Prototype-based Learning on Concept-drifting Data Streams

Junming Sha[1,2], Zahra Ahmadi[1], Stefan Kramer[1]
[1]University of Mainz, 55128 Mainz, Germany
[2]University of Electronic Science and Technology of China , 611731 Chengdu, China
{junmshao, zaahmadi}@uni-mainz.de, kramer@informatik.uni-mainz.de

ABSTRACT

Data stream mining has gained growing attentions due to its wide emerging applications such as target marketing, email filtering and network intrusion detection.In this paper, we propose a prototype-based classification model for evolving data streams, called SyncStream, which dynamically models time-changing concepts and makes predictions in a local fashion. Instead of learning a single model on a sliding window or ensemble learning, SyncStream captures evolving concepts by dynamically maintaining a set of prototypes in a new data structure called the P-tree. The prototypes are obtained by error-driven representativeness learning and synchronization-inspired constrained clustering. To identify abrupt concept drift in data streams, PCA and statistics based heuristic approaches are employed. SyncStream has several attractive benefits: (a) It is capable of dynamically modeling evolving concepts from even a small set of prototypes and is robust against noisy examples. (b) Owing to synchronization-based constrained clustering and the P-Tree, it supports an efficient and effective data representation and maintenance. (c) Gradual and abrupt concept drift can be effectively detected. Empirical results shows that our method achieves good predictive performance compared to state-of-the-art algorithms and that it requires much less time than another instance-based stream mining algorithm.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications - Data Mining

Keywords

Data stream; Concept drift; Classification; Synchronization

1. INTRODUCTION

Data stream classification is a challenging data mining task because of two important properties: potentially infinite length and evolving nature. Currently, there are two main strategies to mine evolving data streams: single model learning and ensemble learning. Single model learning tries to learn and update a classification model by training on a fixed or adaptive window of recent incoming examples [9], [12]. The prediction performance of this type of learning usually suffers in the presence of concept drift. By contrast, ensemble learning partitions continuous data streams into smaller chunks and uses them to train a number of base classifiers to capture evolving concepts [16], [2]. Ensembles can scale up to large volumes of streaming data and adapt quickly to new concepts. However, two existing limitations of ensemble learning are considered only little. First, the performance of ensemble learning depends on the accuracy of each base classifier capturing the concept, where each base classifier is actually used in a black-box fashion. For illustration, let us take Fig. 1 as an example. If the size of data chunk is small (data arrived during $[t0, t1)$), the concept of the data stream for each base classifier cannot be sufficiently learned. On the other hand, if the size of a data chunk is too large (data arrived during $[t0, t2)$), base classifiers may learn more than one concept. Second, it is difficult to decide whether more recent or older examples are more relevant (e.g., the relatively old example P_1 is important for predicting the class label of P_2 in Fig. 1 (a)). Learning from recent examples, as in most established algorithms, is not always optimal. A more natural way is to dynamically select short-term and/or long-term representative examples to capture the trend of time-changing concepts, and use them to predict the test data (cf. Fig. 1 (b)).

In this paper, we propose a new prototype-based learning method to mine evolving data streams. To dynamically identify prototypical examples, error-driven representativeness learning is used to compute the representativeness of examples over time. Noisy examples with low representativeness are discarded, and representative examples can

KDD'14, August 24–27, 2014, New York, NY, USA.
Copyright 2014 ACM 978-1-4503-2956-9/14/08 ...$15.00.
http://dx.doi.org/10.1145/2623330.2623609.

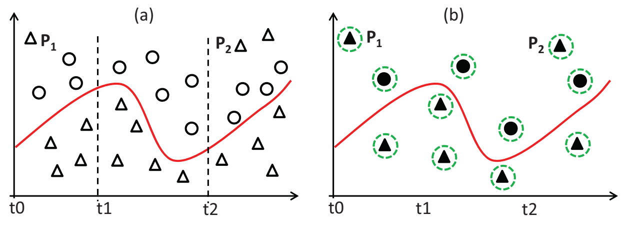


Figure 1: How to select suitable training examples for learning a classification model on an evolving data stream?

be further summarized by a smaller set of prototypes using synchronization-inspired constrained clustering. To dynamically represent the online training data, a new data structure, the P-Tree, is proposed to update the set of prototypes, to capture time-changing concepts and to support accurate predictions.

Contributions

Building upon the data representation of synchronization-inspired constrained clustering, SyncStream has several attractive benefits, most importantly:

- **Prototype-based Data Representation:** Due to the properties of synchronization-inspired constrained clustering, examples in data streams can be well summarized and represented as a set of prototypes, which allows preserving the data structure as well as the class distribution of the original examples.
- **Concept Modeling:** Instead of using a set of base classifiers or learning on a window of recent data, SyncStream provides an intuitive way to model evolving concepts by dynamically maintaining a small set of short-term and/or long-term prototypes based on *error-driven representativeness learning* (see Section 3.4.1).
- **High Performance:** Thanks to the efficient and effective online data maintenance relying on the proposed *P-Tree* data structure (see Section 3.4.2), SyncStream allows adapting to time-changing concepts quickly to make accurate predictions.

The remainder of this paper is organized as follows: In the following section, we briefly survey related work. Section 3 presents our algorithm in detail. Section 4 contains an extensive experimental evaluation. We finally conclude the paper in Section 5.

2. RELATED WORK

Data stream classification has received much attention in recent decades. Early algorithms focused on learning a single model from evolving data (e.g., [9]). But very soon they were replaced by ensemble models in order to obtain more accurate predictions. Ensemble models (e.g., [16, 2]) are of great interest, as each model can represent one possible distribution over the data, therefore current data can be generated from a mixture of distributions with changing weights over time. Also, if one of the previous concepts reappears, they can handle this case more effectively [8]. However, as explained in Section 1, ensemble models have some shortcomings in determining a suitable window size and useful instances. One efficient solution to address these shortcomings is the use of instance-based learning, which has been considered in some papers in recent years [10, 17, 13]. In instance-based learning, instead of taking time to build a global model, the target function is approximated locally by means of selected instances. The inherent incremental nature of instance-based learning algorithms and their simple and flexible adaptation mechanism makes this type of algorithm suitable for learning in complex dynamic environments. Nevertheless, instance-based learning algorithms have to face their own challenges, such as their space requirements and their sensitivity to noise. To manage memory consumption and accelerate nearest neighbor lookup, researchers usually use a tree based data structure such as the M-Tree [17, 13]. In this tree, instances of small neighborhoods are kept in the leaves; inner nodes combine their subnodes to form bigger spheres, and the root encompasses all the data. The number of subnodes is determined heuristically. In contrast, we provide an intuitive and effective strategy to summarize data, and dynamically maintain a small set of prototypes in the proposed P-Tree to support effective and efficient online data maintenance.

