Knowl Inf Syst (2010) 22:371–391
DOI 10.1007/s10115-009-0206-2

REGULAR PAPER

Tracking recurring contexts using ensemble classifiers: an application to email filtering

Ioannis Katakis · Grigorios Tsoumakas · Ioannis Vlahavas

Received: 16 July 2008 / Revised: 31 January 2009 / Accepted: 15 March 2009 /
Published online: 24 April 2009

Abstract Concept drift constitutes a challenging problem for the machine learning and data mining community that frequently appears in real world stream classification problems. It is usually defined as the unforeseeable concept change of the target variable in a prediction task. In this paper, we focus on the problem of *recurring contexts*, a special sub-type of concept drift, that has not yet met the proper attention from the research community. In the case of recurring contexts, concepts may re-appear in future and thus older classification models might be beneficial for future classifications. We propose a general framework for classifying data streams by exploiting stream clustering in order to dynamically build and update an ensemble of incremental classifiers. To achieve this, a transformation function that maps batches of examples into a new *conceptual* representation model is proposed. The clustering algorithm is then applied in order to group batches of examples into concepts and identify recurring contexts. The ensemble is produced by creating and maintaining an incremental classifier for every concept discovered in the data stream. An experimental study is performed using (a) two new real-world concept drifting datasets from the email domain, (b) an instantiation of the proposed framework and (c) five methods for dealing with drifting concepts. Results indicate the effectiveness of the proposed representation and the suitability of the concept-specific classifiers for problems with recurring contexts.

A preliminary version of this paper appears in the proceedings of the 18th European Conference on Artificial Intelligence, Patras, Greece, 2008.

I. Katakis (✉) · G. Tsoumakas · I. Vlahavas
Department of Informatics, Aristotle University of Thessaloniki,
54124 Thessaloniki, Greece
e-mail: katak@csd.auth.gr

G. Tsoumakas
e-mail: greg@csd.auth.gr

I. Vlahavas
e-mail: vlahavas@csd.auth.gr

1 Introduction

Recent advances in sensor, storage, processing and communication technologies have enabled the automated recording of data, leading to fast and continuous flows of information, referred to as data streams [1]. Examples of data streams are the web logs and web page click streams recorded by web servers, transactions like credit card usage, data from network monitoring and sensor networks, video streams such as images from surveillance cameras, news articles in an RSS reader, etc.

The dynamic nature of data streams requires continuous or at least periodic updates of the current knowledge in order to ensure that it always includes the information content of the latest batch of data. This is particularly important in stream classification problems where the concept of a target class and/or the data distribution changes over time. This phenomenon is commonly known as *concept drift* [2].

A very special type of concept drift is that of *recurring contexts* [3] where concepts that appeared in the past may recur in the future. Although the phenomenon of reappearing concepts is very common in real world problems (weather changes, buyer habits, etc), only few stream classification methodologies take it into consideration [3–5].

In this paper, we propose an ensemble of classifiers that utilizes a new conceptual representation model for data streams suitable for problems involving concept drift. More specifically, we map batches of data into what we call *conceptual vectors*. These vectors contain conceptual information that describes the established concepts in a particular batch of data. Taking advantage of the proposed representation, we apply clustering to the stream of conceptual vectors. This way, we organize and summarize batches of instances into clusters which represent different concepts. The final objective is to train an incremental classifier on every cluster of conceptual vectors. When a new batch of instances arrives, we identify the concept (cluster) in which this batch belongs to and apply the corresponding classifier in order to predict the true labels of its instances.

An instantiation of this framework was evaluated on two email datasets and compared against five methodologies for dealing with drifting concepts. Results indicate the effectiveness of the proposed representation and the suitability of the concept-specific classifiers that outperform the rest of the methods in problems with recurring contexts. The main contributions of this work are the following:

- A new conceptual representation model suitable for problems with concept drift and recurring contexts.
- The proposal of a general framework for dealing with recurring contexts.
- An experimental study involving an instantiation of the aforementioned framework and five methodologies for dealing with drifting concepts.
- Two new datasets from the email domain that simulate concept drift that we make available online. In addition, the first dataset includes recurring contexts.

The rest of the paper is organized as follows. In Sect. 2, we present background information on mining data streams whereas in Sect. 3 we summarize related work on concept drift detection and adaptation. Section 4 proposes a batch transformation procedure that will be used as a basic component of the framework proposed in Sect. 5. In Sect. 6 we present the datasets created and used in the experimental evaluation that are presented in Sect. 7. Section 8 concludes the paper by presenting our plans for future work.

2 Mining data streams

All data mining problems comprise an interesting challenge in a streaming environment [6–8] mainly because of the one-pass requirement and strict memory limitations. In this section we discuss some of the main issues in mining streams and particularly in classification and clustering.

2.1 Stream classification and concept drift

The stream classification problem is modeled as follows: Unlabeled instances arrive one at a time and need to be rapidly classified into one out of a predefined set of labels. The stream is considered infinite and therefore mining algorithms cannot store many instances into main memory and consequently can process them only once. After classification, the true label of the sample is considered to be available [9–13] and therefore (a) the system's performance can be calculated and (b) the system's model can be updated. It is not uncommon to assume that the data (examples) arrive at batch mode [14,15]. For successful automatic classification of concept drifting data streams, we are not only seeking for fast and accurate incremental algorithms, but also for complete methodologies that can detect and quickly adapt to time varying concepts.

There are two main kinds of concept drift as follows: (a) instant (abrupt or sudden) and (b) gradual. In the case of *abrupt concept drift*, concepts alter instantly. Consider a machine learning enhanced email filtering system. The task is to filter out email messages that are considered uninteresting (junk) for the user. The user informs the system about his/her interests by marking messages as "interesting" or "junk". For example, a user subscribed to a computer-related mailing list might stop instantly to be interested in messages containing notebook reviews just after the purchase of a notebook. The classifier must rapidly adapt to this change and accurately classify future messages. An example of gradual concept drift can be observed in the spam filtering problem. Spam messages change (usually become harder to identify) at a certain rate. Again, the classifier must be able to follow this variation and maintain a stable performance.

An interesting situation that can occur in both instant and gradual concept drift is the repetition of certain concepts at different time periods. In the email filtering system for example, the user could re-gain interest in topics that he/she was interested in the past. Furthermore, in our second example, there are some certain groups of spam messages that re-appear in certain time periods (e.g. spam messages about gifts on mother's day or Christmas). This subtype of concept drift has been noted as *recurrent themes*[5] or *recurring contexts*[3]. In this case, the system should recognize the re-appearing concepts and re-activate older classification models.

Tsymbal [2] summarizes the characteristics of an ideal concept drift handling system in the following three requirements:

- Quick adaptation to concept drift.
- Robustness to noise and ability to distinguish it from concept drift.
- Ability to recognize and react to reoccurring themes.

2.2 Stream clustering

The task in the stream clustering problem is to immediately assign an item to a cluster as it arrives. The problem is far more interesting than the case of static data since there is no prior knowledge of the data distribution and thus, new clusters should be created and old

ones might be merged dynamically. Well known representatives of streaming clusterers are SWClustering [16], STREAM [17], HPStream [18] and BIRCH [19]. An informative review on stream clustering can be found in [7].

3 Related work

A common observation is that methodologies proposed for tackling concept drift can be organized into three main groups, namely *instance selection*, *instance weighting* and *ensemble methods* [2]. Furthermore, we consider an additional group composed of *quickly adapting incremental algorithms*. In this section, we present these groups commenting on (a) main assumptions made, (b) advantages and disadvantages and (c) requirements that a classifier should fulfill in order to operate in the framework of each group.

