which are transformed into conceptual vectors. Conceptual vectors describe the batches by their mean and standard deviation for numeric attributes, and probability of attribute given the class for nominal attributes. The vectors are clustered by an incremental clustering algorithm, which either assigns the vector of a new batch to an existing cluster or creates a new cluster. For each cluster, a classifier is learned. Each batch is classified by a classifier that is related to a cluster of the conceptual vector of the batch.

In this work, we present an approach to selecting older models learned on data with similar underlying context. A single classifier is used for predicting the class labels, and a meta-learner is used to select the most appropriate classifier from history. Every time a new classifier is not sufficient and a warning is signaled, meta-learners provide predictions of the performance of the older classifiers. Whenever a drift is detected, the classifier and its meta-classifier are stored in a pool for further use. A classifier is reused when the percentage of votes of its meta-classifier exceeds some given threshold; otherwise a new one is learned. The idea of using similar meta-classifiers can be found in Ortega [15], Ortega et al. [16], where the meta-learning scheme is used in offline learning to select classifiers from an ensemble. Later, Seewald and Fürnkranz [18] report an extensive evaluation of different meta-learners for batch learning. We use a similar scheme in an online scenario to characterize the domain of applicability of classifiers.

3 The two-layer learning system

In this section, we present an overview of the main ideas behind the proposed system, which detects contexts and their recurrence. The system uses a two-layer learning scheme. The first layer ($layer_1$) is the learning layer, where we learn a decision model to solve the original decision problem. The second layer ($layer_2$) is a control layer. The goal of this layer is to monitor the evolution of the learning process on the first layer.

3.1 Learning with model applicability induction

As mentioned above, the proposed system uses a two-layer learning scheme. Each layer receives its own data and trains its own classifier. The first layer receives the data stream and trains a classifier using the labeled examples. For each incoming example $(\vec{x}, y)$, the current classifier predicts a class label $\hat{y}$. If the example is not labeled, that is y is unknown, the current model classifies the example and proceeds to the next example. If the example is labeled, it is possible to compute the loss, $l(y, \hat{y})$, and update the current decision model with the example. Assuming the 0–1 loss function, the prediction is either correct or incorrect, and an example is generated to train the second layer classifier. The example for the meta-classifier on the second layer has the same attribute values as for the classifier on the first layer, but the class label is either *True*, if the example was correctly classified, or *False*, if the example was misclassified. The decision problem on $layer_2$ is always binary. A data point on $layer_2$ is either $(\vec{x}, True)$, meaning that the base classifier correctly classified the original example, or $(\vec{x}, False)$, meaning that the $layer_1$ classifier misclassified the original example.

As in real situations, there is delay between obtaining example and observing true label. The $layer_1$ classifier makes the prediction whenever new examples are available. Later on, once the true value of example's class is observed, the current model makes a new prediction for that example, so that the evaluation would reflect the current state of the $layer_1$ model. Then, the classifier is updated and the example is passed to the control layer with a new class attribute, which reflects whether or not the prediction was correct. The process is illustrated in Fig. 1. The meta-classifier is learning in parallel with the $layer_1$ classifier. This way, the meta-classifier learns the regions of the instance space where the classifier performs well, in other words, the context where the classifier is applicable.

3.2 The control layer: change detection and model management

When dealing with possibly infinite data streams, changes in the process generating data can be expected. The changes can occur due to changes in the context, changes in hidden variables, or changes in some characteristic properties of data. Moreover, contexts can reappear over time. Therefore, it is desirable to have a mechanism to recognize if an older model is appropriate for new data. The proposed system monitors the learning process of the $layer_1$ classifier using a change detection algorithm that monitors the evolution of a performance measure. Several change detection algorithms that trace the evolution of the error rate can be

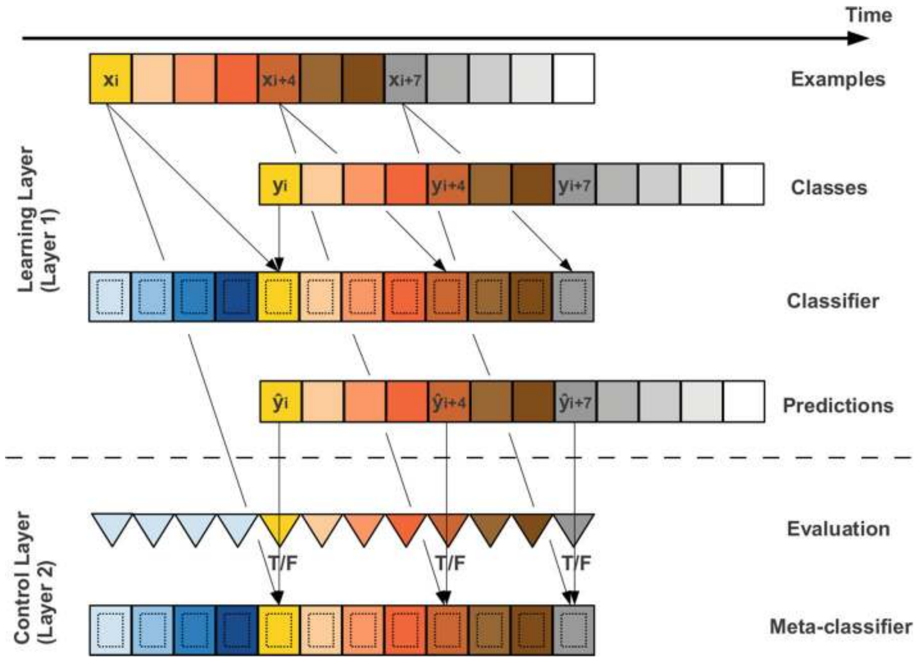

Fig. 1 The learning process. When the true label is obtained, the example is used to train the $layer_1$ classifier. Then, the prediction is evaluated and the example's class label is replaced by true/false label and used to train the $layer_2$ classifier

used. For example, the SPC algorithm [6], the ADWIN algorithm [2], or EDDM [1]. When a change in the context is signaled, the current classifier and the corresponding meta-classifier are stored in a long-term memory of models (see Fig. 2) for possible future use.