To handle concept drift in an evolving data stream, one strategy is to dynamically monitor the change of a distribution over time and set a flag whenever drift is detected. This type of method is referred to as explicit concept drift detection. In contrast to this strategy, implicit methods adapt to new concepts by updating a model continuously. The benefit of using explicit methods is the ability to react to sudden drift more quickly, while implicit methods are more suitable for data streams with gradual concept drift. Generally, methods for concept drift detection are categorised into statistical techniques (e.g., [5]) and those based on performance monitoring between two consecutive data chunks (e.g., [1]). If concept drift is detected, the model may be reconstructed from scratch (e.g., [7]) or only the affected parts of the model are updated (e.g., [9]). Locally updating a model has the benefit of maintaining useful knowledge from previous data and is especially useful in recognizing recurring concepts. In this paper, error-driven representativeness learning is introduced to identify the most important prototypes to capture the trend of time-changing concepts in an implicit and local way, while sudden concept drift is handled based on two heuristic strategies.

3. PROPOSED METHOD

In this section, we present the SyncStream algorithm for mining evolving data streams. Before that, we provide an overview of the algorithm and the intuition behind it.

3.1 Overview

As introduced in Section 1, the basic idea of our algorithm is to dynamically keep track of short-term and/or long-term representative examples to capture the trend of time-changing concepts, and use them for prediction in a local instance-based fashion. Specifically, a small set of data examples is first regarded as initial prototypes and inserted into a proposed hierarchical tree structure called the P-Tree, which is used to maintain prototypes and drifting concepts on two levels, respectively. For each new example, the nearest prototype in the P-Tree is used to predict its label. Meanwhile, during the classification process, an error-driven learning approach is employed to determine the representativeness of prototypes in the P-Tree. If the nearest prototype correctly predicts the new example, this indicates that the prototype is in line with the trend of the current concept of the new example. The representativeness of a prototype is thus increased (and decreased in the opposite case). However, due to restricted memory, we cannot store all prototypes in the P-Tree. To fit the P-Tree in an any-space memory framework and support efficient prediction, inappropriate or noisy examples with low representativeness are discarded. The representative examples are further summarized as a smaller set of prototypes based on synchronization-inspired constrained clustering, where each prototype solely belongs to one specific class (cf. Section 3.2). Relying on constrained clustering, historical data is summarized and, more importantly, the data structure of the original examples is well preserved. To further handle the evolving nature of data streams, PCA and statistics based methods are introduced to detect sudden concept drift, so that prototypes in the P-Tree can be updated quickly to capture a new concept. Once a new concept is emerging, the previous concept is modeled as a set of prototypes and stored on the concept level of the P-Tree. In the following, we will first elaborate on how to summarize historical data based on synchronization-inspired constrained clustering, before we move on to drift detection and the P-tree.

3.2 Prototype-based Data Representation

In the context of an evolving data stream, it is infeasible to keep track of the entire history of the data. To support effective and efficient online data maintenance, we first introduce a data summarization technique by adapting synchronization-inspired clustering [14], [3]. The key philosophy of synchronization-based clustering approaches (e.g., *Sync* [3] or *pSync* [14]) is to regard each example as a phase oscillator and simulate the dynamical behavior of the examples over time. Based on the non-linear local interactions among examples, similar examples will finally synchronize together and end up in the same common phase. The synchronized common phase represents the set of similar examples, and thus provides an intuitive way to summarize the data.

Typically, a synchronization-based clustering algorithm needs three definitions to simulate a dynamic clustering process. In the following, we give a short summary of all necessary definitions.

Definition 1 (**ϵ-Range Neighborhood**)

Given a $\epsilon \in \mathcal{R}$ and $x \in \mathcal{D}$, the ϵ-range neighborhood of an example x, denoted by $N_\epsilon(x)$, is defined as:

$$N_\epsilon(x) = \{y \in \mathcal{D} | dist(y, x) \leq \epsilon\}, \quad (1)$$

where $dist(y, x)$ is a metric distance function, and the Euclidean distance is used in this study.

Definition 2 (**Interaction Model**)

Let $x \in \mathcal{R}^d$ be an example in the data set $\mathcal{D}$ and x_i be the i-th dimension of the example x, respectively. Each example x is viewed as a phase oscillator. With an ϵ-range neighborhood interaction, the dynamics of each dimension x_i of the example x is defined by:

$$x_i(t+1) = x_i(t) + \frac{1}{|N_\epsilon(x(t))|} \cdot \sum_{y \in N_\epsilon(x(t))} \sin(y_i(t) - x_i(t)), \quad (2)$$

where $sin(\cdot)$ is the coupling function. $x_i(t+1)$ describes the renewal phase value of the i-th dimension of example x at $t = (0, \ldots, T)$ during the dynamic clustering.

To characterize the level of synchronization between oscillators during the synchronization process, a cluster order parameter R_c is defined to measure the coherence of the local oscillator population.

Definition 3 (**Cluster Order Parameter**)

The cluster order parameter R_c is used to terminate the dynamic clustering by investigating the degree of local synchronization, which is defined as:

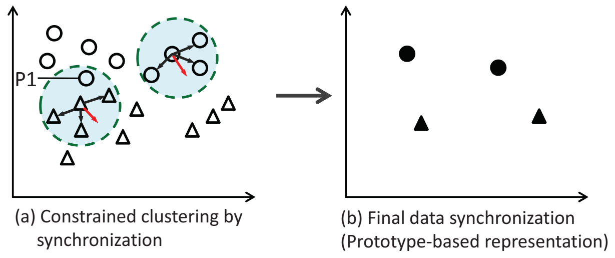

P1
(a) Constrained clustering by synchronization
(b) Final data synchronization (Prototype-based representation)

Figure 2: Illustration of synchronization-inspired constrained clustering.

$$R_c = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{|N_\epsilon(x)|} \sum_{y \in N_\epsilon(x)} \mathbf{e}^{-||y-x||}. \quad (3)$$

The dynamic clustering terminates when $R_c(t)$ converges, which indicates local phase coherence. At this moment, cluster examples have the same phase (common location in the feature vector space).

One of the most salient features of synchronization-based clustering is its dynamic property. During the process of clustering, each object moves in a non-linear way driven by the local data structure, and finally a set of examples with similar local structure will group together. The synchronized phase thus provides a natural way to summarize the local data. In contrast to computing the center of cluster examples or other summarized statistics, the synchronized phases represent the original data without losing the local structure of the data.