3.1 Instance selection

In instance selection, proposed systems try to select the most appropriate set of past cases in order to make future classifications. Typical representatives of this group are moving window-based methods (also known as time window),[1] where the classifier is always trained from a fixed or adaptive size moving window of examples [20]. In most cases, this window is constructed out of the last w examples of the stream. The assumption made in these approaches is that older examples are practically worthless for classification of new data and therefore, adapting to concept drift is tantamount to successfully forgetting old instances [3,21]. In order to operate in a moving window framework, a classifier should implement two additional functions that incorporate/abstract an instance into/from the model respectively. The main advantage of moving windows is the fact that the classifier always models the last batch of information and therefore might be accurate for future classifications even in concept drifting environments. Although the problem of determining the window size has been solved by using adaptive time windows [20], these approaches do not consider recurring contexts.

Other approaches that are considered in this group are methods that maintain a considerable amount of examples in memory in order to select dynamically a training set for the current classifier [21–23]. Although the intuition of exploiting the history of the stream is correct (especially in the case of recurring contexts), the memory limitations of data stream applications make these approaches impractical.

3.2 Instance weighting

In instance weighting, the main assumption made is that old knowledge becomes less important as time goes by. All instances are taken into consideration for building classification models, but new instances have larger effect. To achieve this goal, a weighting scheme is defined [21]. In order to operate within this framework, a classifier must be updateable and support weighted learning. The naive Bayes classifier, for example, fulfills these requirements. In case of concept drift, weighted learning will aid the model to adapt quickly, but in case of recurring contexts, the system will fail to soundly exploit information from old data.

[1] Very frequently in data stream classification, the stream is modeled as a sequence of examples without taking into consideration the actual time-stamp of each example (arrival time). Therefore we consider the term "moving window" more appropriate.

3.3 Ensemble methods

In ensemble methods, the main strategy is to maintain a dynamic set of classifiers. When a decrease in performance is observed, new models are added into the ensemble whereas old and bad performing models are removed. For classification, decisions of models in the ensemble are combined, usually with a weighted voting scheme [9,13–15,24,25]. The advantage of weighted ensembles over single classifiers in the data stream classification problem has been proved empirically and theoretically [10,14]. However, few ensemble methods have been designed to take into consideration the problem of recurring contexts [3–5].

Specifically, in problems where concepts re-occur, models of the ensemble should be maintained in memory even if they do not perform well in the latest batch of data. Moreover, every classifier should be specialized in a unique concept, meaning that it should be trained from data belonging to this concept and used for classifying similar data.

In [4], a methodology that identifies concepts by grouping classifiers of similar performance on specific time intervals is described. Weights are then assigned to classifiers according to performance on the latest batch of data. Predictions are made by using weighted averaging. Although this strategy fits very well with the recurring contexts problem, it has an offline step for the discovery of concepts that is not suitable for data streams. In particular, this framework will probably be inaccurate with concepts that did not appear in the training set.

An interesting idea is presented by Forman [5], where a great number of "daily" classifiers are maintained in memory. Their predictions for incoming data are incorporated as additional features into the most recent classifier which will learn how to take advantage of them in order to make future classifications. Unfortunately, the framework is not designed for data streams and in addition classifiers are built on small sample sets as they are not organized into concepts.

Our approach also resides in the category of ensemble methods, by maintaining a classifier for every concept that appeared in the stream and activating the most appropriate one when needed. Concepts are identified by exploiting stream clustering. A recent work that also exploits clustering but for discriminating drift from novelty is presented in [26].

3.4 Quickly adapting incremental algorithms

Methods like CVFDT [27] (concept-adapting very fast decision tree learner) adapt rapidly to the newest batch of data in order to cope up with the drift. In particular, CVFDT stays current while making the most of the old data by growing an alternative subtree whenever an old one becomes questionable, and replacing the old with the new when the new becomes more accurate. Unfortunately, methods of this category have not been designed to track recurring contexts.

4 Data representation model

In this section, we first discuss representation issues in data streams and then propose a mapping procedure that transforms data into a new conceptual representation model suitable for problems involving concept drift and recurring contexts.

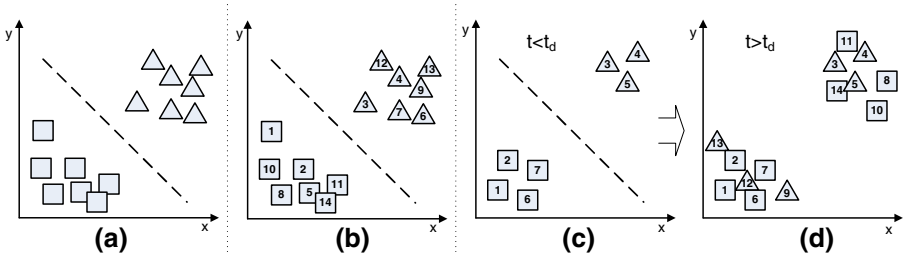

Fig. 1 Static (**a**) and stream (**b**) classification problem without concept drift. Stream classification problem with concept drift before (**c**) and after (**d**) drift time point (t_d). *Numbers* at items denote order of arrival

4.1 Representation issues in concept drifting data streams

The main difficulty in concept drifting data is that geometrically close items in the feature space will belong to different classes at different time periods. We use a simple example in order to demonstrate this problem and explain the intuition behind our transformation procedure. Figure 1a displays a static (without concept-drift) two dimensional classification problem. Items of class "square" (low x and low y) and items of class "triangle" can be easily separated with a linear classifier. Figure 1b depicts a stream classification problem with no concept drift. Numbers at items denote order of arrival. Classes still share common characteristics as time passes and therefore can be (even incrementally) separated in the current feature space. Figure 1c and d represents a stream classification problem with concept drift in two successive time periods. In the first period (Fig. 1c) before drift time point (t_d) items with low x and y values characterize the class square. However, after drift time point (i.e. after item 7 arrives—Fig. 1d), the concepts of class square and triangle change (in our case, reverse) making difficult for a classifier trained from previous data to classify incoming instances.

4.2 Conceptual representation model

In order to overcome the aforementioned obstacles and to identify (recurring) concepts in the data stream we apply a transformation function (M) that maps batches of examples into what we call *conceptual vectors*. These vectors contain conceptual information that describes the concepts that are established in a particular batch of data. The goal, as we will discuss later, is to exploit this conceptual representation in order to identify concepts and apply an individual classifier in each one.