3.2.1 Drift detection

In the current implementation of the system, we use a modified version of the drift detection method presented in Gama et al. [6] mainly because it is able to signal not only drifts, but also warnings. Other online detection methods might be applied, but the ability to signal warnings is beneficial for proposed system. The labeled examples occurring between warning and drift signals are used in relearning the decision model.

The SPC monitors the evolution of the learning process by monitoring the error rate. The pseudo-code of SPC is described in Algorithm 1. The learning process might be in 3 stages: *in-control*, *out-of-control*, or in *warning*. The SPC algorithm manages two registers during the training of the learning algorithm, $p_{\min}$ and $s_{\min}$, where p is error rate and s is standard deviation. Every time a new example i is processed, the values in the registers are updated when $p_i + s_i$ is lower than $p_{\min} + s_{\min}$.

In the experiments reported here, we follow the 3-sigma rule [9]: the warning level is reached if $p_i + s_i \geq p_{\min} + 2 \times s_{\min}$ and the drift level is reached if $p_i + s_i \geq p_{\min} + 3 \times s_{\min}$. The rational behind the SPC is as follows: under the 0–1 loss function, the error is a Bernoulli trial, described by a binomial distribution. For large enough observations, the error rate can be approximated using a Gaussian distribution. The probability that an observation deviates from its expected value plus 3* sigma is <99.7 %. Suppose a sequence of examples where

Algorithm 1: The SPC algorithm for drift detection

```
Input  : $\Phi$: Current decision model;
         Current labeled example: $\vec{x}_j, y_j$ ;
         Prediction for current labeled example $\hat{y}_j$;
Output: Status $\in$ {InControl, Warning, Out-of-Control}
begin
    Let $error_j \leftarrow L(\hat{y}_j, y_j)$;
    Compute error's mean $p_j$ and variance $s_j$;
    if $p_j + s_j < p_{min} + s_{min}$ then
        $p_{min} \leftarrow p_j$;
        $s_{min} \leftarrow s_j$;
    if $p_j + s_j < p_{min} + 2 \times s_{min}$ then
        /* In-Control                                              */
        Status ← 'InControl' ;
        FirstTime? ← TRUE;
    else
        if $p_j + s_j < p_{min} + 3 \times s_{min}$ then
            /* Warning Zone                                        */
            Status ← 'Warning';
            if FirstTime? then
                Reset $STM_{labeled}$;
                FirstTime? ← FALSE ;
            Add $\{\vec{x}_j, y_j\}$ to $STM_{labeled}$ ;
        else
            /* Out-of-Control                                      */
            Status ← 'Out-of-Control';
            $p_{min} \leftarrow 1$;
            $s_{min} \leftarrow inf$;
    Return: Status;
end
```

the error of the actual model increases reaching the warning level at example k_w, and the drift level at example k_d. A new decision model is learned using the examples starting in k_w till k_d. It is possible to observe an increase in the error reaching the warning level, followed by a decrease. We assume that such situations correspond to a false alarm, without changing the context.

3.2.2 The short-term memory

The most recent examples in the stream are unlabeled. They are stored in a short-term memory (STM) for later use when the system receives the corresponding class label.

The STM might contain also labeled examples. The drift detection algorithm works with labeled examples. While SPC is in *warning* status, these labeled examples are stored in STM in addition to the unlabeled recent examples. They are used both to relearn a new decision model when the *out-of-control* is signaled and to evaluate the performance of the meta-classifiers, that is, to determine whether any of the decision models that are in *sleeping mode* correspond to the context that is currently generating data. The structure of the STM is shown in Fig. 3.

3.3 The recurrent meta algorithm

When the drift detection algorithm signals a change, the system must decide between:

- Start learning a new decision model.

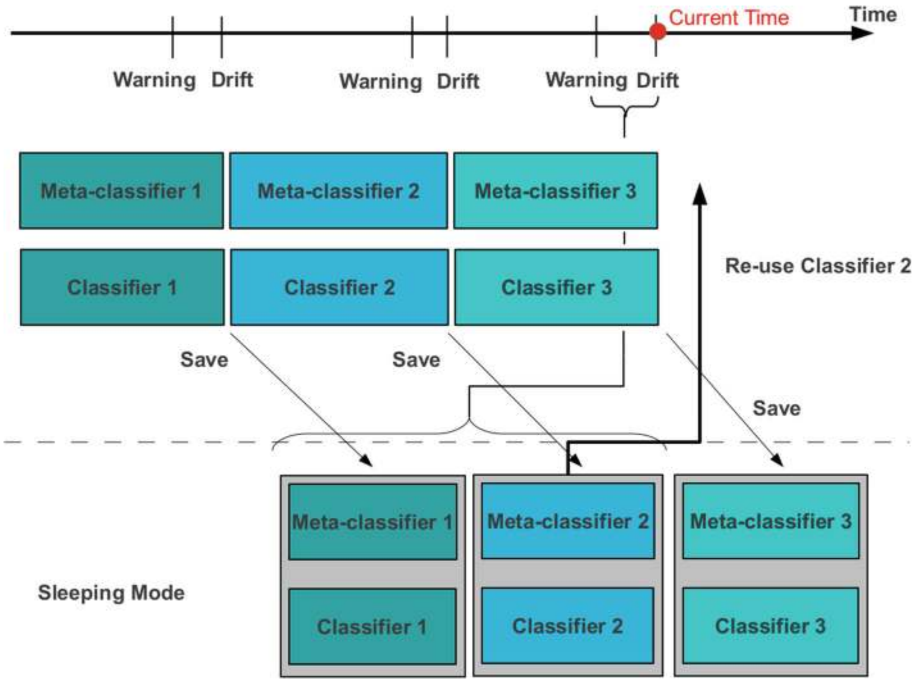

Fig. 2 When a drift is signaled, the current model and the corresponding meta-classifier are stored in a pool of models in sleeping mode. The best model is reused for the incoming examples only if its applicability is greater than a given threshold, otherwise a new classifier and its corresponding meta-classifier are trained