However, for data stream classification, clustering-based data summarization needs to be conducted in a supervised manner, i.e. it needs to take into account class information. Therefore, the interaction model needs to be extended to make sure the examples in a cluster solely belong to the same class. To achieve this goal, there are two strategies: active or passive. The active strategy is to exert the opposite interaction on two examples with different class labels. Although this strategy can push the examples with different class labels away from each other, the local structure cannot be maintained. The passive strategy is to impose a cannot-link constraint [15], where two examples with different class labels cannot interact with each other. Without interaction, the examples with distinct class labels cannot synchronize together. Thereby, we reformulate the interaction model using the cannot-link constraint as follows.

Definition 4 (**Interaction Model with Constraint**)

Let $x \in \mathcal{R}^d$ be an example in the data set $\mathcal{D}$ and x_i be the i-th dimension of the example x, respectively. $l_x \in L$ is the class label of example x. With an ϵ-range neighborhood interaction, the dynamics of each dimension x_i of the example x is defined by:

$$x_i(t+1) = x_i(t) + \frac{1}{|N_\epsilon(x(t))|} \cdot \quad (4)$$

$$\sum_{y \in N_\epsilon(x(t)), eq(l_x, l_y)} \sin(y_i(t) - x_i(t)),$$

where l_x and l_y indicate the class labels of examples x and y, respectively. $eq(\cdot, \cdot)$ is the function to check whether two class labels are equal.

Based on the extended interaction model, Fig. 2 displays two snapshots of the simulated dynamical example movement for data synchronization. Specifically, there are two classes of examples with different shapes. Given an interaction range (Fig. 2 (a)), examples of the same class will interact together, while examples of different classes will not (e.g., example $P1$). Finally, due to the non-linear interaction driven by local structure, similar examples of the same label synchronize together (Fig. 2 (b)). In this paper, we define the synchronized phases as prototypes. Due to the benefits of synchronization-based constrained clustering, we obtain two desirable properties of a prototype-based data representation:

- **Data Structure Preservation**. Since a prototype is defined as a synchronized phase, it provides an intuitive way to summarize the set of synchronized examples and further envelops the local cluster structure (both data structure and class distribution are represented).
- **Multi-scale Representation**. As a prototype can be viewed as a regular example, it supports a multi-scale representation, which means a set of prototypes can be further summarized by clustering.

In Section 3.4, we will demonstrate that the two properties are useful for online data maintenance.

3.3 Concept Drift Detection

After the prototype-based data representation, we introduce two heuristic strategies to identify abruptly drifting concepts, so that the classification model allows quickly learning and adapting to emerging new concepts. Instead of only investigating the change of the data distribution (for details, see the review paper by Gama *et al.* [6]), we exactly examine the change of the target concepts (i.e., the class distributions).

Formally, let D_t and D_{t+1} be two given sequential data chunks in a data stream D, and $L = \{l_1, l_2, \cdots, l_{|L|}\}$ be the set of labels. The objective is to analyze whether or not the statistical properties of each target concept (class distribution) between D_t and D_{t+1} have changed over time.

3.3.1 Principal Component Analysis (PCA)

To identify concept drift, the first proposal is to analyze the change of each class distribution using principal component analysis (PCA):

- For each class label $l \in L$, the examples associated with class label l in two data sets (D_t and D_{t+1}) are collected, respectively, denoted as D_t^l and D_{t+1}^l (e.g., D_t^1 and D_{t+1}^1 in Fig. 3), if existing.
- PCA is used to decompose the covariance matrix Σ of examples of each individual collected data chunk, which is denoted by $\Sigma = VEV^T$. The orthogonal matrix V is called eigenvector matrix and the diagonal matrix E is eigenvalue matrix. The eigenvectors represent the principal directions of the data set, and the eigenvalues represent the variance along these directions. We sort the eigenvectors according to the corresponding eigenvalues in descending order. Finally, the major direction of the data set is obtained (e.g., $V_{D_t}^1(1)$ is the major direction of D_t^1 for class **1**).

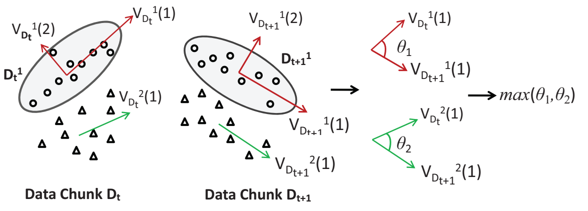


Figure 3: PCA-based concept drift analysis.

- We analyze the concept drift between two data sets (D_t and D_{t+1}) by investigating the variances of all class distributions. Specifically, let $V_{D_t}^l(1)$ and $V_{D_{t+1}}^l(1)$ be the major eigenvectors for class label l on the data set D_t and D_{t+1}, respectively, the degree of concept drift is finally defined as:

$$Angle(D_t, D_{t+1}) = \max_{l \in L} acos(V_{D_t}^l(1) \cdot V_{D_{t+1}}^l(1)), \quad (5)$$

where $\cdot$ denotes the inner product. For illustration, Fig. 3 describes the procedure of PCA-based concept drift analysis.

3.3.2 Statistical Analysis

The second strategy for concept drift detection is based on statistical analysis. The basic idea is to compute a suitable statistic, which is sensitive to class distribution changes between two sets of examples. The degree of concept drift is measured by the resulting p-value. Although statistical analysis is a straightforward solution to concept drift detection, computing statistics on multivariate data is usually complicated and time consuming. Here, we extend Brunner and Munzel's generalized Wilcoxon test statistic [4], [11] to compare the differences of class distributions of two data sets. In contrast to most existing statistics (e.g., Welch test, Mann-Whitney U test), this non-parametric statistical test allows for relaxation of the assumption of equality of variance and normal distributions simultaneously. More importantly, it allows an efficient computation. Formally, Brunner and Munzel's generalized Wilcoxon test statistic is calculated as follows.

