

Using a classifier pool in accuracy based tracking of recurring concepts in data stream classification

Mohammad Javad Hosseini · Zahra Ahmadi ·
Hamid Beigy

Received: 7 February 2012 / Accepted: 31 August 2012 / Published online: 9 October 2012
© Springer-Verlag 2012

Abstract Data streams have some unique properties which make them applicable in precise modeling of many real data mining applications. The most challenging property of data streams is the occurrence of “concept drift”. Recurring concepts is a type of concept drift which can be seen in most of real world problems. Detecting recurring concepts makes it possible to exploit previous knowledge obtained in the learning process. This leads to quick adaptation of the learner whenever a concept reappears. In this paper, we propose a learning algorithm called Pool and Accuracy based Stream Classification with some variations, which takes the advantage of maintaining a pool of classifiers to track recurring concepts. Each classifier is used to describe an existing concept. Consecutive batches of instances are first classified by the pool of classifiers. Two approaches are presented for this task: active classifier and weighted classifiers methods. Then the true labels are revealed and the pool is updated at the end of the batch. Updating the pool is done using one of the following methods: exact Bayesian, Bayesian and Heuristic. As the algorithm may assign multiple classifiers to a single concept, a classifier merging process is used to resolve this problem. Experimental results on real and artificial datasets show the effectiveness of weighted classifiers method while dealing with sudden concept drifting datasets. In

addition, the proposed updating methods outperform the existing algorithms in datasets with arbitrary attributes. Finally some performed experiments represent superiority of using merging process in large datasets.

Keywords Recurring concepts · Concept drift · Stream mining · Ensemble learning

1 Introduction

In many real problems there are large volumes of data available and we need to deal with huge amount of data. The data may change during the time and it may not be possible to store all the data, so a new field of data mining called stream mining is presented. Stream mining is processing of continuous and unlimited stream of data. It could be defined more precisely as a stochastic process of continuous and independent events and has the following properties (Tsymbol 2004):

- The amount of events is huge and could not be stored completely, so a forgetting mechanism is needed to ignore ineffective events (or data).
- Event processing should be online and the processing algorithm complexity should be low.
- Each event is drawn from a feature space ($\Omega_X = \Omega_{X_1} \times \dots \times \Omega_{X_n} \subset \mathbb{R}^n$, where X_i is the i th feature and Ω_{X_i} is the space corresponding with feature X_i and an output space (Ω_C), each drawn from a distribution which may change over the time. This change is called concept drift and it is the change in the joint probability of feature space (X) and/or output space (C), which could be written as:

$$P(X, C) = P(C|X) \times P(X). \quad (1)$$

M. J. Hosseini (✉) · Z. Ahmadi · H. Beigy
Department of Computer Engineering,
Sharif University of Technology, Tehran, Iran
e-mail: mjhosseini@ce.sharif.edu

Z. Ahmadi
e-mail: z_ahmadi@ce.sharif.edu

H. Beigy
e-mail: beigy@sharif.edu

The change in the feature or input space ($P(X)$) is called virtual concept drift and it may occur when the training instances are skewed. When the change takes place in the target function ($P(C|X)$), it is named real concept drift (Tsybmal 2004).

Data streams have many real applications such as fraud detection, spam filtering, intrusion detection, text classification, economical prediction and user interest forecasting; to mention a few. For example, in spam filtering problem, it is more realistic to assume that in the training phase, all emails are not available. So in stream mining approach, we assume that emails arrive through the time and are not available at first. Each email is treated first as a test example and after some time when its label (spam or ham) is revealed by an expert, it is treated as a training example and used to update the classifier. In fact the training and the test phases are mixed. Instances could be thought to arrive in batch (or window) or single.

One of the main challenges in data stream learning is the detection of drift from noise. As data is not stationary and drift may occur, noise may be recognized as drift. It should be mentioned that because of non-stationary nature of data stream, Independent Identically Distribution (I.I.D) condition is not valid, but it is rational to assume that I.I.D condition holds in small size windows of data.