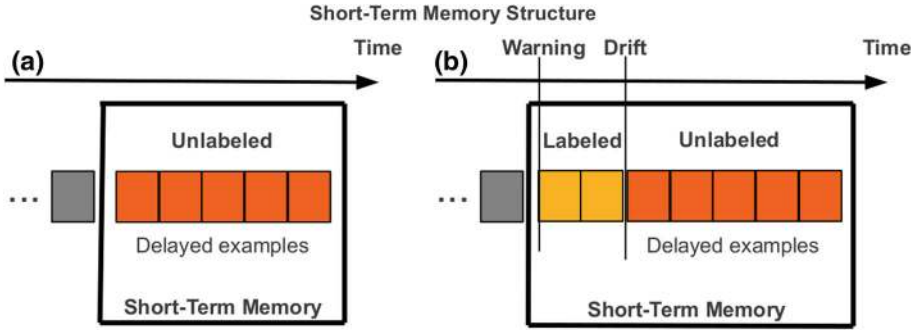

Fig. 3 The structure of short-term memory. **a** STM during the stationary context learning contains only the most recent examples which are unlabeled (the true label is delayed). **b** When a drift is detected, STM contains also labeled examples from window between *warning* and *out-of-control* signaled by SPC algorithm in addition to the unlabeled examples

– Activate one of the previously learned models.

The system uses the meta-classifiers to predict the performance of their corresponding $layer_1$ models. Each meta-classifier makes a prediction for all the labeled and unlabeled examples in STM. The prediction estimates whether the corresponding classifier will correctly classify the example. This way, we can estimate the error of the $layer_1$ decision models stored in sleeping mode. In order to decide whether we should use one of the previous learned

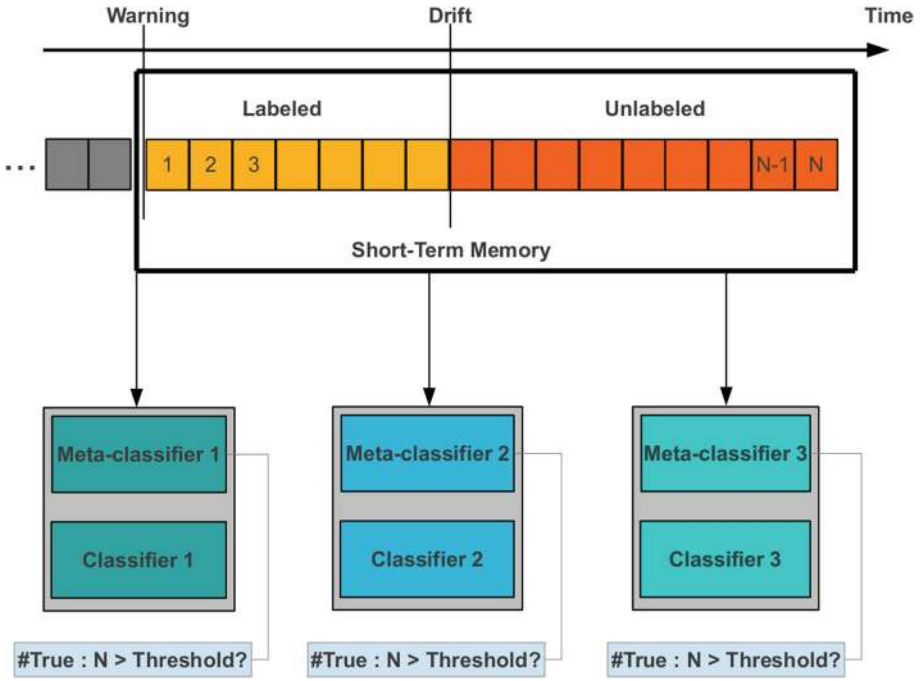

Fig. 4 At the drift point, STM contains labeled examples, from the warning signal till drift signal, and unlabeled examples. All the examples in STM are used by meta-classifiers to estimate the performance of the corresponding $layer_1$ classifier in the most recent data

models or learn a new classifier, we consider the classifier with lowest estimated error. If the lowest estimated error is lower than a *threshold*, it is selected to be used for evaluating the next incoming examples. The threshold, data dependent, is defined to be the error of the active $layer_1$ model. Otherwise, we start learning a new classifier and its meta-classifier. The process is illustrated in Fig. 2. In this process, we use unlabeled data. This is the main advantage of using meta-classifiers. The process is illustrated in Fig. 4. This way, it is possible to pro-actively search for and select recurrent models. The recurrent meta algorithm for monitoring model evolution is presented in Algorithm 2. This algorithm is triggered at time t when we receive $y_{t-\Delta t}$, the label of example $t - \Delta t$. This example has been previously classified at time $t - \Delta t$. Meantime, the decision model has evolved by learning from the labeled examples meanwhile available. Using the current decision model, a new prediction is generated for the example $t - \Delta t$ and the error is computed using this prediction.