First, the data stream is separated into a number of small batches of examples. Each batch is transformed into a conceptual vector that consists of a number of *conceptual feature* sets. Each feature set consists of a number of elements (see below) and corresponds to a feature from the initial feature space. Let us assume that unlabeled (U) and labeled (L) examples are represented as vectors:

$$\mathbf{x}_U = (x_1, x_2, \ldots, x_n) \tag{1}$$

$$\mathbf{x}_L = (x_1, x_2, \ldots, x_n, c_j) \tag{2}$$

where x_i is the value of the feature f_i, and $c_j \in C$ with C being the set of available classes (labels). The number of attributes is n and the number of classes is m. Let B_U and B_L be a *batch* of unlabeled and labeled instances of size b,

$$B_U = \left\{\mathbf{x}_{U(k)}, \mathbf{x}_{U(k+1)}, \ldots, \mathbf{x}_{U(k+b-1)}\right\} \tag{3}$$

$$B_L = \left\{\mathbf{x}_{L(k)}, \mathbf{x}_{L(k+1)}, \ldots, \mathbf{x}_{L(k+b-1)}\right\} \tag{4}$$

where $\mathbf{x}_{U(k)}$ and $\mathbf{x}_{L(k)}$ are the first instances of each batch. Every batch of examples (B_L) is transformed into a conceptual vector $\mathbf{Z} = (z_1, z_2, \ldots, z_n)$, where z_i are the conceptual feature sets. Considering the examples contained in the batch B_L and for every feature f_i of the original feature space the conceptual feature sets are calculated as follows:

$$z_i = \begin{cases} \left\{P^v_{i,j} : j = 1, \ldots, m, v \in V_i\right\}, & \text{if } f_i \text{ is nominal} \\ \left\{\mu_{i,j}, \sigma_{i,j} : j = 1, \ldots, m\right\}, & \text{if } f_i \text{ is numeric} \end{cases} \tag{5}$$

where $P^v_{i,j} = P(f_i = v|c_j), i \in [1, n], j \in [1, m], v \in V_i$, where V_i is the set of values of the nominal attribute f_i. Following the paradigm of the naive Bayes classifier, $P^v_{i,j}$ is considered to be equal to $n_{v,j}/n_j$, where $n_{v,j}$ is the number of training samples of class c_j having the value v at attribute i in batch B_L and n_j is the number of training samples belonging to c_j in batch B_L. For continuous (numeric) attributes, we use the mean ($\mu_{i,j}$) and standard deviation ($\sigma_{i,j}$) of attribute f_i for training samples of class c_j in batch B_L. After each instance arrives, if the true class is known, we incrementally calculate the above statistics in order to construct the conceptual vector immediately after the batch is finished.

The expected average dimensionality of the new representation will be approximately $a\bar{v}m + e2m$, where a and e are the number of nominal and numeric attributes, respectively, and $\bar{v}$ is the average number of values for nominal attributes. For the text classification problem, we study in this paper with the widely used Boolean bag-of-words representation ($a = n, e = 0$) the space complexity is $O(nm)$.

The notion behind this representation is that every element of the conceptual vectors expresses in what degree a feature characterizes a certain class. For example, in email classification, if we notice that in two different batches of examples $P(\text{cpu}|\text{interesting})$ and $P(\text{monitor}|\text{interesting})$, are similarly high, we could assume that these batches might belong to the same or similar conceptual theme (i.e. where computer-related emails are considered interesting). The same principle applies to numeric attributes. Batches of instances that have similar mean values for many attributes, should be close conceptually.

Consequently, conceptual distance between two batches $B_{L(\psi)}$and $B_{L(\omega)}$ can be defined as the Euclidean distance of the corresponding conceptual vectors:

$$\begin{aligned} \text{ConDis}(B_{L(\psi)}, B_{L(\omega)}) &= \text{Euclidean}(Z_{(\psi)}, Z_{(\omega)}) \\ &= \left\{\text{dis}(z_{1(\psi)}, z_{1(\omega)}) + \cdots + \text{dis}(z_{n(\psi)}, z_{n(\omega)})\right\}^{1/2} \end{aligned} \tag{6}$$

where,

$$\text{dis}(z_{i(\psi)}, z_{i(\omega)}) = \left(\zeta^1_{i(\psi)} - \zeta^1_{i(\omega)}\right)^2 + \cdots + \left(\zeta^l_{i(\psi)} - \zeta^l_{i(\omega)}\right)^2 \tag{7}$$

and $\zeta^j_{i(\psi)}$ is the jth element of the ith conceptual feature-set of the vector ψ, whereas l is the length of the feature set.

This mapping procedure tries to ensure that the more similar two batches will be conceptually, the closer in distance their corresponding conceptual vectors will be. The definition of this distance will be also beneficial for the clustering algorithm of the framework we present in the following section. Note that the proposed representation could be replaced by another function that would map a batch of instances into a space where similar batches conceptually would be close in distance. Figure 2 depicts the stream classification problem with concept drift in a representation space like the one discussed in this section. Note that items in this

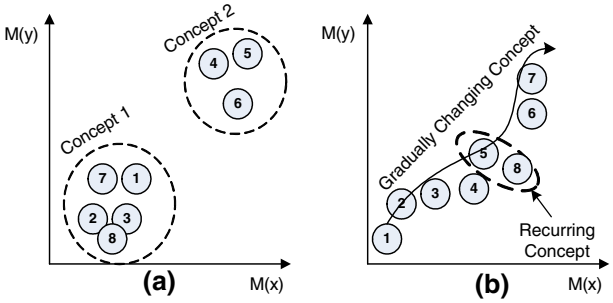

Fig. 2 Batches of information represented in a conceptual representation model. **a** Sudden concept drift with recurring contexts. **b** Gradual concept drift with recurring context. *Numbers* at items denote order of arrival

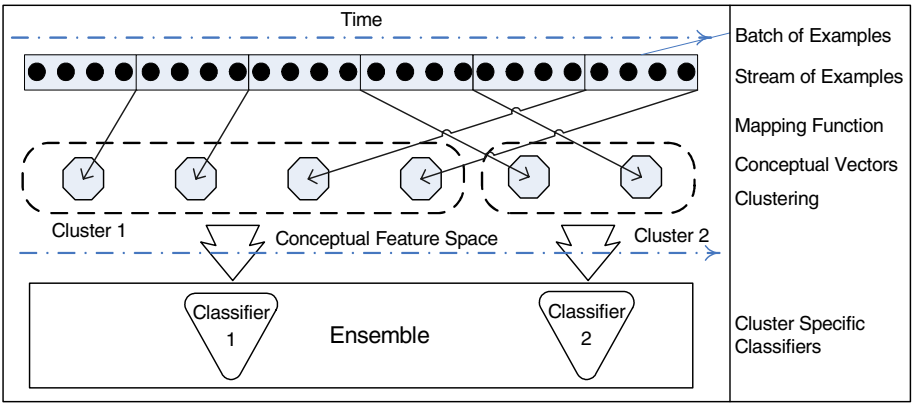

Fig. 3 The CCP framework

case represent *batches* of examples. Numbers on items denote sequence of arrival. Figure 2a represents sudden concept drift whereas Fig. 2b represents gradual concept drift. In both cases there are recurring contexts. More specifically, in Fig. 2a, the first batches (1,2,3) belong in the same concept (concept 1). After batch 3, concept drift occurs. However, concept 1 re-appears with batches 7 and 8. Intuitively, in this case, the ideal solution would be able to train an individual classifier for each concept trained from the corresponding batches. In Fig. 2 the concept shifts gradually. However, batch 8 appears to be very close conceptually with batch 5. In this case, old classifiers should be maintained in case, an older concept re-appears.

In order to evaluate the suitability of the proposed representation, we examine later the cluster assignments of the conceptual vectors constructed from the email streams used in this paper.

5 The CCP framework

We propose a data stream classification framework conceptual clustering and prediction (CCP) that dynamically creates an ensemble of classifiers by exploiting stream clustering in order to group the stream's conceptual vectors (see Fig. 3). The main aim of clustering is the

on-line identification of concepts and the maintenance of cluster-specific classifiers in the ensemble.

The main components of the framework are:

- *A mapping function (M)*, that transforms batch $B_{L(j)}$ into conceptual vector Z_j.
- *An incremental clustering algorithm (R)*, that groups conceptual vectors into clusters.
- *An incremental classifier (h)*, for every concept (cluster) discovered in the stream.