We can categorize the concept drift according to the way changes happen in time into the following four groups (Zliobaite 2010):

- Sudden: the simplest way of drift where the distribution of events are changed at a specific time.
- Gradual: in gradual drift, before completing the distribution change, there is a non-deterministic stage where the data drawn from both of distributions and the probability of old distribution decreases and the probability of new distribution increases during the time.
- Incremental: a generalized form of gradual drift in which during the non-deterministic period of distribution change, there are more than two distributions to draw data from and the difference between the distributions are small.
- Recurring: if the distribution of data reoccurs after some time it is said recurring concepts.

Usually gradual and incremental drifts are assorted into one category and are called gradual drift. There have been extensive studies on the detection and learning of sudden and gradual concept drift in recent decades (Minku 2012; Elwell and Polikar 2011; Baena-García et al. 2006; Bifet 2009; Bifet et al. 2010; Gama and Castillo 2006; Gao et al. 2008; Garnett 2010; Ikononovska 2011; Kolter and Maloof 2007; Kuncheva and Žliobaitė 2009; Nishida 2008; Zliobaite 2010). Though recurring concepts were not

considered much until recent years (Gama and Kosina 2009; Gomes et al. 2011; Katakis et al. 2009; Lazarescu 2005; Morshedlou and Barforoush 2009) and now are assumed as a challenging problem in data streams. As the learning process is continuous in data streams, the old unused concepts (distributions where data is drawn) which were learned by learner can be forgotten and new concepts are learned. After some sufficient time when the old concept reappears, the learner may label its instances incorrectly and the learning process should be repeated. If we could use the previously learned knowledge in the classification algorithm, this issue can be resolved appropriately. This is done by using a pool of classifiers, which is an idea presented in the previous researches (Gama and Kosina 2009; Gomes et al. 2011; Katakis et al. 2009).

In this paper, a new ensemble learning algorithm called Pool and Accuracy based Stream Classification (PASC) is presented. This algorithm tries to exploit recurring concepts and is based on a pool of classifiers in which one classifier is used for each concept. The pool of classifiers is updated iteratively while receiving consecutive batches of data. A group of classifiers is maintained in the pool and each of them describes one of the existing concepts. In our previous algorithm (Hosseini et al. 2011), a maximum number of classifiers in the pool was defined as a parameter. After receiving a batch of instances, a new classifier could be added to the pool. However, if the similarity of the recent batch and one of the classifiers of the pool was high enough, the batch was used to update the most similar classifier. After reaching the limit of classifiers, the batch was used to update the most similar classifier without checking that the batch was similar enough to that classifier or not. In this paper, we present a merging procedure to manage the pool and not to update an existing classifier with a batch of data which is not similar enough to it. In this case, the possibility of merging a pair of existing classifiers should be checked. The benefit of merging procedure is apparent as one concept may appear in many batches of instances and so there may be many classifiers in the pool representing that concept. Due to space limitation, we could merge the knowledge of these classifiers and release a free space for a new classifier and also make the concept and corresponding classifier, more informative. If the merging procedure cannot be done, the most similar classifier to the current batch will be updated.

When a window of data arrives, it is first assumed as a test data and is classified by the learner. The classification is done by the classifiers available in the pool in an effective way. The base of classification algorithm is similar to (Katakis et al. 2009) but there are major differences. One of the contributions of this paper is the methods of classification which are active and weighted classifiers methods. After revealing true labels of instances, they are

used to update one of the existing classifiers of the pool or there may be the need to add a new classifier to the pool. Decision on adding a new classifier or updating one of the existing ones is done by some examinations on the recent window and the pool. Other contribution of paper is new methods to update the pool which are exact Bayesian, Bayesian and Heuristic methods. In exact Bayesian there are more relax assumptions and I.I.D condition is not assumed to be held. This case is more general than the other Bayesian method in which this assumption is exploited to reduce complexity.