Given each data pair (D_t^l and D_{t+1}^l), we investigate whether the sets of examples corresponding to class label l in the data sets D_t and D_{t+1} are significantly different. For this purpose, we define the intra- and inter-rank on both data sets as follows. Let $\Gamma^l = D_t^l \bigcup D_{t+1}^l$, $R_{D_t^l}^{ij}$, $R_{D_{t+1}^l}^{ij}$ and $R_{\Gamma^l}^{ij}$ are defined as the rank of examples $x \in D_t^l$, $y \in D_{t+1}^l$ and $z \in \Gamma^l$ on the j^{th} dimension, respectively. In addition, we further calculate $\overline{R_{D_t^l}^j}$ and $\overline{R_{D_{t+1}^l}^j}$, which are the j^{th} dimensional mean ranks of examples from D_t^l and D_{t+1}^l in the data set Γ^l. Formally, we compute the estimated variance σ_{BF}^2 as follows:

$$\sigma_{BF}^2 = \frac{(|D_t^l| + |D_{t+1}^l|)\nu_{D_t^l}^2}{|D_{t+1}|} + \frac{(|D_t^l| + |D_{t+1}^l|)\nu_{D_{t+1}^l}^2}{|D_t^l|}, \quad (6)$$

where

$$\nu_{D_t^l}^2 = \frac{1}{|D_t^l| - 1} \sum_{j=1}^{d} \sum_{i=1}^{|D_t^l|} (R_{\Gamma^l}^{ij} - R_{D_t^l}^{ij} - \overline{R_{D_t^l}^j} + \frac{|D_t^l| + 1}{2})^2, \quad (7)$$

$$\nu^2_{D^l_{t+1}} = \frac{1}{|D^l_{t+1}|-1} \sum_{j=1}^{d} \sum_{i=1}^{|D^l_{t+1}|} (R^{ij}_{\Gamma^l} - R^{ij}_{D^l_{t+1}} - \overline{R^j_{D^l_{t+1}}} + \frac{|D^l_{t+1}|+1}{2})^2. \tag{8}$$

Finally, the adapted Brunner and Munzel's generalized Wilcoxon test statistic, W^l_{BF}, is defined as:

$$W^l_{BF} = \sqrt{\frac{|D^l_t||D^l_{t+1}|}{|D^l_t|+|D^l_{t+1}|}} \cdot \frac{\sum_{j=1}^{d}(\overline{R^j_{D^l_t}} - \overline{R^j_{D^l_{t+1}}})}{\sigma_{BF}} \tag{9}$$

If both numbers of examples are large enough (>20), then the statistic W^l_{BF} is asymptotically standard normal. The test statistic is compared with a normal distribution of zero mean and unit standard deviation to obtain a p-value. The minimal value of all resulting p-values on the data pairs is used to measure the degree of concept drift between the two data chunks D_t and D_{t+1}.

3.4 Online Data Maintenance

In this section we will first demonstrate how to learn the representativeness of prototypes over time. Subsequently, a hierarchical tree structure, called P-Tree, is proposed to dynamically maintain the set of prototypes and evolving concepts building upon the prototype-based data representation and concept drift detection as introduced above.

3.4.1 Error-driven representativeness learning

In an evolving data stream, either short-term or long-term historical examples may be important for prediction. Traditional single model learning or ensemble learning approaches usually capture the current data concept by operating on a sliding window of data or a set of data chunks. However, how to decide the size of a window or horizon of the training data is a non-trivial task. In addition, for both traditional single model learning or ensemble learning, usually, not all examples in the window or in a data chunk are relevant for classification. Therefore, in this paper, we propose an error-driven approach to automatically learn the representativeness of data and dynamically identify the short-term or long-term prototypical examples to capture the data concepts.

The basic idea is to leverage the prediction performance of incoming examples to infer the representativeness of examples in the training data. Formally, given a new example x, let $\mathcal{C}$ be the current training data set (a set of prototypes), and $y \in \mathcal{C}$ the one prototype nearest to x. x_l and x_{pl} are the true label and the predicted label of x by y, respectively. The representativeness of the prototype y is updated as follows:

$$Rep(y) := Rep(y) + Sign(x_{pl}, x_l), \tag{10}$$

where the initial value of $Rep(y)$ is one and $Sign(x, y)$ is the sign function, and 1 if x equals y, -1 otherwise.

3.4.2 P-Tree

To support efficient and effective online data maintenance, we propose a tree structure called the P-Tree. The P-Tree is a hierarchical tree structure consisting of two levels, where the first level stores a set of time-changing prototypes capturing the current concept, and the second level stores drifting concepts over time (see Fig. 4). Except for the inclusion of new, incoming examples continuously, the P-Tree is also required to be updated when the following two scenarios occur: (a) *maximum boundary*, or (b) *concept drift*.

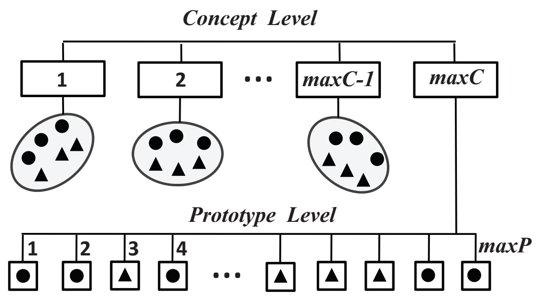


Figure 4: An illustration of the P-Tree structure.

Maximum boundary limits the number of prototypes and the number of concepts that can be stored in the P-Tree. This restriction allows us to perform data stream classification on any computer at hand in an any-space memory framework. The updating procedure is as follows: For each incoming example, if the size of the prototypes in the P-Tree does not exceed the maximum boundary (i.e., $maxP$), it is directly inserted into the entry on the first level of the P-Tree, and meanwhile the representativeness of existing prototypes in the P-Tree is computed based on error-driven learning (cf. Section 3.4.1). Once the size exceeds $maxP$, prototypes with different representativeness are handled separately. (a) Prototypes with low representativeness (e.g., a negative value) in the P-Tree are directly removed. (b) Prototypes with high representativeness (e.g., a positive value) in the P-Tree are kept, which indicates these prototypes can capture the current concept well. (c) Prototypes with unchanged representativeness are further summarized by a new and smaller set of prototypes based on synchronization-inspired constrained clustering (cf. Section 3.2). Due to the two desirable properties of prototype-based data representation (cf. Section 3.2), the newly generated prototypes preserve the original data structure well and can be summarized further on a higher level of abstraction. Moreover, to maintain the time-changing concepts on the second level, once the number of concepts is greater than $maxC$, the oldest concept is removed.

The second scenario occurs when a new concept is emerging, which can be detected by PCA or statistical analysis (cf. Section 3.3). In this case, we draw sample data from all prototypes representing a previous concept, and perform synchronization-inspired constrained clustering on it. The resulting set of new prototypes is inserted into one entry of the concept level of the P-Tree. Meanwhile, all prototypes representing the previous concept are removed from the first level of the P-Tree. Moreover, if there is no significant concept change over time (i.e., gradual concept drift or no concept drift), we also extract the concept based on the prototypes in the P-Tree by clustering after a specified time T; yet the prototypes are not removed. This strategy helps to give a summarization of data over time on the concept level.