Note that $layer_2$ training data might be unbalanced. The class distribution of $layer_2$ training examples reflect the error rate of the $layer_1$ classifier. A good performance of the $layer_1$ classifier means that there will be a lot of positive examples and few negative examples. This implies the need for a method to deal with this skewed data. The *Update meta model* part in Algorithm 2 explains how we deal with unbalanced data, minimizing the impact of a large number of positive examples. For each training example, the current $layer_2$ model predicts whether the $layer_1$ classifier decision for that example is correct or not. The prediction of $layer_2$ meta-classifier is correct in two different cases: (1) the prediction of the meta-classifier is *True* and the $layer_1$ classifier correctly classifies the example; or (2) the prediction of the meta-classifier is *False*) and the $layer_1$ classifier incorrectly classifies this example. These

Algorithm 2: The recurrent meta algorithm for monitoring model evolution

```
Input : $\Phi_1$: Current $layer_1$ decision model;
        $\Phi_2$: Current $layer_2$ (meta) decision model;
        $y_{t-\Delta t}$: Label at time $t - \Delta t$;
begin
    /*Predict label for the instance at time $t - \Delta t$                */
    Recover $\vec{x}_{t-\Delta t}$ from STM;
    $\hat{y}_{t-\Delta t} \leftarrow \Phi_1(\vec{x}_{t-\Delta t})$ ;
    /*Estimate Drift Status                                                 */
    Status $\leftarrow$ SPC($\Phi_1, \vec{x}_{t-\Delta t}, \hat{y}_{t-\Delta t}$) ;
    if Status = 'InControl' then
        /*Update current decision model                                     */
        Update $\Phi_1$ with $(\vec{x}_{t-\Delta t}, y_{t-\Delta t})$ ;
        /*Update meta model                                                 */
        $\hat{y2}_{t-\Delta t} \leftarrow \Phi_2(\vec{x}_{t-\Delta t})$;
        if $\hat{y}_{t-\Delta t} = y_{t-\Delta t}$ and $\hat{y}2_{t-\Delta t}$ = 'False' then
            Update $\Phi_2$ with $(\vec{x}_{t-\Delta t},' True')$ ;
        if $\hat{y}_{t-\Delta t} <> y_{t-\Delta t}$ and $\hat{y}2_{t-\Delta t}$ = 'True' then
            Update $\Phi_2$ with $(\vec{x}_{t-\Delta t},' False')$ ;
    if Status = 'Warning' then
        Insert $(\vec{x}_{t-\Delta t}, y_{t-\Delta t})$ in STM;
    if Status = 'Out-of-Control' then
        Store $\Phi_2$ in the pool of meta-learners ;
        foreach meta model $\Phi_{2k}$ in the pool do
            Let $hits_k$ be the number of times the meta model $\Phi_{2k}$ predicts True for the examples stored
            in STM;
        Let $\Phi_{2i}$ be the meta model with highest number of $hits_i$ ;
        if percentage of $hits_i \geq$ threshold then
            Recover $\Phi_{1i}$ and $\Phi_{2i}$ ;
        else
            Start new $\Phi_1$ and $\Phi_2$ using the labeled examples in STM;
            Release STM$_{labeled}$;
end
```

are the cases when the meta-classifier is not updated, otherwise it is updated. This way, the $layer_2$ model is the relevant examples, reducing the problem of over fit in one of the classes.

4 Experimental evaluation using real-world data

In this section, we present a simple illustrative example and experiments with three datasets from real-world applications. The real-world datasets represent much more complex problems as opposed to the illustrative example. The three problems we analyze are from text mining tasks: Emailing list, Usenet, and Spam Detection. The datasets are based on real-world problems where concepts evolve over time and the probability of concept recurrence is relatively high. The CCP system [12] for recurrent concepts was tested in similar datasets. However, the system is not publicly available and we cannot make direct comparison. For each problem, we plot the evolution of the error of 3 settings: (1) the error rate of a model without drift detection, (2) the error rate of system that learns a new model after each drift, and (3) the error rate of a model that can reuse learned models (the proposed system). For each dataset, we show 2 plots: one presenting the evolution of the error of each individual model in each context and the other the global error of the system. The error of each individual

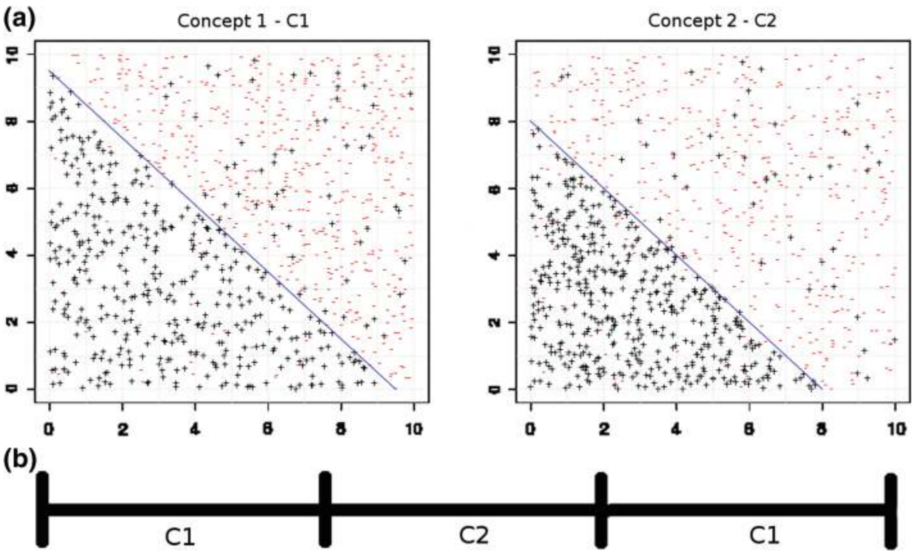

Fig. 5 Illustrative example of a data stream with recurrent concepts. **a** The two concepts generating examples with $\beta = 9.5$ and $\beta = 8$. **b** Layout of the sequence of concepts in the stream used in the experimental evaluation. Each concept is composed of 2,500 examples

classifier reflects the error of the *current* model, and it is computed since it has started to be learned. The global error is the error of the adaptive system as a whole. It is computed from the beginning of the stream.

In all the experiments reported here, the $layer_1$ and $layer_2$ algorithms are the naive Bayes classifier [4].