The proposed methods are compared with the promising and state of the art methods presented in the recurring concepts. The experimental results show the effectiveness of proposed algorithm in terms of accuracy, especially in stream of data with sudden drifts, arbitrary attributes or large size. In addition, the parameter tuning of the proposed methods are more straightforward in comparison with previous methods.

The rest of this paper is organized as follows: in the Sect. 2, the related work and previous researches in the context of data stream are discussed. In Sect. 3, the proposed algorithm is presented. Section 4 contains the experimental evaluations of the proposed algorithm and makes comparisons to some previous methods. Section 5 concludes the paper and discusses some of future developments.

2 Related works

Stream mining and handling concept drift have received much attention in recent decades. First algorithms which were capable of controlling concept drift, were STAGGER (Schlimmer et al. 1986), FLORA (Widmer and Kubat 1996) and IB3 (Aha et al. 1991) which used simple rule based or k-nearest neighbor learning algorithms. Passing through time, more researches have been done on data streams and wider range of problems has been considered. An ideal concept drift learning algorithm should support sudden, gradual and recurring drifts and should be robust to noise. It also should not have many parameters to be tuned for a specific dataset.

As discussed previously, there are different types of concept drifts. Many algorithms have been presented to detect sudden drift (Nishida 2008; Widmer and Kubat 1996; Bifet and Gavaldà 2007; Castillo 2006; Gama et al. 2004, 2006). They usually track the performance of the algorithm in the recent window and when there is a significant degradation on the performance, the occurrence of drift is alarmed. Gradual drifts are usually learned implicitly by updating the learner (Widmer and Kubat 1996; Klinkenberg 2004). The learner itself could be single

(Kolter and Maloof 2007; Lazarescu et al. 2005; Klinkenberg 2004; Hulten et al. 2001; Domingos et al. 2000) or ensemble (Lazarescu 2005; Littlestone 1988; Street et al. 2001; Stanley 2003; Wang et al. 2003; Gao et al. 2007; Kolter et al. 2005; Scholz and Klinkenberg 2005; Widmer and Kubat 1998; Freund and Schapire 1997; Littlestone and Warmuth 1994; Woolam et al. 2009). Each time the learner may receive an instance or a window of instances. Window based algorithms could be of fixed or variable size (Widmer and Kubat 1996). Most of algorithms use windows of data because of their more robustness to noise. In window based algorithms, variable window size approaches (Kuncheva and Žliobaitė 2009; Bifet and Gavaldà 2007) will change and adapt the window size in stationary or non-stationary environments. When data are drawn from one concept (which means stationary environments) the window size is increased and after a concept drift, the window size is set to a minimum size. However, most of the approaches use a fixed size of window.

However, recurring concepts detection is a new and challenging problem which is considered recently (Gama and Kosina 2009; Gomes et al. 2011; Katakis et al. 2009; Lazarescu 2005; Morshedlou and Barforoush 2009). Due to forgetting mechanism in the learning of data streams, if a concept reappears after some long time, the learner may classify it incorrectly. This makes the system impractical in real world problems. The algorithms supporting recurring concepts, try to extract concepts from received instances. Then they maintain a pool of classifiers. When a new instance is received, its similarity to the concepts of the pool is measured and an available model is selected or a new one is created. We will explain stream classification algorithms which support recurring concepts, in the following:

The first algorithm (Ramamurthy and Bhatnagar 2007), consists of an ensemble classifier where each classifier is built on a window. To select classifiers for the ensemble, the algorithm chooses relevant ones. As no classifier is deleted, the algorithm supports recurring concepts.