What is maintained as time (t) passes is a set of clusters $G_t = \{g_1, g_2, \ldots, g_q\}$ and an ensemble of corresponding classifiers $H_t = \{h_1, h_2, \ldots, h_q\}$. Classifier h_i is trained from batches that belong conceptually to cluster g_i. Initially, $G_o = \emptyset$ and $H_o = \emptyset$. The maintenance of clusters depends on the clustering algorithm used. In the leader–follower algorithm exploited in this work only the centroids of clusters need to be maintained. The main operations of the CCP framework are *Dynamic Training* and *Online Classification* of incoming instances. We briefly describe them below.

- **Dynamic training** After the arrival of a batch $B_{L(j)}$, CCP constructs its conceptual vector $Z_{(j)}$. Incremental clusterer (R) either assigns Z_j to a cluster $g_s \in G$ or creates a new cluster if necessary. In the second case, a new incremental classifier is created and trained on these data. Otherwise, the corresponding incremental classifier h_s is updated with the examples of $B_{L(j)}$.
- **Online classification** Instances of batch $B_{U(j+1)}$ are classified by classifier h_s that corresponds to the cluster of the previous conceptual vector $Z_{(j)}$[2].

Note that the above two operations execute in parallel when true labels of examples are available. By classifying the current batch according to the classifier built from the cluster of the previous batch we make a kind of a "locality assumption". We assume that in successive batches (of small size), most of the time will belong to the same concept. This is an assumption often made in such problems (see STAGGER Concepts [9] or Moving Hyperplanes [27]) mainly because the only way to detect drift is by observing mistakes of the model in a small time window, and then adapt in order to achieve better performance in the next batch of data. Continuous changes of concepts at almost every instance are the special case that are difficult to discriminate from noise. It is not usually observed in real life problems and therefore we do not deal with it in this paper. Concerning the batch size, we should note that it is a problem-dependent parameter but, in general, it should be small enough in order not to invalidate the aforementioned locality assumption and sufficient in order to calculate the conceptual vectors.

The pseudocode of the framework is presented in Fig. 4. Note that the true labels of the instances are not immediately (just after classification) required for CCP. As can be seen from line 12 and 18 updates are performed only after a batch is finished. Moreover, the framework could work without continuous knowledge of the true labels. In particular, CCP could execute lines 7–19 only when a number of true labels (e.g. equal to the batch size) are available.

As basic implementation components, any incremental clustering algorithm could be exploited like the one mentioned in Sect. 2.2 or the basic leader–follower clusterer [28]. This clusterer (see details in Sect. 7) either assigns a cluster to an item or creates a new one if necessary. Therefore there is no requirement for keeping old items in order to merge or split clusters. For the classification component, any incremental classifier could be used.

It is important to notice that there is no need to maintain past batches or conceptual vectors in memory. What is maintained is the cluster information (i.e. the centroids' of each cluster)

[2] Inevitably, the first batch of instances will be randomly classified by the first untrained classifier which can be updated when labels are known.

Input: A stream of unlabeled examples $x_{U(i)}$. After batch is finished, true labels are considered known. b is the batch size.
Output: Continuous classifications p_i of every $x_{U(i)}$

```
1  begin
2      H ← ∅, G ← ∅;
3      s ← 1, j ← 1;
4      H ← H ⋃ {h_s};
5      for i=1 to infinity do
6          p_i ← h_s.classify(x_{U(i)});
7          if i mod b = 0 then                          /* batch finished */
8              B_{L(j)} ← {x_{L(i−b)}, ..., x_{L(i)}};
9              Z_j ← M(B_{L(j)}) ;
10             s ← R.cluster(Z_j);
11             if ∃ g_s ϵ G then
12                 h_s.update(B_{L(j)});
13                 R.update(Z_j);
14             else
15                 g_s ← Z_j;
16                 G ← G ⋃ {g_s};
17                 H ← H ⋃ {h_s};
18                 h_s.update(B_{L(j)});
19             j ← j + 1
20 end
```

Fig. 4 The main operation of the CCP framework

and the appropriate classifiers for every cluster. Concerning the time complexity of the proposed framework compared to a simple incremental classifier the additional computational cost is (a) the transformation of a batch into a conceptual vector and (b) the clustering of conceptual vectors in order to find the right classifier for classification and update.

6 Datasets

In order to evaluate the proposed methodology, we constructed two high dimensional datasets from the email filtering domain. The first one represents the problem of sudden concept drift and recurring contexts and the second one represents gradual concept drift. Both datasets are available in Weka (ARFF) format in Boolean bag-of-words vector representation at the following URL: http://mlkd.csd.auth.gr/concept_drift.html (Datasets 3).

6.1 Emailing list dataset

The emailing list (elist) dataset simulates a stream of email messages from different topics that are sequentially presented to a user who then labels them as interesting or junk according to his/her personal interests. The filtering task is to incrementally take advantage of this user feedback in order to train a model that will automatically classify messages into

Table 1 Emailing list dataset (elist)

	0–300	300–600	600–900	900–1,200	1,200–1,500
Medicine	+	−	+	−	+
Space	−	+	−	+	−
Baseball	−	+	−	+	−

interesting/junk as they arrive. To construct this email stream we have collected messages from usenet posts that exist in the 20 Newsgroup collection [29].

The topics that were used are: science/medicine, science/space, and recreation/sports/baseball. The dataset is a stream of 1,500 examples and 913 attributes which are words that appeared at least 10 times in the corpus (boolean bag-of-words representation). We split the stream into five time periods of 300 examples. In the first period, we consider that the user is only interested in messages of the topic medicine. At the end of each period, the user's interest in a topic alters in order to simulate the occurrence of concept drift. Table 1 shows which messages are considered interesting (+) or junk (−) in each period.

6.2 Spam filtering dataset

For this dataset, we used email messages from the Spam Assassin Collection.[3] This corpus comes in four parts: spam, spam2, ham (legitimate), and easy ham, which is a collection of more easily recognized legitimate messages. The spam ratio of the Spam Assassin collection is approximately 20%. In order to convert this collection into a longitudinal data set we extracted the date and time that the mail was sent. Then we converted the time into GMT time. Date was also changed where needed. We stamped each email with its date and time by renaming it in the format yyyy MM dd hh mm ss (yyyy: year, MM: month, dd:day, hh: hours, mm: minutes, ss: seconds). If an email was more than once in the corpus (sometimes a user may get the same email more than once) we kept all copies. All attachments were removed. The boolean bag-of-words approach was used for representing emails. This dataset consists of 9,324 instances, 500 attributes (words derived after applying feature selection with the x^2 measure). As we have observed at a previous study [30], the characteristics of spam messages in this dataset gradually change as time passes (gradual concept drift).

7 Evaluation

In this section, we first comment on the suitability of the proposed conceptual representation for the problem of identifying recurring contexts. We then perform comparative evaluation of an instantiation of the proposed framework with five stream classification methods in the two datasets described in the previous section.

7.1 Evaluating the Conceptual Representation Model

In order to evaluate the representation model proposed in Sect. 4.2, we have obtained the conceptual vectors from both datasets and applied two off-line clustering algorithms in order to study cluster assignments. We used a batch size of 50 instances for both datasets which led to 30 conceptual vectors for the elist dataset and 186 for the spam filtering dataset.

[3] The Apache SpamAssassin Project—http://spamassassin.apache.org/.