4.1 The SEA concepts

In this part, we present an illustrative example of how the system works. Specifically for this purpose, we use adapted benchmark dataset for time-changing stream mining—the SEA concepts [19]. The main characteristics remain as in the original dataset, i.e., it is a two-class problem, defined by three attributes, where only two are relevant. The domain of the attributes is $x_i \in [0, 10]$, where $i = 1, 2, 3$. The target concept is *positive* if $x_1 + x_2 \leq \beta$, where $\beta \in \{7, 8, 9, 9.5\}$. As opposed to the original, where there are four concepts we use only two for easier understanding. For the first concept, a *positive* example corresponds to $\beta = 9.5$ and for the second $\beta = 8$. A data stream is generated by concatenating 2,500 examples with 10 % of class noise from each concept. This way, the target concept changes over time. Figure 5a depicts the two concepts by plotting fraction of concept data with only the two relevant features. In order to illustrate the behavior on recurrent data, the first concept is generated twice. We obtained a dataset with 7,500 examples and three concepts such that concept 1 has the same threshold like concept 3. Figure 5b illustrates the stream generating process.

Examples are processed sequentially. Every time a new example is loaded, it is passed to the $layer_1$ classifier to make a prediction. To simulate behavior in real world, the true value of the class needed for online evaluation of the decision model is not observed immediately, but after some delay. In the experimental study, the delay is measured in terms of number

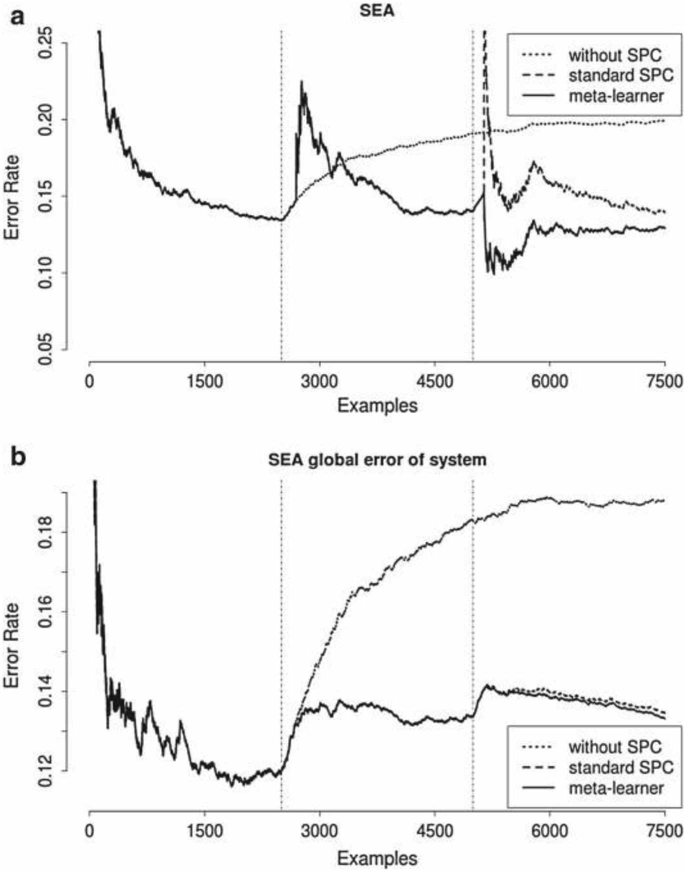

Fig. 6 *Plot* of the evolution of the error rate of each model (*top*) and the global error of the system (*bottom*) in SEA dataset

Table 1 Drift information in SEA concepts

Concept	Start	Occurs	Warning	Drift	Accuracy (%)	False alarms	Model used
1	1	2,500	2,585	2,670	86.85	0	New $model_1$
2	2,501	5,000	5,052	5,156	84.88	0	New $model_2$
3	5,001	7,500	–	–	88.36	0	Reused $model_1$

of examples. In order to provide reproducibility, this delay is fixed and can be an arbitrary number of examples.[1] Meanwhile, the examples are stored in the memory.

Handling concept drifts is a well-studied problem in data stream mining, and many algorithms can be used. Typical approaches include detecting the drift and building a new model from scratch while forgetting the old one. Furthermore, with the SPC, we can store examples in short-term memory during the warning phase and use these recent examples, representative of the new concept, for learning a new model.

[1] The default value is 250 examples.

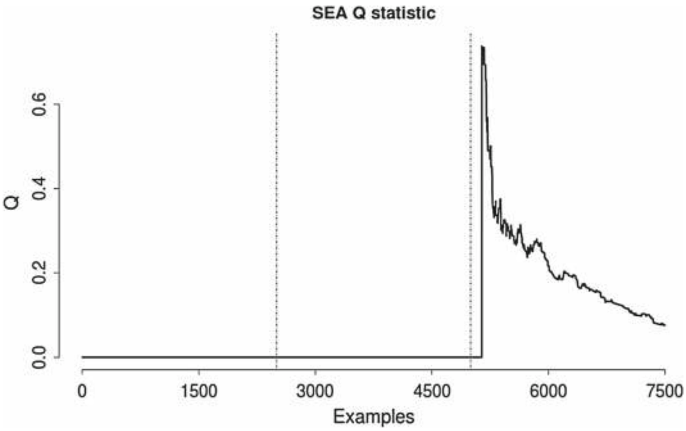

Fig. 7 *Plot* of the $Q = \log($ standard SPC/meta-classifier SPC$)$. $Q = 0$ means equal performance, while positive Q means lower error rate for meta-learner. The advantage of using the meta-classifier, in terms of accuracy, is mainly just after drift allowing fast adaptation to drift

In Fig. 6, we present the results of non-adapting approach (without SPC), naive Bayes with SPC (standard SPC), and the meta-learning approach (meta-learner) for the SEA concept dataset. The non-adapting method in changing environment does not produce satisfactory results. Since the difference between the concepts is not very large and the error rate does not increase rapidly, its curve appears below the adaptive methods when plotting the rates of each classifier. The global error rate of the system in Fig. 6 (bottom) shows that adaptive methods are superior to the non-adaptive.

Table 1 summarizes the information about the drift detection and accuracy statistics in the illustrative example. As previously mentioned, the difference between concepts is not large; therefore, the changes were not detected immediately. The first change was detected after 170 examples of the new concept, and the accuracy of the first model was 86.85 %. The detection of the second change occurred after 156 examples with the final accuracy of the second model being 84.88 %. At this point, the first classification model was selected and achieved accuracy of 88.36 %. There were no false alarms in this task.