The other algorithm (Morshedlou and Barforoush 2009), uses the information of mean and standard deviation used in the conceptual features of numeric attributes (Hulten et al. 2001). The approach uses a pro-active behavior which means that next concept probability is calculated conditional to the current concept. If the concept is probable (its probability is more than a threshold), it will be added to a buffer. If a drift is detected and the algorithm decides to behave proactively, the first concept from the buffer is selected. If it matches the recent batch, it is updated by batch instances. Else it could select the next concept in the buffer or if it decides to behave reactively, a new classifier will be created on the recent batch. To decide on reactive or proactive behavior, a heuristic method is pursued.

Selecting suitable probability threshold and proactive or reactive behavior are very time consuming. Meanwhile, this algorithm supports only the datasets with numeric features not nominal.

One other research (Padovitz et al. 2004) inspires context space model to extract concepts for each classifier (Gomes et al. 2011). N -tuple of form $R = (a_1^R, a_2^R, \dots, a_N^R)$ is a context space, where a_i^R is the acceptable regions of feature a_i . The classifier and its corresponding context space are maintained in the pool. This approach is similar to our approach, as it extracts the context of a batch of data and trains a classifier on it; however, the representation of context is different from our concept representation. It also, uses an explicit drift detection method to detect stable concepts.

One meta-learning approach is presented recently (Gama and Kosina 2009). If a drift is alarmed, performance is determined by meta-learners. If it is more than a predefined threshold, the base learner will be used to label the instance. Here also a pool is used to keep all base and meta learners.

One of the most promising frameworks presented in the context of recurring concepts, extracts a conceptual vector from each window (Katakis et al. 2009). Assume that instances of a labeled batch are presented by:

$$B_L = \{x_{L(k)}, x_{L(k+1)}, \dots, x_{L(k+b-1)}\}, \quad (2)$$

where $x_{L(k+i)}$ is the $(i + 1)$ th instance of the k th batch. The conceptual vector $Z = (Z_1, Z_2, \dots, Z_n)$ of each batch is constructed using a transformation function. In the conceptual vector, each of features (Z_i) is calculated from:

$$z_i = \begin{cases} \{P(f_i = v|c_j) : i = 1..n, j = 1..m, v, \in V_i\} & \text{if } f_i \text{ is nominal} \\ \{\mu_{ij}, \sigma_{ij} : j = 1..m\} & \text{if } f_i \text{ is numeric} \end{cases} \quad (3)$$

where f_i is the i th feature, V_i is the set of possible values for the nominal feature i , μ_{ij} and σ_{ij} are the mean and standard deviation of j th class for feature i . Each concept in the pool has a corresponding classifier which is updated by time. Then a clustering algorithm is used to detect recurring concepts. To do clustering, conceptual vectors are compared via Euclidean distance measure:

$$\begin{aligned} ConDis(B_{L(\psi)}, B_{L(\omega)}) &= Euclidean(Z_{(\psi)}, Z_{(\omega)}) \\ &= (dis(z_{1(\psi)}, z_{1(\omega)}) + \dots + dis(z_{n(\psi)}, z_{n(\omega)}))^{\frac{1}{2}} \end{aligned} \quad (4)$$

and distance function is calculated as:

$$dis(z_{i(\psi)}, z_{i(\omega)}) = (\zeta_{i(\psi)}^1 - \zeta_{i(\omega)}^1)^2 + \dots + (\zeta_{i(\psi)}^l - \zeta_{i(\omega)}^l)^2, \quad (5)$$

where $\zeta_{i(\psi)}^j$ is the j th element of i th conceptual feature set in vector ψ and l is the length of feature set. If the distance of recent conceptual vector to a concept available in the pool

is less than a predefined threshold, its corresponding classifier will be updated by instances of the recent window. Otherwise a new classifier and concept is added. As the threshold parameter is problem specific, one major shortcoming of this framework (CCP framework) is how to determine the threshold. Our algorithm improves CCP framework in several ways: first, it modifies the distance function in order to normalize the distance values when the range of features are not similar. Second, in the classifying phase of a batch, we use a weighted majority approach and compare it with the active classifier which was used in CCP. Third, we propose three new batch assignment methods and also a merging process to manage the pool. The detailed comparison of algorithms is discussed in Sect. 4.