3.5 SyncStream Algorithm

Building upon the proposed online data maintenance, finally instance-based learning in the form of the nearest neighbor classifier is used for data stream classification. In contrast to eager learning approaches, SyncStream only needs

Algorithm 1 SyncStream

```
Input: $D$, $D_{init}$, $maxP$, $maxC$, $chunkSize$, $type$, $T$

P-Tree.insert($D_{init}$); //P-Tree initialization
while NoError do
  for each new example $x \in D$ do
    //error-driven representativeness learning
    $y := NN$(P-Tree, $x$); $x.prelabel := y.label$;
    if $x.prelabel == x.label$ then
      $y.Rep++$;
    else
      $y.Rep--$;
    end if
    $measure := computAccuracy(x.prelabel)$;
    P-Tree.insert($x$);

    //maximum boundary in prototype level
    if P-Tree.prototypeSize $== maxP$ then
      $S_n :=$ P-Tree.getNegativePrototypes();
      P-Tree.remove($S_n$);
      $S_u :=$ P-Tree.getUnchangedPrototypes();
      $P := ConstrainedSync(S_u)$;
      P-Tree.remove($S_u$); P-Tree.insert($P$);
    end if

    //maximum boundary in concept level
    if P-Tree.conceptSize $== maxC$ then
      P-Tree.removeOldestConcept();
    end if

    //concept drift detection
    if $numObj == chunkSize$ then
      if $type == PCA$ then
        $\theta := angle$(curChunk, preChunk);
      else
        $p := p\text{-}value$(curChunk, preChunk);
      end if
      if $\theta \geq 60°$ or $p \leq 0.01$ or $numInstance \geq T$ then
        $P_s := Sample$(P-Tree.prototypes);
        $C_s := ConstrainedSync(P_s)$;
        P-Tree.addConcept($C_s$);
        if $\theta \geq 60°$ or $p \leq 0.01$ then
          P-Tree.prototypes.clear();
        end if
      end if
    end if
  end for
end while

Output: P-Tree, $measure$;
```

to maintain the online training data (represented by a small set of time-changing prototypes) in the P-Tree. Finally, the pseudocode of SyncStream is shown in Algorithm 1.

4. EXPERIMENTAL EVALUATION

In this section, we evaluate our proposed algorithm SyncStream on both synthetic and real-world data streams.

4.1 Experiment Setup

Data sets. We evaluate the proposed method on synthetic data and four different types of real-world data: Spam, Electricity, Covtype, and Sensor. Table 1 lists statistics of the four real-world data streams.

Synthetic Data was generated based on a moving hyperplane [16]. The hyperplane in 2-dimensional space was used to simulate different time-changing concepts by altering its orientation and position in a smooth or sudden manner. In this study, two synthetic data consisting of one million examples each were created to characterize the data stream with gradual and sudden concept drift, respectively (Fig. 5 and Fig. 6).

Spam Filtering Data is a collection of 9324 email messages derived from the Spam Assassin collection[1]. Each email is represented by 500 attributes using the Boolean bag-of-words approach.

Electricity Data contains 45,312 instances, which was collected from the Australian New South Wales Electricity Market for every five minutes. In this market, prices are not fixed and affected by demand and supply on the market.

Covtype Data containing 581,012 instances describes seven forest cover types on a 30×30 meter grid with 54 different geographic measurements.

Sensor Data collects information (temperature, humidity, light, and sensor voltage) from 54 sensors deployed in the Intel Berkeley Research Lab over a two months period (one reading per 1-3 minutes). It totally contains 2,219,803 instances belonging to 54 classes.

Evaluation Metrics. To quantitatively evaluate SyncStream, we consider the following performance metrics:

- *Prediction performance.* We evaluate the performance of SyncStream in terms of Accuracy, Precision, Recall and F1-Measure.
- *Efficiency.* We evaluate the computation time as efficiency metric.
- *Sensitivity.* We test how classification performance is sensitive against parameter variation.

Selection of comparison methods. To extensively evaluate the proposed algorithm SyncStream, we compare its performance to several representatives of data stream classification paradigms.

Adaptive Hoeffding Tree [9]: is an incremental, anytime decision tree induction algorithm capable of learning from massive evolving data streams, using ADWIN to monitor the performance of branches in the tree and to replace them by new branches if they turn out to be more accurate.

IBLStreams [13]: an instance-based learning algorithm for classification on data streams. A prediction for a new example is achieved by combining the outputs of the neighbors of this example in the training data.

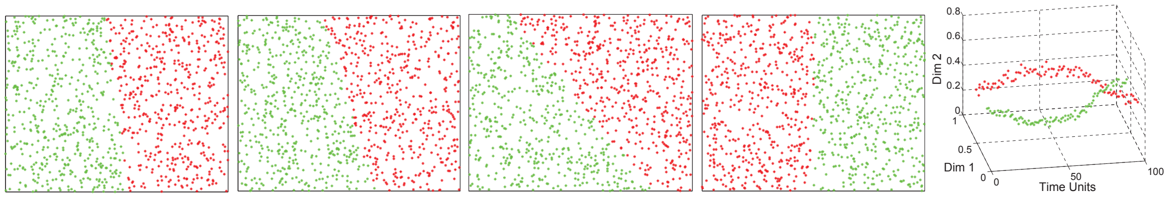
Weighted Ensemble [16]: a general framework for mining concept-drifting data streams using weighted ensemble classifiers.

OzaBagAdwin [2]: is an online bagging method by Oza and Russell with the addition of the ADWIN algorithm as a change detector and an estimator for the weights of the boosting method. When a change is detected, the worst classifier of the ensemble of classifiers is removed and a new classifier is added to the ensemble.

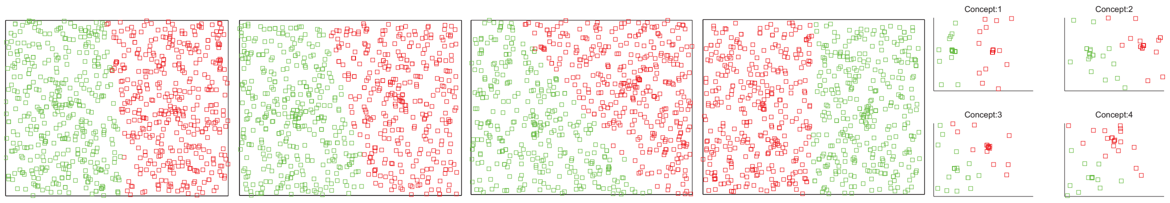
PASC [8]: is a pool and accuracy based strategy to exploit the existence of recurring concepts in the learning process and improve the classification of data streams.