Both adaptive methods behave exactly the same until the second drift is detected. The meta-learning approach has already one concept available in the pool of historic classifiers. Based on the predictions for data in short memory, the meta-learning approach selects the latter classifier and uses it for incoming data. The advantage of this step is apparent on both global system and classifier's error rate plots. To compare the relative performance over time of two classifiers, we use the $Q_i = \log(A_i/B_i)$ statistic [7], where the index i refers to the time-stamp, and A and B refers to the error rate of the classifiers under comparison. We plot the logarithm of the ratio between the two error rates. If both classifiers exhibit the same performance, $Q = 0$, positive values for Q mean that the error rate of classifier B is lower than the error rate of classifier A, while negative values refer to the opposite. Figure 7 clearly illustrates the advantage of reusing the previous model when the occurrence of a drift.

This example clear illustrates the main advantages of the proposed framework. On the one hand, it provides useful information about the unknown dynamics of data: the recurrence of concepts. On the other hand, the reaction of the system in the presence of drift is much faster than learning a new model. We should point out that learning a new model from scratch and

Table 2 Description of *emailing list* dataset

Concept	1	2	3	4	5
Medicine	+	−	+	−	+
Space	−	+	−	+	−
Baseball	−	+	−	+	−
Start time	1	242	484	727	968

The columns refer to the sequence of concepts presented in the data stream. The user is interested in the topics marked with (+) and not interested in the topics marked with (−)

reusing a previously learned model will converge to similar performance after processing significant number of examples.

4.2 The Emailing list problem

[12] used *Emailing list* dataset generated from 20 Newsgroups. It simulates a stream of email messages and a user labels them as interesting or junk depending on his/her interests. Similarly, the topics we used are the following: science/medicine, science/space, and recreation/sports/baseball. After preprocessing (conversion to lowercase, removing punctuation and numbers, stemming, and keeping only words that appeared 10 times or more), the dataset consists of 1,209 examples with 317 Boolean attributes (bag-of-words representation). We split the stream into five time periods. In the first concept, the user is only interested in messages of the topic medicine. At the end of each period, the user's interest in a topic changes thus presenting a concept drift. Table 2 shows which messages are considered interesting (+) or junk (−) in each period.

The *Emailing list* dataset presents two alternating types of concepts. The first is repeated three times, and the second is repeated twice. The system recognizes the recurrence of concepts and learns only two $layer_1$ classifiers—one for each concept type when it first appears. Later on, when the concepts change, the system correctly reuses those learned models for the respective recurring concepts. It again noticeably improves the performance in the early stages after drift for each of the recurring concepts, depicted in Fig. 8 (top), and the global error rate of the system, which can be observed in Fig. 8 (bottom).

4.3 The Usenet problem

The *Usenet* dataset is also a text dataset, which was inspired by Katakis et al. [11]. This is a simulation of news filtering with a concept drift related to the change of interest of a user over time. For this purpose, we use the data from 20 Newsgroups and handle it as follows. There are six topics chosen. For each concept, the simulated user subscribes to the mailing list four of them being interested only in two. Over time the virtual user decides to unsubscribe those he was not interested in, and subscribe two new ones. The previously interesting topics become out of his main interest. Table 3 summarizes the concepts. The topics of interest are repeated to simulate recurring concepts. The original dataset is divided into training data and test data. The training data appears in the first three concepts, whereas test data is in the last three (recurring) concepts. The dataset has 659 attributes and 5,931 examples.

The *Usenet* dataset includes previously known recurrence, where the lengths of reappearing concepts are shorter than those of the first occurrence. The performance is depicted in Fig. 9. The decision to reuse an older model is significant enough only for the last concept

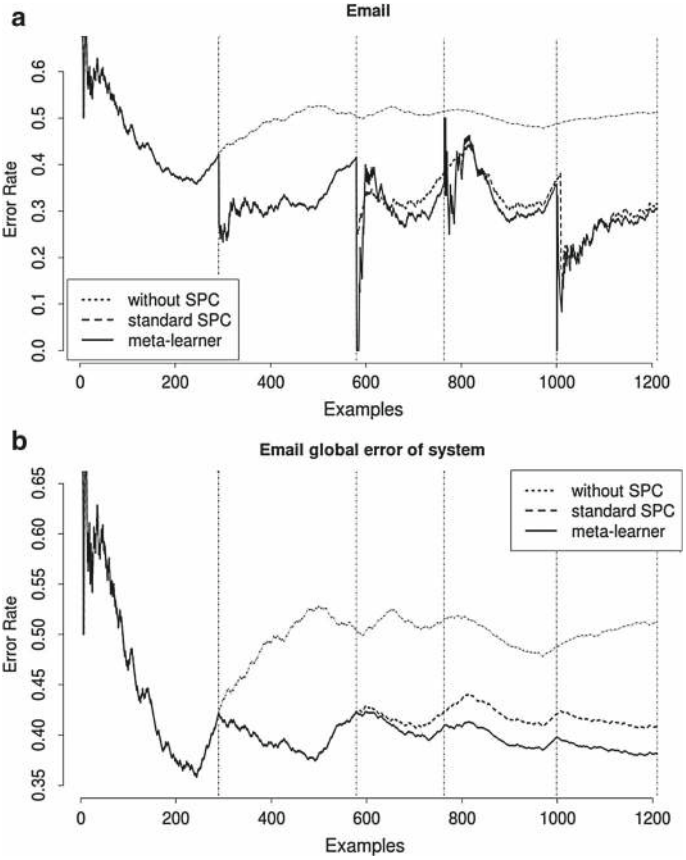

Fig. 8 The evolution of the error rate of each model (*top*) and the global error of the system (*bottom*) in *Emailing list* dataset