3 Proposed learning algorithm

In this section we propose a novel algorithm for data stream classification. The algorithm is named Pool and Accuracy based Stream Classification (PASC). The methodology used in this algorithm is similar to the one proposed in (Katakis et al. 2009) in general, but it differs significantly in the content. The algorithm maintains a pool of classifiers to deal with recurring concepts. Each classifier describes a concept of the environment. When a batch of instances is received, the algorithm first predicts the corresponding labels of the instances. Then, the true label of each instance is revealed. After receiving the true labels of the whole instances of the batch, the algorithm either chooses the most appropriate classifier and updates it, or creates a new classifier that suits the received data and their labels. If a new classifier is created, it will be added to the pool. To avoid getting redundant classifiers, there is a constraint on the maximum number of classifiers. It makes sense in the real world too, because there are usually a finite number of concepts in many applications.

The general framework of PASC is shown in Procedure 1. The inputs of this algorithm are unlabeled batches of instances $B_t = (x_{t,1}, x_{t,2}, \dots, x_{t,k})$, where $x_{t,i}$ is the i th data of the t th batch and k is the batch size; true labels of B_t which are received as $L_t = (l_{t,1}, l_{t,2}, \dots, l_{t,k})$ where $l_{t,i}$ is the label of $x_{t,i}$; and an updating parameter θ .

Lines 9–13 constitute the main part of the algorithm. In this loop, classifying a batch and updating the pool is done iteratively. The algorithm has three main phases shown in lines 10–12 and will be discussed in the following subsections. The first 8 lines are the initialization parts of the method that will be described in details later in this section. But in general, C is the classifier created by the first batch of instances as its input. RDC or Raw Data Classifier is used for predicting the concept which B_t is drawn from. RDC do not mention the labels, L_t , and is used in

Bayesian or exact Bayesian methods to update the pool. W_j , the weight of the first classifier, is set to 1 and the classifier C is added to the pool. X_j is a summarization of the first batch that is used to classify the concept of the input batch. In line 8 the raw data classifier is updated by X_j .

3.1 Phase 1: classifying the batch

One possible approach to classify the input batch is based on choosing the active classifier (Katakis et al. 2009). Another approach is to assign a weight to each classifier. These weighted classifiers can be used for classification. Both of these methods will be described in the next two subsections.

3.1.1 Active classifier

In this method the batch is classified using the active classifier. The active classifier is a classifier which was updated by the previous batch of instances. This classifier can exist in the pool before the last iteration or it can be a new added one. The reason for choosing the active classifier is that if there is no concept drift between the two last batches of instances, the same concept is appropriate for both of them.

The method is shown in Procedure 2. The variable pl stores the predicted labels and ac is the active classifier.

Input: $B_i (\geq 1)$: an infinite stream of batches of instances.
 $L_i (\geq 1)$: After classification of each instance $x_{t,i}$, its label is revealed to the algorithm.
 θ : Threshold parameter (θ_j, θ_2 or θ_3)

Output: Predicted labels of instances $x_{t,i}$.

```

1  Pool =  $\emptyset$ ; // the pool of classifiers
2  C = make_classifier( $B_1, L_1$ );
3  RDC = new_classifier(); // used in (exact) Bayesian method
4  ac = 1; // active classifier
5   $W_j = 1$ ;
6  Pool = Pool  $\cup$  {C};
7   $X_j = \text{sum\_data}(B_j)$ ;
8  RDC.update( $X_j, 1$ ); // 1 is the label of  $X_j$ 
9  for  $t=2$  to infinity do
10     Classify  $B_t$ .
11     Update Pool with  $B_t$  and  $L_t$ ;
12     determine active classifier (classifier weights);
13 end for

```

Procedure 1. The main framework of PASC.

```

1  for  $i=1$  to  $k$  do
2      $pl[x_{t,i}] = \text{Pool}[ac].\text{classify}(x_{t,i})$ ;
3  end for

```

Procedure 2. Classify batch according to active classifier.