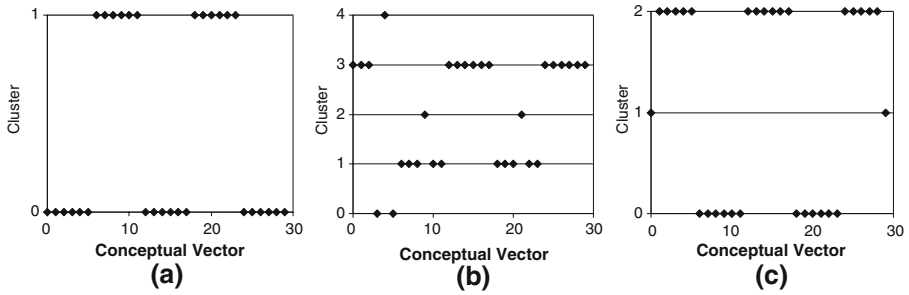

Fig. 5 Cluster assignments—elist dataset. **a** *k*means, $k=2$, **b** *k*means, $k=5$, **c** EM

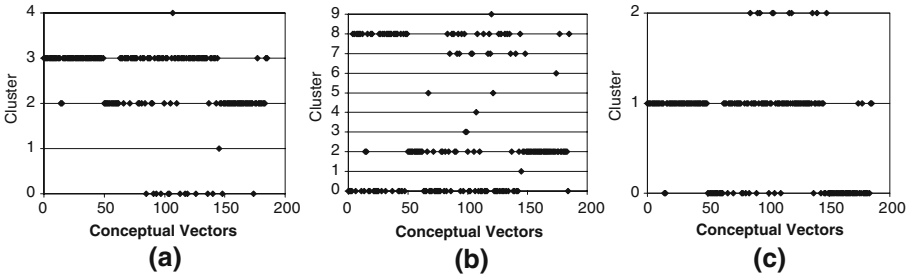

Fig. 6 Cluster assignments—spam data. **a** *k*means, $k=5$, **b** *k*means, $k=10$, **c** EM

By studying the elist dataset (see Table 1), we observe that there exist two different (recurring) contexts. In the first one (Concept 0) only medicine articles are considered interesting. In the second one (Concept 1) only space and baseball articles are considered interesting. Concept 0 occurs in periods [0, 300], [600–900], [1,200–1,500] whereas Concept 1 in [300–600] and [900–1,200]. What one would expect from the clusterer is to group vectors that belong to periods of Concept 0 into the same cluster and vectors that belong to periods of Concept 1 into another cluster.

We applied the *k*means and the expectation-maximization (EM) [31] clustering algorithm on the conceptual vectors of the elist dataset.

Cluster assignments are visualized in Fig. 5. For *k*means with $k = 2$ we observe that vectors 0 to 5 (representing period [0–300]), 12–17 ([600–900]) and 24–29 ([1,200–1,500]) are assigned to the same cluster (Cluster 0). The rest of the vectors are assigned to Cluster 1. Hence, under these settings, we observe that the proposed representation creates items (that correspond to batches) that can be grouped into the right concepts. For $k = 5$ (Fig. 5b), *k*means is forced to assign some vectors into other clusters but still we have two dominating, clusters (Cluster 1 and Cluster 3) that represent the correct recurring contexts. Finally EM (Fig. 5c) provides similar assignments failing only on two vectors.

For the spam dataset (Fig. 6), there is no prior knowledge about some recurring contexts. However, all three clustering algorithms create three dominating clusters. Smaller clusters may be representing batches of outlier or noisy messages. As we discuss in the following sections, this batch organization is beneficial for the classifiers.

7.2 Evaluating the proposed framework

In this section, we experimentally evaluate the proposed framework. We begin by introducing the methods that are involved in the experiments. Next, we discuss on the implementation details of all methods and finally present and discuss the experimental outcome.

Methods

The instantiation of the proposed framework is compared against the following stream learning methods.

- *Simple incremental classifier (SIC)* In this scheme, a single incremental learning algorithm first classifies an instance and then updates its model.
- *Batch learner (BL)* A classifier that is re-trained every b instances from the last b instances. In this case there is no requirement for the learner to be incremental, a fact that broadens the choice of classifier.
- *Moving window (MW)* A machine learning algorithm that it always models the last w instances. The algorithm must implement an "*update*" function in order to aggregate a new example and a "*forget*" function, in order to retract from its model the first example of the window.
- *Weighted examples (WE)* This method consists of an incremental base classifier that supports weighted learning. Bigger weights are assigned to more recent examples in order to force the classifier to focus on new concepts and gradually forget the old ones. The learner should be incremental and support weighted learning.
- *Dynamic weighted majority (DWM)* This is our implementation of the algorithm presented in [13]. DWM maintains a set of learners each one associated with a weight w_i. The decision of the ensemble is obtained by weighted majority voting. Every time a learner makes a mistake, its weight is reduced by a factor β. If the weight falls below a threshold θ then the learner is removed from the ensemble. Every time the ensemble makes a mistake, a new learner is introduced with a weight equal to 1. After classification of an instance the true label is considered known and all learners are updated. Parameter p expresses the frequency that the algorithm removes or adds learners and updates weights.

Experimental setup

All methodologies and algorithms are implemented on top of the Weka [32] API. All methods, with the exception of BL use an updateable version of the naive Bayes classifier [33] as base learner. Naive Bayes was chosen mainly because it is inherently incremental (class priors and feature probability distributions can be calculated incrementally) and has been widely used in text classification tasks in general [34,35] and in email classification in particular [36,37], which is the application under study in this paper. Since BL can operate without an incremental algorithm we additionally exploited support vector machines (SVM) [38] which are considered more accurate classifiers in text classification [39,40] and also used in email classification [41]. The SVM was set up with default Weka parameter settings (SMO implementation, linear kernel, $C = 1$). Note that linear SVMs have proven to perform at least as well as non-linear SVMs in text classification problems [42,43]. BL was executed with three different batch sizes (50, 100, 200) for both classifiers. Similarly, the MW framework was run using three different window sizes (50, 100, 200). Concerning the WE method, we conducted preliminary experiments and found out that an appropriate way to update

```
   Input: x_i: An infinite stream of points, θ: sensitivity, c_max: maximum
          number of clusters, m_j: number of items assigned to cluster c_j.
   Output: Cluster assignment J for every x_i.
 1 begin
 2     c_1 ← x_1, m_1 ← 1, n ← 1;
 3     for i ← 2 to infinity do
 4         J ← argmin_j(distance(x_i, c_j)) ;
 5         if distance(x_i, c_J)) < θ or n > C_max then
 6             m_J ← m_J + 1;
 7             c_J ← ((m_J − 1)/m_J)c_J + (1/m_J)x_i;
 8         else
 9             n ← n + 1;
10             create new cluster c_n ← x_i;
11             m_n ← 1;
12 end
```

Fig. 7 The leader–follower clustering algorithm implemented in CCP

weights is $w(n) = w(n-1) + n^2$ for the elist dataset and $w(n) = w(n-1) + 1$ for the spam dataset, where $w(n)$ is the weight of the n-th example. Finally, for DWM we used the settings proposed in [13] (i.e. $\beta = 0.5, \theta = 0.01, p = 1$).

Concerning the batch size of CCP, preliminary experiments showed that a batch size around 50 instances is appropriate. Larger batches invalidate the locality assumption, whereas smaller batches do not suffice for calculating the summary probabilistic statistics.

Our instantiation of the CCP framework includes the mapping function discussed in Sect. 4.2, a modified version of the leader–follower algorithm described in [28] as the clustering component and an incremental Naive Bayes classifier. Note that the leader–follower is only a baseline clusterer. More sophisticated solutions can be retrieved from the literature (see Sect. 2.2).