We have implemented SyncStream in Java. The source code of PASC was obtained from the authors, and the IBLStreams algorithm is available at: `http://www.uni-marburg.de/fb12/kebi/research/software/iblstreams`. Adaptive Hoeffding Tree, Weighted Ensemble and OzaBagAdwin have

[1]`http://spamassassin.apache.org/`

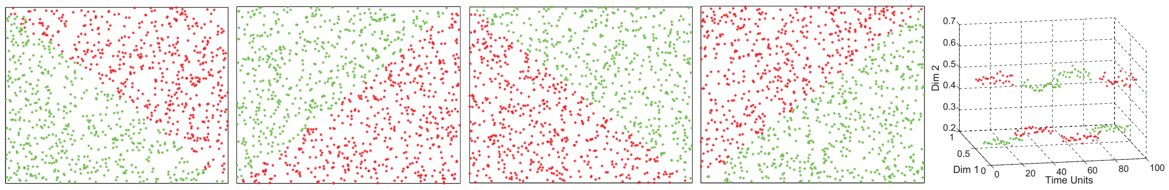


(a) T_5 (b) T_{10} (c) T_{20} (d) T_{100} (e) Data Dynamics

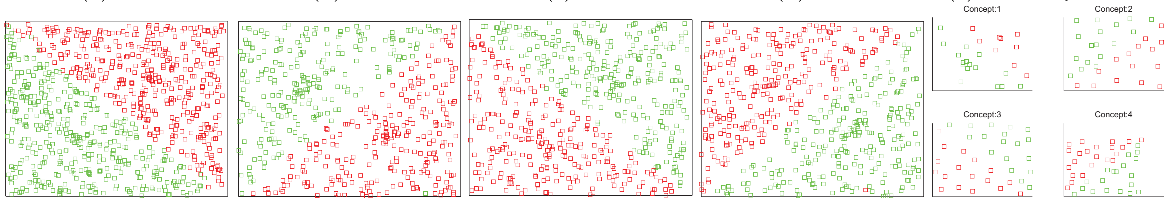


(f) P-Tree (T_5) (g) P-Tree (T_{10}) (h) P-Tree (T_{20}) (i) P-Tree (T_{100}) (j) P-Tree (Concepts)

Figure 5: Performance of SyncStream algorithm on synthetic data stream with gradual concept drift.



(a) T_{25} (b) T_{26} (c) T_{51} (d) T_{76} (e) Data Dynamics



(f) P-Tree (T_{25}) (g) P-Tree (T_{26}) (h) P-Tree (T_{51}) (i) P-Tree (T_{76}) (j) P-Tree (Concepts)

Figure 6: Performance of SyncStream algorithm on synthetic data stream with sudden concept drift.

been implemented in the MOA framework available at `http://moa.cms.waikato.ac.nz/`. All experiments have been performed on a workstation with 3.0 GHz CPU and 32 GB RAM.

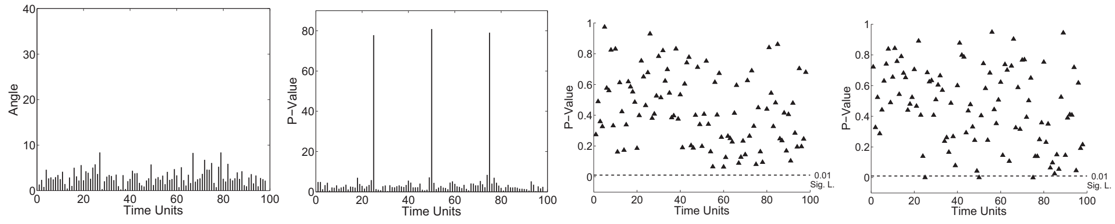
4.2 Proof of Concept

We start the evaluation with two-dimensional synthetic data streams to facilitate presentation and demonstrate some properties of SyncStream.

Concept Modeling: We first evaluate whether or not SyncStream can capture the data concept with the proposed P-Tree. Fig. 5 (a) - (d) and Fig. 6 (a) - (d) show two synthetic data streams consisting of one million examples with gradual and sudden concept drift, respectively. The evolving mean values of the data for each class are plotted in Fig. 5 (e) and Fig. 6 (e), where each time unit (T) is composed of one thousand examples. It is interesting to note that the derived prototypes in the P-Tree at any time interval enable to capture the changing data concept regardless of the type of data stream (Fig. 5 (f) - (i) and Fig. 6 (f) - (i)). Instead of learning a recent window of historical data or learning a set of base classifiers as a black box, SyncStream well captures the trend of time-changing data concepts by maintaining a set of short-term or long-term prototypes dynamically. Fig. 6 (j) displays the four changing concepts, which are preserved in the concept level of the P-Tree. For the data stream with gradual concept drift, the concepts are also learned after a specified time window (e.g., 25,000 examples in this experiment) and inserted into the concept level of the P-Tree (see Fig. 5 (j)).

Concept Drift Detection: To investigate the performance of sudden concept drift detection, Fig. 7 shows the results based on the PCA and statistical analysis, respectively. For the data stream with gradual concept drift, we can observe that the PCA-resulting angles over time are relatively stable and lower than 10°, and similarly, the derived p-values are high and no significant concept drift is detected (see Fig. 7 (a) and (c)). In contrast, for the sudden concept-drifting data stream, both the PCA and the statistical approach are capable of identifying the abruptly changing concept effectively (Fig. 7 (b) and (d)).

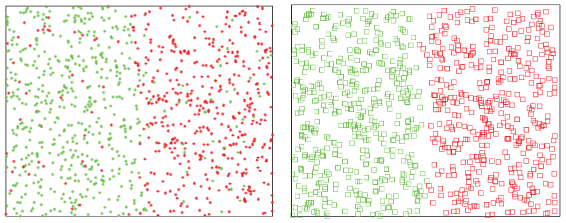
Prototype-based Data Representation: Based on the synchronization-inspired constrained clustering, important examples are further summarized as a set of prototypes. From Fig. 5 and Fig. 6, it is interesting to observe that the derived prototypes stored in the P-Tree preserve the data structure as well as the class distribution. Moreover,



(a) PCA (G) (b) PCA (S) (c) Statistical Test (G) (d) Statistical Test (S)

Figure 7: Results of two strategies for concept drift detection, where "G" and "S" indicate gradual and sudden concept drift, respectively.

inappropriate examples can be handled even in the presence of noise by error-driven representativeness learning (Fig. 8). The proposed prototype-based data representation provides an effective and efficient strategy to handle a large amount of online training data.



(a) Data stream with noise (b) Prototypes in P-Tree

Figure 8: Effective prototype-based data representation in a data stream with noise.