3.1.2 Weighted classifiers

The active classifier method works well as long as there is no concept drift between two consecutive batches of instances. However, when a drift occurs the active classifier will no longer be adequate and the performance decreases significantly. To avoid this, our approach uses an adaptive method for choosing the suitable classifier from the pool. At the beginning of the iteration, we assign each classifier an initial weight. These weights are computed according to current state of the pool and the last batch of instances. Here it is assumed that, we will be given the true label of $x_{t,i}$ or $l_{t,i}$ right after its classification. So after predicting label of $x_{t,i}$, we update the weights of classifiers using $l_{t,i}$. Updating the weights is done using:

$$w'_j = w_j * \beta^{M(j,i)}, \tag{6}$$

where w_j is the current weight of the j th classifier and w'_j is the new weight and β is a parameter in $(0,1)$. $M(j,i)$ would be 0 if the j th classifier predicts the label correctly and 1 otherwise.

Equation (6) is inspired from (Freund and Schapire 1996) which models the online prediction problem with a two-player repeated game. The first player, here the pool of classifiers, is the learner. The actions of the first player include choosing a classifier among the ensemble (pool) of classifiers to classify an instance. A mixed strategy P is used by the first player to choose its actions. P is a probability distribution function and according to it, each action will have a probability to be selected. The learner computes mixed strategy P by normalizing the weights w_j which are assigned to the classifiers. This is equivalent to use a majority vote among the classifiers of the pool according to their weights. The second player, here the source of producing the batches, is the environment. The actions of the second player include choosing the instances which are given to the learner for classification. The second player uses the mixed strategy Q to choose its actions. The strategies P and Q change themselves as the game proceeds. P is updated according to the loss of the last iteration by updating the classifiers' weights and Q can be updated by the environment arbitrarily. In (Freund and Schapire 1996) it's shown that if the number of instances is sufficiently large, and the learner uses (6) to update the weights of its classifiers, the prediction error converges to the best classifier's error on the last batch. Considering this, if the size of the input batch is large enough, the accuracy of our ensemble classifier on the last batch of instances is as good as using the best classifier in the pool. However, in the context of concept drift, the size of the batch should not be very large, because the I.I.D assumption holds only when the size of the batch is small enough.

For the sake of efficiency our classification algorithm is slightly different from (Freund and Schapire 1996). First, instead of getting a majority vote, we classify the instance with the classifier which has the highest weight. Second, we do not update our classifiers for every instance. Instead, we use a subsample of the batch. In this method, we chose the square root of the batch size as the number of elements for updating the classifiers.

Weighted classifiers method is shown in Procedure 3. In line 1, we get a subsample and in line 5 we make sure to update the classifiers’ weights just for the members of the subsample S_t .

```

1   $S_t = \text{sub\_sample}(B_t, m);$  /* makes a sub_sample of size  $m$ */
2  for  $i=1$  to  $k$  do
3       $p[[x_{t,i}]] = \text{classifyw}(Pool, W, x_{t,i});$ 
4      /*Uses the highest weighted classifier*/
5      if  $S_t$  does not contain  $x_{t,i}$ 
6          continue;
7      end if
8      for  $j=1$  to  $Pool.size()$  do
9           $W_j = W_j * Pool[j].\text{error}(x_{t,i}, L_{t,i});$  /*error is a binary function*/
10     end for
11 end for

```

Procedure 3. Classify batch according weighted classifiers.