In brief, the leader–follower clusterer is an online version of the well known k-means algorithm and works as follows. The first instance is assigned to the first cluster. From then, every time a new instance arrives, the algorithm calculates the distances between the instance and every cluster (i.e. each cluster's centroid). If this distance exceeds a threshold θ then a new cluster is created having this instance as its first member. If the distance is smaller than θ, then the instance is assigned to the closest cluster and its center is updated. There is a stability parameter h that determines in what degree the cluster's center will move towards the new instance.

In our implementation of the leader–follower, we have made two modifications to the algorithm presented in [28]. The pseudocode can be seen in Fig. 7. First, in order to eliminate the stability parameter h used in [28] to calculate new centroids, we set equal weights for all instances by using the number (m_j) of items assigned to each cluster (line 7). This modification not only simplifies the algorithm (in terms of parameters need to be tuned) but also assures that stable clusters will not be much affected by new distant members.

The second modification is associated with the memory constraints of the data stream classification problem. The user may enter a maximum allowed number ($c_{\max}$) of clusters (i.e. concepts and corresponding classifiers). When the limit is reached, the new item

(conceptual vector) is incorporated into the nearest cluster. The purpose of $c_{\max}$ is to limit the number of classifiers maintained in memory.

Parameter θ will define how sensitive the framework will be to concept drift. This is an unavoidable parameter that exists in various forms in many drift detection mechanisms [2]. The weighted scheme of WE, the batch size of BL, the window size of MW, and the β and θ parameters of DWM can be considered as such parameters. Obviously as θ decreases, number of clusters (i.e. classifiers) will increase. However, in our case, $c_{\max}$ will prevent memory overflows independently of θ. After preliminary experimentation we set $\theta = 4$ for the elist dataset and $\theta = 2.5$ for the spam dataset and $c_{\max} = 10$ for both problems. In the next section we examine how variation of θ affects the number of classifiers and the system's performance.

The experiments include a benchmark version of our framework (dubbed Oracle), where the system knows beforehand the cluster (context) that the next batch belongs to. This is to approximate the upper bound of our framework's performance (i.e. with perfect clustering and minimum batch size).

Every method, given an unlabeled instance makes a prediction about the class of the instance and the method's predictive accuracy is updated. Subsequently, the true label is considered known and every method can use this extra example to update its knowledge. Note, however, that CCP only takes advantage of the updates after a batch is finished.

For each method we calculated classification accuracy, precision, recall and run time. Because the datasets used in this paper correspond to filtering problems, we consider the class *spam* (for the spam dataset) and *junk* (for the elist dataset) as the positive classes for the calculation of precision and recall.

All experiments were executed on an Intel Pentium 4 CPU at 2.0 GHz with 2.0 GB of RAM.

Results and Discussion

Table 2 presents the predictive accuracy, precision, recall and runtime (in seconds) of all methods in the elist dataset. We observe that even a basic instantiation of the CCP framework outperforms all methods. As it is shown later, CCP correctly clusters batches of examples and builds concept-specific classifiers that are evoked when necessary and updated with proper examples. The second best performance is achieved by the moving window framework. The main advantage of MW is that it always models the latest window of instances and therefore, although lacking a drift detection mechanism, is able to adapt to concept drift quickly. Batch learner is not performing as well as moving windows due to the classifier only updating every b instances. Support vector machines certainly comprise an improvement but still do not match the performance of MW with a simple naive Bayes classifier. weighted examples considers all instances for building the model but it also lacks of a detection mechanism and therefore adapts ineffectively to drift. However, the improvement over the simple incremental naive Bayes is substantial, underlining the utility of the weighted learning scheme. Regarding the optimal batch/window size of BL/MW we notice that in this dataset, small sizes of b/w are more appropriate. In fact, the best performance of BL was achieved when $b = 50$ (accuracy 0.704) and $w = 65$ (0.751). DWM is not designed to cope up with recurring contexts and therefore performs poorly on this dataset. Finally and most importantly, the performance of Oracle verifies the suitability of the context-specific classifiers for problems with concept drift and recurring contexts and encourage further investigation towards this direction. Concerning run time, as expected, simpler methods like SIC or WE require less time for classifying the stream. However, no method proved to be significantly faster.

Table 2 Accuracy, precision, recall and run time of all methods in the elist dataset

Method	Base classifier	Accuracy	Precision	Recall	Time
Simple incremental	NB	0.545	0.612	0.435	1.994
Batch learner ($b = 50$)	NB	0.704	0.725	0.718	1.885
Batch learner ($b = 50$)	SVM	0.714	0.737	0.721	2.088
Batch learner ($b = 100$)	NB	0.607	0.642	0.607	1.822
Batch learner ($b = 100$)	SVM	0.642	0.674	0.648	1.880
Batch learner ($b = 200$)	NB	0.425	0.463	0.426	1.793
Batch learner ($b = 200$)	SVM	0.465	0.505	0.438	2.093
Moving window ($w = 50$)	NB	0.725	0.742	0.741	2.609
Moving window ($w = 100$)	NB	0.747	0.779	0.732	2.557
Moving window ($w = 200$)	NB	0.662	0.694	0.653	2.601
Weighted examples	NB	0.669	0.685	0.702	1.970
Dynamic weighted majority	NB	0.438	0.470	0.425	5.265
CCP (leader–follower)	NB	0.775	0.797	0.776	3.034
CCP (Oracle)	NB	0.875	0.883	0.869	3.070

Table 3 Accuracy, precision, recall and run time of all methods in the spam dataset

Method	Base classifier	Accuracy	Precision	Recall	Time
Simple incremental	NB	0.912	0.809	0.856	4.896
Batch learner ($b = 50$)	NB	0.880	0.851	0.646	4.531
Batch learner ($b = 50$)	SVM	0.881	0.820	0.685	8.083
Batch learner ($b = 100$)	NB	0.887	0.845	0.685	4.495
Batch learner ($b = 100$)	SVM	0.899	0.848	0.738	8.124
Batch learner ($b = 200$)	NB	0.889	0.818	0.728	4.463
Batch learner ($b = 200$)	SVM	0.914	0.865	0.790	9.137
Moving window ($w = 50$)	NB	0.915	0.911	0.742	5.102
Moving window ($w = 100$)	NB	0.919	0.902	0.770	5.221
Moving window ($w = 200$)	NB	0.921	0.881	0.801	5.344
Weighted examples	NB	0.921	0.836	0.864	4.786
Dynamic weighted majority	NB	0.918	0.848	0.831	4.753
CCP (leader–follower)	NB	0.923	0.857	0.839	8.750

In the spam filtering dataset (Table 3) there is no artificially inserted concept drift. However, the difference in performance between weighted examples and simple incremental classifier as well a previous study of ours [30] suggests the existence of gradual concept drift. In this dataset, we would expect CCP to perform badly because it uses many classifiers with smaller training sets. However, it seems that the online organization of batches was beneficial to the classification performance as the clusterer grouped similar batches and built cluster-specific classifiers. In addition, clustering conceptual vectors will probably isolate noisy batches of examples that would negatively affect a simple incremental classifier. Concerning the optimal batch/window size of BL/MW we notice that in this dataset, larger sizes of b/w are more