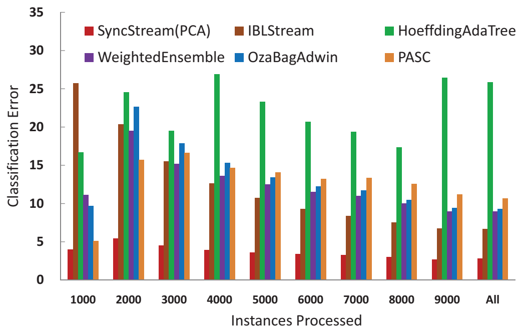


Figure 9: Performance of different data stream classification algorithms on spam data.

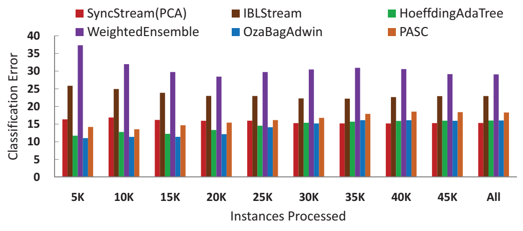


Figure 10: Performance of different data stream classification algorithms on electricity data.

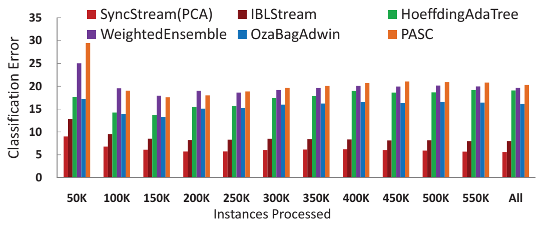


Figure 11: Performance of different data stream classification algorithms on covtype data.

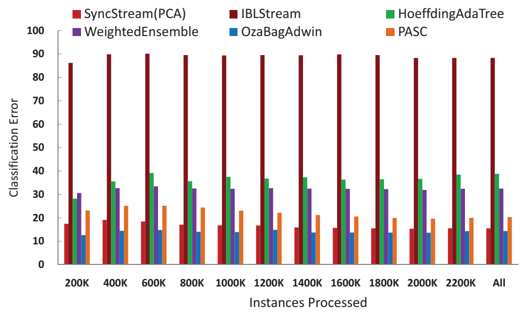


Figure 12: Performance of different data stream classification algorithms on sensor data.

4.3 Prediction Performance Analysis

In this section, we compare the performance of different stream classification algorithms on the mentioned four real data streams. For SyncStream, unless specified otherwise, the parameters were set to $maxP = 1000$, $maxC = 10$ and $chunkSize = 1000$. For all other algorithms, the default parameters suggested by the authors were used.

Fig. 9 - Fig. 12 plot the prediction performance against the number of processed instances. Generally, with an increasing number of processed examples, the prediction error of SyncStream is quite small and relatively stable. This is the result of modeling concepts by sets of dynamically adjusted prototypes. Instead of focusing on recent data or a set of data chunks, SyncStream can select the prototypes to capture the data concept on the instance level. Although IBLStream is similar to SyncStream in capturing local data characteristics by instance-based learning, it cannot adapt well to evolving concepts, and thus prediction performance fluctuates over time. HoeffdingAdaTree is a typical single model learner: due to the insufficient handling of time-changing concepts, it achieves the worst prediction performance in most cases. Although the ensemble learning ap-

Table 1: **Performance of different data stream classification algorithms on real-world data sets.**

Data	#Obj	#Dim	#Class	Methods	Acc.	Prec.	Rec.	F_1	Time (ms)
Spam	9324	500	2	**SyncStream (PCA)**	**0.9719**	**0.9590**	**0.9665**	**0.9627**	60410
				SyncStream (Stat.)	**0.9719**	**0.9590**	**0.9665**	**0.9627**	29780
				IBLStream	0.9370	0.9070	0.372	0.9218	702632
				HoeffdingAdaTree	0.9071	0.8717	0.8935	0.8824	2252
				WeightedEnsemble	0.8629	0.8139	0.8176	0.8158	13000
				OzaBagAdwin	0.9108	0.8765	0.8973	0.8868	10848
				PASC	0.8931	0.9178	0.9415	0.9295	**2142**
Electricity	45,312	8	2	**SyncStream (PCA)**	0.8457	0.8423	0.8420	0.8421	3118
				SyncStream (Stat.)	**0.8459**	**0.8425**	0.8419	0.8422	3280
				IBLStream	0.7688	0.7648	0.7584	0.7616	7512
				HoeffdingAdaTree	0.8398	0.8409	0.8296	0.8352	**750**
				WeightedEnsemble	0.7092	0.7024	0.7022	0.7023	3920
				OzaBagAdwin	0.8397	0.8399	0.8302	0.8350	3810
				PASC	0.8170	0.8316	**0.8552**	**0.8432**	1327
Covtype	581,012	54	7	**SyncStream (PCA)**	**0.9438**	**0.8915**	**0.8980**	**0.8947**	207176
				SyncStream (Stat.)	**0.9438**	**0.8915**	**0.8980**	**0.8947**	226331
				IBLStream	0.9197	0.8620	0.8573	0.8597	3005412
				HoeffdingAdaTree	0.8087	0.7085	0.7173	0.7129	**31692**
				WeightedEnsemble	0.8033	0.7476	0.6690	0.7061	365582
				OzaBagAdwin	0.8383	0.7848	0.7722	0.7784	176000
				PASC	0.7972	0.8291	0.8348	0.8319	125387
Sensor	2,219,803	5	54	**SyncStream (PCA)**	0.8453	0.8508	0.8460	0.8484	244110
				SyncStream (Stat.)	0.8453	0.8508	0.8460	0.8484	246492
				IBLStream	0.1173	0.1805	0.1397	0.1575	345930
				HoeffdingAdaTree	0.6121	0.6269	0.6282	0.6276	**166600**
				WeightedEnsemble	0.6752	0.7918	0.6805	0.7319	2105133
				OzaBagAdwin	**0.8563**	**0.8660**	**0.8639**	**0.8649**	1343065
				PASC	0.7968	0.8420	0.8150	0.8283	264161

proaches, Weighted Ensemble, OzaBagAdwin and PASC, allow adapting to changing concepts quickly according to previous studies, the classification accuracies are not promising, which may caused by the two limitations stated in Section 1. Table 1 further summarizes the performance of the algorithms in terms of different evaluation measures. Regarding the computation time, HoeffdingAdaTree is the fastest algorithm due to the Hoeffding bound, yet the classification accuracy suffers. SyncStream, Weighted Ensemble, OzaBagAdwin and PASC are comparable, while IBLStream is the most time-consuming algorithm. From the experiments, we can see that SyncStream not only allows making accurate classifications, but also working efficiently in terms of computation time (Table 1).