Table 3 Description of Usenet concepts

Concept	1	2	3	4	5	6
Electronics	Yes	No	–	Yes	No	–
Motorcycles	No	–	Yes	No	–	Yes
Crypt	Yes	No	–	Yes	No	–
Space	No	–	Yes	No	–	Yes
Hockey	–	Yes	No	–	Yes	No
Forsale	–	Yes	No	–	Yes	No
Start time	1	1,192	2,377	3,562	4,354	5,143

The first column refers to the topics available. The other columns refer to the sequence of concepts presented in the data stream. The user subscribes only the topics marked with *yes* (interested in), and the topics marked with *no* (not interested in)

where model 3 is selected. The effect of this correct decision is that besides the information provided by the identification of context that reoccurs, which itself might be useful, the over-

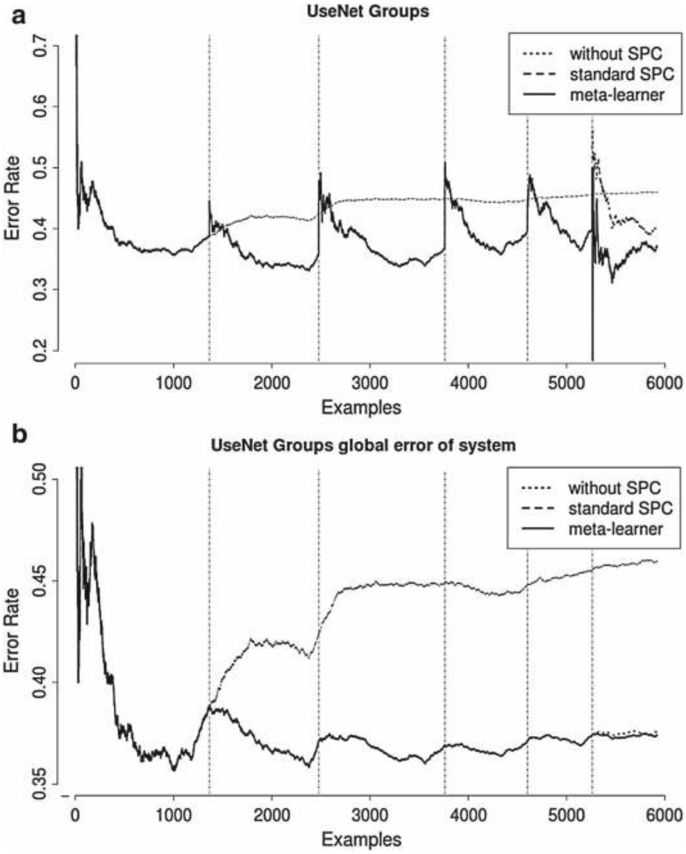

Fig. 9 The evolution of the error rate of each model (*top*) and the global error of the system (*bottom*) in *Usenet* dataset

all performance also improves. The differences might not appear very large, but they are not expected to be. It is expected that with enough samples the classifiers converge to the same error rate. However, the proposed system can improve predictive capabilities in the early stages after drift, and, more importantly, it provides additional information about similarity of previous concepts if they appear.

4.4 The *Spam Detection* Problem

We use the *Spam Detection* dataset, a real-world text data stream [11] that is chronologically ordered to represent the evolution of Spam messages over time. This dataset consists of 9,324 examples with 850 informative attributes. There are two classes that represents the two types of messages: legitimate and Spam messages. The Spam messages constitute around 20 % of the dataset. Drifts and recurrence of concepts are not known in this dataset.

The results are depicted in Fig. 10. The second drift occurred at the time when label of example 652 arrived. The system reused first model, which improved the predictions, but the next change was detected after only 52 (56 for `standard SPC`). After this detection, the system reused the second model. It did not improve performance in the very beginning

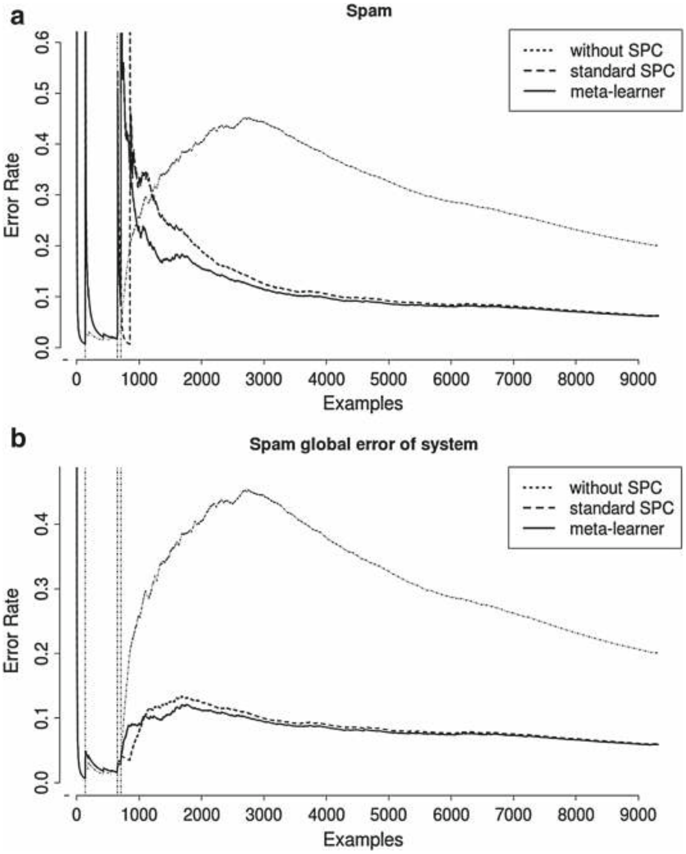

Fig. 10 The evolution of the error rate of each model in *Spam Detection* dataset. The error rates of three approaches are compared–learning without drift detection, learning with `SPC`, and using meta-learners to detect recurring concepts

as previously; nevertheless, it quickly started to decrease the error. Note that the length of a concept might be shorter than the delay and therefore the decision to reuse a model might be affected by instances from the next one. `Standard SPC` had better predictions for the examples incoming soon after drift, but quickly after that another change, which did not occur in `meta-learner`, was detected.