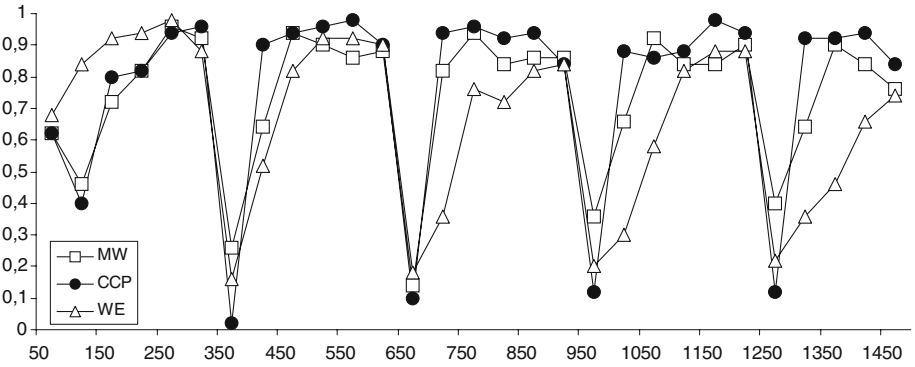

Fig. 8 Average accuracy over 50 instances for WE, MW and CCP. Drift time points at: 300, 600, 900 and 1,200 instances

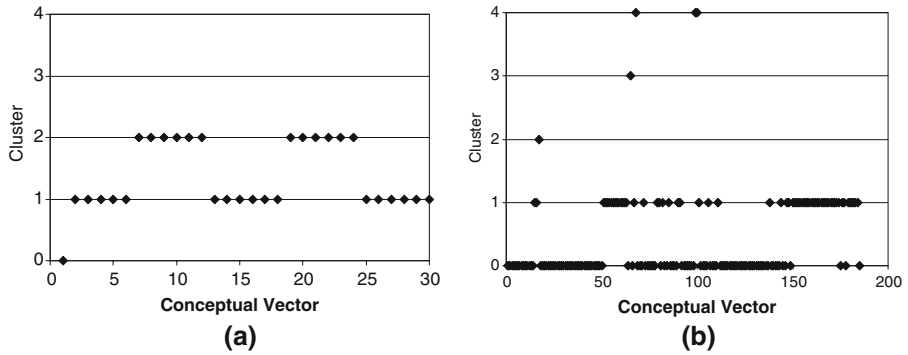

Fig. 9 On-line cluster assignments of the stream of conceptual vectors by the leader–follower clusterer. **a** Elist dataset, **b** Spam dataset

appropriate. In fact the best performance of BL was achieved when $b = 145$ (0.900) and of MW at $w = 150$ (0.922). The best overall performance in this dataset is achieved by CCP followed by WE, MW and DWM. Considering time, as in the previous dataset, simpler methods like SIC and WE require less time to classify the stream.

Figure 8 presents the average accuracy of CCP, MW and WE over 50 instances for the elist dataset. Notice the sudden drops of all methods at drift time points (after 300, 600, 900 and 1,200 instances). However, CCP manages to recover faster in all cases exploiting and updating older models.

Figure 9 presents the cluster assignments of the leader–follower algorithm that led to the results shown at Table 2. In the elist dataset, we observe that the clusterer made the correct assignments with the exception of the first instance (see in comparison with Fig. 5a). In the spam data, the clusterer correctly identified the two larger clusters (see in comparison with Fig. 6a).

Finally, Fig. 10 depicts the variation of number of classifiers (Fig. 10a) and predictive accuracy (Fig. 10b) with respect to the sensitivity parameter θ. In this case we set $c_{\max} = 100$ in order not to limit the maximum number of clusters. As expected (see "Discussion" in the previous section), the number of classifiers decreases as the sensitivity parameter increases.

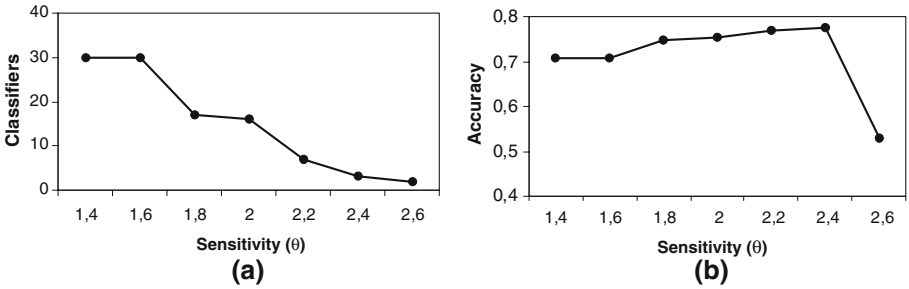

Fig. 10 Number of classifiers (**a**) and accuracy (**b**) with respect to the sensitivity parameter θ in the elist dataset

Naturally, better performance is achieved when the number of classifiers created is close to number of concepts in the stream (Fig. 10b).

8 Conclusions and future work

We proposed a framework that dynamically creates and maintains an ensemble of classifiers for streaming data that are characterized by concept drift and especially by recurring contexts. After the transformation of data into a new conceptual representation model, incremental clustering was exploited in order to discover concepts in the stream and maintain a classifier for each one of these concepts. In order to evaluate the framework we developed an instantiation and constructed two real world email streams with concept drift. For the sake of comparison, we included in our experiments five methods for dealing with varying concepts in data streams. Experimental results show that the instantiation of the CCP framework successfully identifies concepts in the stream outperforming the rest of the methods. This experimental outcome with the addition of the performance of a benchmark version of our framework (dubbed Oracle) supports the intuition of concept specific classifiers.

In the future, we intend to investigate other approaches for identifying and organizing concepts in the stream. Instead of clustering batches of examples, an alternative approach would be to cluster the classifiers. Clustering classifiers has been already discussed in the literature [44] but has not yet been applied to the stream classification problem. Finally, it is in our immediate plans to incorporate an instantiation of CCP into our adaptive news aggregator, named Personews [45], which currently uses a dynamic learning algorithm without the ability to identify concepts and recurring themes.

Acknowledgments We would like to thank the anonymous reviewers for their valuable comments. This work was partially supported by a PENED program (EPAN M.8.3.1, No. 03Δ73), jointly funded by the European Union and the Greek Government (General Secretariat of Research and Technology/GSRT).

References

1. Aggarwal C (ed) (2007) Data streams: models and algorithms. Springer, Heidelberg
2. Tsymbal A (2004) The problem of concept drift: definitions and related work. Technical report, Department of Computer Science Trinity College
3. Widmer G, Kubat M (1996) Learning in the presense of concept drift and hidden contexts. Mach Learn 23(1):69–101
4. Harries MB, Sammut C, Horn K (1998) Extracting hidden context. Mach Learn 32(2):101–126