4.4 Sensitivity Analysis

In this section, we perform a sensitivity analysis of SyncStream on the cover type data.

Maximum Boundary: As stated in Section 3.4.2, the maximum number of prototypes $maxP$ needs to be specified to indicate how many prototypes can be dynamically maintained in the P-Tree according to the available computational resources. The higher the value of $maxP$, the

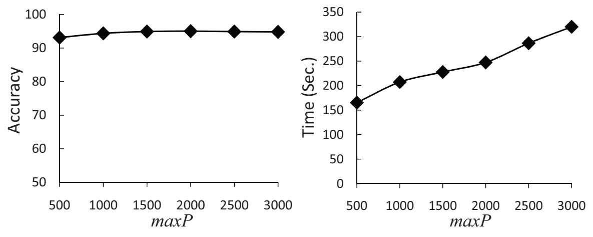


Figure 13: The sensitivity analysis with $maxP$.

more prototypes can be managed to model the concepts. In this experiment, we varied $maxP$ from 500 to 3000. Fig. 13 (a) plots the classification accuracies dependent on the numbers of prototypes, the corresponding computation time is shown in Fig. 13 (b). We observe that the classification performance is quite stable, while the computation time is gradually growing due to the increased effort needed for nearest neighbor search.

Chunk Size: The chunk size determines how many examples have been used to examine whether the concept has changed across two consecutive data chunks. Fig. 14 (a) shows the classification performance with respect to different chunk sizes ranging from 500 to 3000. Similarly, the corresponding number of time-changing concepts is reported in Fig. 14 (b). With increasing chunk size, the number of detected concepts is gradually decreasing, as more data make the transition from one concept to another appear smoother. However, thanks to error-driven representativeness learning, the set of dynamical prototypes in the P-Tree has already learned the evolving concepts implicitly. By combining representativeness learning and concept drift detection, SyncStream allows adapting to both gradual and sudden concept drift quickly, which results in high classification accuracies in the experiments (Fig. 14 (a)).

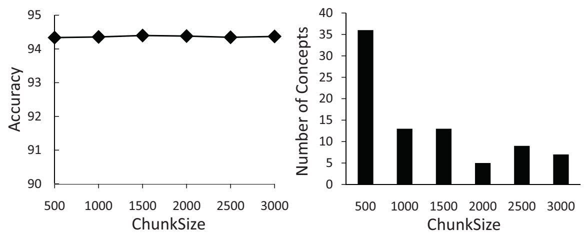


Figure 14: Sensitivity analysis with *chunkSize*.

5. DISCUSSION AND CONCLUSION

In this paper, we introduce a new prototype-based classification algorithm, SyncStream, to learn from evolving data streams. While existing approaches focus on learning a single model on a window of recent data or a set of base classifiers on recent data chunks, time-changing concepts may not be learned well in this way due to two factors: the difficulty of selecting a suitable horizon of the training data and the loss of relevant historical data. In the light of these problems, this paper proposes error-driven representativeness learning to determine the importance of training examples. Only representative examples, allowing to model the current concept, are kept and further summarized as a smaller set of prototypes by synchronization-inspired constrained clustering. With this strategy, SyncStream allows dynamically learning a set of prototypes to capture evolving concepts implicitly on the level of instances. Moreover, in order to adapt to abrupt concept drift quickly, two heuristic concept drift detection approaches are introduced. Equipped with both implicit and explicit concept drift handling, SyncStream allows modeling time-changing concepts (either gradual or sudden) effectively. One other attractive property of SyncStream is its any-memory property. Owing to the prototype-based data representation, a multi-scale representation of the data (a set of prototypes) is possible. Although SyncStream is an instance-based learning scheme, it largely differs from traditional instance-based learning such as IBLStreams. One main difference is that IBLStreams keeps track of recent instances, while SyncStream dynamically learns a set of prototypes and supports efficient data maintenance. In comprehensive experiments, we have shown that SyncStream outperforms several other state-of-the-art data stream classification methods.

6. REFERENCES

[1] A. Bifet and R. Gavalda. Learning from time-changing data with adaptive windowing. In *Proceedings of the 11th SIAM International Conference on Data Mining*, pages 443–448, 2007.

[2] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 139–148, 2009.

[3] C. Böhm, C. Plant, J. Shao, and Q. Yang. Clustering by synchronization. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 583–592, 2010.

[4] E. Brunner and U. Munzel. The nonparametric Behrens-Fisher problem: Asymptotic theory and a small-sample approximation. *Biometrical Journal*, 42(1):17–25, 2000.

[5] A. Dries and U. Rückert. Adaptive concept drift detection. *Statistical Analysis and Data Mining*, 2(5-6):311–327, 2009.

[6] J. Gama, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, (5-6):1–35, 2013.

[7] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. *Lecture notes in computer science*, pages 286–295, 2004.

[8] M. J. Hosseini, Z. Ahmadi, and H. Beigy. Using a classifier pool in accuracy based tracking of recurring concepts in data stream classification. *Evolving Systems*, 4(1):43–60, 2013.

[9] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106, 2001.

[10] L. I. Kuncheva and J. S. Sánchez. Nearest neighbour classifiers for streaming data with delayed labelling. In *IEEE 8th International Conference on Data Mining*, pages 869–874, 2008.

[11] M. Neuhäuser and G. D. Ruxton. Distribution-free two-sample comparisons in the case of heterogeneous variances. *Behavioral Ecology and Sociobiology*, 63(4):617–623, 2009.

[12] S. Papadimitriou, A. Brockwell, and C. Faloutsos. Adaptive, hands-off stream mining. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 560–571, 2003.

[13] A. Shaker and E. Hüllermeier. IBLStreams: a system for instance-based classification and regression on data streams. *Evolving Systems*, 3(4):235–249, 2012.

[14] J. Shao, X. He, C. Böhm, Q. Yang, and C. Plant. Synchronization-inspired partitioning and hierarchical clustering. *IEEE Transaction on Knwoledge and Data Engineering*, 25(4):893–905, 2013.

[15] K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *Proceedings of the 7th International Conference on Machine Learning*, pages 1103–1110, 2000.

[16] H. Wang, W. Fan, P. S. 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*, pages 226–235, 2003.

[17] P. Zhang, B. J. Gao, X. Zhu, and L. Guo. Enabling fast lazy learning for data streams. In *IEEE 11th International Conference on Data Mining*, pages 932–941, 2011.