5 Conclusions

In this paper, we discuss stream mining problems where contexts change over time and might reoccur. We present a generic framework that identifies context using drift detection, characterizes contexts using meta-learning, and selects the most appropriate base model for the incoming data using unlabeled examples. The proposed framework is based on a meta-learning schema which aims to recognize the area of applicability of the base classifier. This can be useful in scenarios where there are recurrent concepts and the knowledge of the meta-learners can be later used for selecting previously learned models to be reused when a change in the concept is indicated. The meta-learners detect recurrence of contexts using labeled and unlabeled examples. The experimental evaluation on two text mining problems point out the main advantages of the proposed system: it provides information about the recurrence of concepts and fast adaptation models after drift.

Evolving data requires that learning algorithms must be able to monitor the learning process and the ability of predictive self-diagnosis. A significant and useful characteristic is diagnosis—not only after failure has occurred, but also predictive (before failure). These aspects require monitoring the evolution of the learning process, taking into account the available resources, and the ability of reasoning and learning about it.

Acknowledgments This work is part-funded by the ERDF—European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness), by the Portuguese Funds through the FCT (Portuguese Foundation for Science and Technology) within project FCOMP—01-0124-FEDER-022701. The authors acknowledge the financial support given by the project *Knowledge Discovery from Ubiquitous Data Streams* (PTDC/EIA/098355/2008), funded by FCT. Petr Kosina acknowledges the support of Masaryk University, Faculty of Informatics.

References

1. Baena-Garcia M, Campo-Avila 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 (ECML-PKDD), Berlin, Germany
2. Bifet A, Gavaldà R (2007) Learning from time-changing data with adaptive windowing. In: Proceedings of the SIAM international conference on data mining, Minneapolis, USA. SIAM, pp 443–448
3. Dijkstra W (1974) Self-stabilizing systems in spite of distributed control. Commun ACM 17(11):643–644
4. Duda RO, Hart PE (1973) Pattern classification and scene analysis, vol 95. Wiley, New York
5. Gama J (2010) Knowledge discovery from data streams. CRC Press, Boca Raton
6. Gama J, Medas P, Castillo G, Rodrigues P (2004) Learning with drift detection. In: SBIA Brazilian symposium on artificial intelligence. Springer, Berlin, pp 286–295
7. Gama J, Sebastiao R, Rodrigues PP (2013) On evaluation stream learning algorithms. Mach Learn 90(3):317–346
8. Granitzer M, Kröll M, Seifert C, Rath AS, Weber N, Dietzel O, Lindstaedt SN (2008) Analysis of machine learning techniques for context extraction. In: Pichappan P, Abraham A (eds) ICDIM. IEEE, pp 233–240
9. Grant E, Leavenworth R (1996) Statistical quality control. McGraw-Hill, London
10. Harries MB, Sammut C, Horn K (1998) Extracting hidden context. Mach Learn 32:101–126
11. Katakis I, Tsoumakas G, Banos E, Bassiliades N, Vlahavas I (2009) An adaptive personalized news dissemination system. J Intell Inf Syst 32:191–212
12. Katakis I, Tsoumakas G, Vlahavas I (2010) Tracking recurring contexts using ensemble classifiers: an application to email filtering. Knowl Inf Syst 22:371–391
13. Klinkenberg R (2004) Learning drifting concepts: example selection vs. example weighting. Intell Data Anal 8(3):281–300
14. Lazarescu MM (2005) A multi-resolution learning approach to tracking concept drift and recurrent concepts. In: Proceedings of the 5th international workshop on pattern recognition in information systems
15. Ortega J (1995) Exploiting multiple existing models and learning algorithms. In: AAAI 96—workshop in induction of multiple learning models, pp 17–21
16. Ortega J, Koppel M, Argamon S (2001) Arbitrating among competing classifiers using learned referees. Knowl Inf Syst 3(4):470–490
17. Ramamurthy S, Bhatnagar R (2007) Tracking recurrent concept drift in streaming data using ensemble classifiers. In: Proceedings of the sixth international conference on machine learning and applications (ICMLA ’07), pp 404–409. IEEE Computer Society, Washington, DC
18. Seewald A, Fürnkranz J (2001) An evaluation of grading classifiers. In: Hoffmann F, Hand DJ, Adams N, Fisher D, Guimaraes G (eds) Advances in intelligent data analysis: proceedings of the 4th international conference (IDA-01), Cascais, Portugal. Springer, pp 115–124
19. Street WN, Kim Y (2001) A streaming ensemble algorithm (sea) for large-scale classification. In: SIGKDD, Knowledge discovery and data mining. ACM Press, New York, pp 377–382

20. Turney P (1996) The management of context-sensitive features: a review of strategies. In: 13th international conference on machine learning (ICML96), workshop on learning in context-sensitive domains, Bari, Italy, pp 60–66
21. Widmer G (1997) Tracking context changes through meta-learning. Mach Learn 27(3):259–286
22. Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. Mach Learn 23(1):69–101
23. Yang Y, Wu X, Zhu X (2006) Mining in anticipation for concept change: proactive–reactive prediction in data streams. Data Min Knowl Discov 13(3):261–289

Author Biographies

João Gama Associate Professor at Faculty of Economics, University of Porto and researcher at LIAAD/INESC TEC, U. of Porto. João Gama works in Machine Learning and Data Mining. His main research interest is in mining data streams, evolving data and change detection. He is author of a recent book on Knowledge Discovery from Data Streams.

Petr Kosina received his bachelor degree from Masaryk University, Brno, Czech Republic, in 2008 and a master degree, also from Masaryk University, in 2009. He is currently a Ph.D. student at the Faculty of Informatics at Masaryk University, Brno, Czech Republic, and does his research under KDUS project at LIAAD-INESC TEC in Porto, Portugal. His research interests include stream mining, change detection, and meta-learning