5. Forman G (2006) Tackling concept drift by temporal inductive transfer. In: SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval. ACM Press, New York, pp 252–259
6. Gaber M, Zaslavsky A, Krishnaswamy S (2007) A survey of classification methods in data streams. In: Aggarwal C (ed) Data streams, models and algorithms. Springer, Heidelberg, pp 39–59
7. Barbará D (2002) Requirements for clustering data streams. SIGKDD Explor 3(2):23–27
8. Cheng J, Ke Y, Ng W (2008) A survey on algorithms for mining frequent itemsets over data streams. Knowl Inform Syst 16(1):1–27
9. Kolter J, Maloof M (2003) Dynamic weighted majority: a new ensemble method for tracking concept drift. In: Proceedings of the Third IEEE international conference on data mining. IEEE Press, Los Alamitos, pp 123–130
10. Kolter JZ, Maloof MA (2005) Using additive expert ensembles to cope with concept drift. In: ICML '05: Proceedings of the 22nd international conference on machine learning. ACM Press, New York, pp 449–456
11. Wenerstrom B, Giraud-Carrier C (2006) Temporal data mining in dynamic feature spaces. IEEE Computer Society, Los Alamitos, pp 1141–1145
12. Gama J, Medas P, Castillo G, Rodrigues PP (2004) Learning with drift detection. In: Bazzan ALC, Labidi S (eds) Advances in artificial intelligence. Proceedings of the 17th Brazilian symposium on artificial intelligence (SBIA 2004). Lecture notes in artificial intelligence, vol 3171. Springer, Brazil, pp 286–295
13. Kolter JZ, Maloof MA (2007) Dynamic weighted majority: an ensemble method for drifting concepts. J Mach Learn Res 8:2755–2790
14. Wang H, Fan W, Yu PS, Han J (2003) Mining concept-drifting data streams using ensembles classifiers. In: 9th ACM SIGKDD international conference on knowledge discovery and data mining. ACM Press, Washington, DC, pp 226–235
15. Martin Scholz RK (2007) Boosting classifiers for drifting concepts. Intell Data Anal, Spec Issue Knowl Discovery from Data Streams 11(1):3–28
16. Zhou A, Cao F, Qian W, Jin C (2008) Tracking clusters in evolving data streams over sliding windows. Knowl Inform Syst 15(2):181–214
17. O'Callaghan L, Mishra N, Meyerson A, Guha S, Motwani R (2002) High-performance clustering of streams and large data sets. In: ICDE 2002
18. Aggarwal CC, Han J, Wang J, Yu PS (2004) A framework for projected clustering of high dimensional data streams. In: VLDB '04: Proceedings of the 30th international conference on very large data bases, VLDB Endowment, pp 852–863
19. Zhang T, Ramakrishnan R, Livny M (1996) BIRCH: an efficient data clustering method for very large databases. ACM SIGMOD Rec 25(2):103–114
20. Klinkenberg R, Joachims T (2000) Detecting concept drift with support vector machines. In: ICML '00: Proceedings of the 17th international conference on machine learning. Morgan Kaufmann, San Francisco, pp 487–494
21. Klinkenberg R (2004) Learning drifting concepts: example selection vs. example weighting. Intell Data Anal 8(3):200–281
22. Fan W (2004) Systematic data selection to mine concept-drifting data streams. In: KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM Press, New York, pp 128–137
23. Delany SJ, Padraig Cunningham ATLC (2005) A case-based technique for tracking concept drift in spam filtering. Knowl Based Syst 18(4–5):187–195
24. Street WN, Kim Y (2001) A streaming ensemble algorithm (SEA) for large-scale classification. In: 7th ACM SIGKDD international conference on knowledge discovery in data mining. ACM Press, pp 277–382
25. Zhu X, Wu X, Yang Y (2006) Effective classification of noisy data streams with attribute-oriented dynamic classifier selection. Knowl Inform Syst 9(3):339–363
26. Spinosa EJ, Carvahlo Ad, Gama J (2007) OLINDDA: a cluster-based approach for detecting novelty and concept drift in data streams. In: 22nd annual acm symposium on applied computing. ACM Press, pp 448–452
27. Hulten G, Spence L, Domingos P (2001) Mining time-changing data streams. In: KDD '01: 7th ACM SIGKDD International conference on knowledge discovery and data mining. ACM Press, pp 97–106
28. Duda RO, Hart PE, Stork DG (2000) Pattern classification. Wiley, New York
29. Asuncion A, Newman D (2007) UCI machine learning repository
30. Katakis I, Tsoumakas G, Vlahavas I (2006) Dynamic feature space and incremental feature selection for the classification of textual data streams. In: ECML/PKDD-2006 international workshop on knowledge discovery from data stream, pp 107–116

31. Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc Ser B (Methodological) 39(1):1–38
32. Witten I, Frank E (2005) Data mining: practical machine learning tools and techniques. 2nd edn, San Francisco
33. John GH, Langley P (1995) Estimating continuous distributions in Bayesian classifiers. In: UAI '95: Proceedings of the 11th annual conference on uncertainty in artificial intelligence. Morgan Kaufman, Montreal, pp 338–345
34. Domingos P, Pazzani MJ (1997) On the optimality of the simple bayesian classifier under zero-one loss. Mach Learn 29(2–3):103–130
35. Sebastiani F (2002) Machine learning in automated text categorization. ACM Comput Surv 34(1):1–47
36. Sahami M, Dumais S, Heckerman D, Horvitz E (1998) A bayesian approach to filtering junk e-mail. In: Learning for text categorization: papers from the 1998 Workshop, Madison, Wisconsin, AAAI Technical Report WS-98-05
37. Rennie J (2000) ifile: an application of machine learning to e-mail filtering. In: KDD-2000 workshop on text mining
38. Vapnik V (1995) The nature of statistical learning theory. Springer, Heidelberg
39. Joachims T (1998) Text categorization with support vector machines: learning with many relevant features. In: Nédellec C, Rouveirol C (eds) Proceedings of ECML-98, 10th European conference on machine learning. Number 1398. Springer, Heidelberg, pp 137–142
40. Peng T, Zuo W, He F (2008) SVM based adaptive learning method for text classification from positive and unlabeled documents. Knowl Inform Syst 16(3):281–301
41. Klimt B, Yang Y (2004) The enron corpus: a new dataset for email classification research. In: ECML 2004, 15th European conference on machine learning. Springer, Pisa, pp 217–226
42. Rennie JD, Rifkn R (2001) Improving multiclass text classification with the support vector machine. Technical Report AIM-2001-026, Massachusetts Institute of Technology
43. Yang Y, Liu X (1999) A re-examination of text categorization methods. In: SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. ACM Press, New York, pp 42–49
44. Tsoumakas G, Angelis L, Vlahavas I (2004) Clustering classifiers for knowledge discovery from physically distributed databases. Data Knowl Eng 49(3):223–242
45. Katakis I, Tsoumakas G, Banos E, Bassiliades N, Vlahavas I An adaptive personalized news dissemination system. J Intell Inform Syst 32:191–212

Author Biographies

Ioannis Katakis is a Ph.D. student at the Department of Informatics of the Aristotle University of Thessaloniki. He received his B.Sc. in 2004 and an M.Sc. in Information Systems in 2007 from the same department. His research interests include machine learning and data mining and in particular text stream classification and multi-label text classification. He is a member of the Machine Learning and Knowledge Discovery group (MLKD, http://mlkd.csd.auth.gr).

Dr. Grigorios Tsoumakas is a Lecturer at the Department of Informatics of the Aristotle University of Thessaloniki (AUTH). He received a B.Sc. in Informatics from AUTH in 1999, an M.Sc. in Artificial Intelligence from the University of Edinburgh in 2000 and a Ph.D. in Informatics from AUTH in 2005. Grigorios Tsoumakas has co-authored more than 40 articles in international journals and conferences. His research interests include machine learning, knowledge discovery and data mining.

Ioannis Vlahavas is a professor at the Department of Informatics at the Aristotle University of Thessaloniki. He received his Ph.D. degree in Logic Programming Systems from the same University in 1988. During the first half of 1997 he has been a visiting scholar at the Department of CS at Purdue University. He specializes in logic programming, knowledge based and AI systems and he has published over 160 papers, 17 book chapters and co-authored 8 books in these areas. He teaches logic programming, AI, expert systems, and DSS. He has been involved in more than 20 research and development projects, leading most of them. He was the chairman of the second Hellenic Conference on AI and the host of the second International Summer School on AI Planning. He is leading the Logic Programming and Intelligent Systems Group (LPIS Group, http://lpis.csd.auth.gr) (more information at http://www.csd.auth.gr/~vlahavas).