3.2 Phase 2: updating the classifier pool

As mentioned earlier, after receiving the batch and classifying it, true labels reveal. Using the true labels, the algorithm may create a new classifier or update one of the existing classifiers. It’s notable that because the size of the batch is small enough, it’s assumed that all the instances of B_t are originated from an identical concept. So, all their labels can be predicted by only one of the classifiers. In this phase (Procedure 4), the algorithm should choose which classifier to update. Concretely, given B_t and L_t we want to find the concept which describes them with the highest probability (line 1). For this purpose, we propose three different approaches which are named batch assignment methods. The first one is to use the Bayes’ theorem to find the probabilities, the second is similar to the first except that it makes some assumptions to decrease the time complexity of the method, and the third one is Heuristic method which is more efficient than the other two methods. These methods output the most similar concept to the labeled batch ($bestC$ or the best classifier of the pool) and a similarity measure between them ($maxS$). If the similarity measure is more than a predefined threshold parameter (θ_1 , θ_2 or θ_3 for exact Bayesian, Bayesian and Heuristic methods, respectively), the batch and its labels should be given to the best classifier to be updated (lines 2 and 3). Otherwise, if there is a free space in the pool or a free space can be created, a new classifier will be created with the

newly arrived batch and its labels as its initial input (lines 4–7). To create a free space, it is checked to see whether a merging process can be done to merge two classifiers of the pool. This space is then given to a new classifier trained on the newly arrived labeled batch. The mergeProcess function in line 4, checks whether the merge can be done. In the case of positive result of this function, two classifiers of the pool will merge. If a new classifier cannot be added to the pool, the labeled batch will be given to the best classifier to be updated (line 9). Finally, RDC is updated according to B_t and $bestC$ (line 11) which will be described in the next subsection. In the following subsections, merging process and various batch assignment methods will be described.

```

1  ( $maxS, bestC$ ) = batch_assignment( $Pool, B_t, L_t$ );
2  if ( $maxS > \theta$ )
3       $Pool[bestC].\text{update}(B_t, L_t);$ 
4  else if ( $Pool.size() < maxC$  or mergeProcess( $Pool, bestC, B_t, L_t$ ))
5       $C = \text{make\_classifier}(B_t, L_t);$ 
6       $Pool = Pool \cup \{C\};$ 
7       $bestC = Pool.size();$ 
8  else
9       $Pool[bestC].\text{update}(B_t, L_t);$ 
10 end if
11  $RDC.\text{update}(B_t, bestC);$  /*only in (exact) Bayesian*/

```

Procedure 4. Update the classifier pool.

3.2.1 Merging process

If a newly arrived batch is not similar enough to any of the classifiers in the pool and the pool is full, then it should be checked whether the merging process can be done or not. To do so, assume the variable nC refers to the nearest concept to $bestC$. If the distance between nC and $bestC$ is shorter than the distance between $bestC$ and the labeled batch, a merging process will be done and $bestC$ and nC are merged together. Merging of $bestC$ and nC should be needless to the instances which made the two corresponding classifiers. We used Naïve Bayes as the base classifier and implemented a simple method to merge two Naïve Bayes classifiers by combining the probability density functions (pdf) maintained for each attribute. For each nominal attribute f_i and class c_j , a probability $P(f_i = v|c_j)$ is maintained for each possible value, v , of f_i . The corresponding probability of the merged classifier is the weighted average of the two probabilities of the two classifiers with respect to the number of instances. For each numeric attribute f_i and class c_j , a normal distribution $P(f_i|c_j)$ is maintained. The mean value of one of the two normal distributions is used to update the mean value and the standard deviation of the other normal distribution according to its number of instances. The result distribution

is then used as the corresponding distribution of the merged classifier. The pdfs maintained for the class distributions are merged similarly. Moreover, in Bayesian or exact Bayesian batch assignment methods, *RDC* classifier must be updated and it will be discussed later.

The similarity measure used here is a modified version of ConDis used in CCP framework method (Katakis et al. 2009) and we refer to it as normalized ConDis or simply NConDis. It also uses the conceptual vectors of the instance batches (or the average of the conceptual vectors of the batches constituting a classifier) as its base vectors. ConDis is a distance function between the conceptual features of two conceptual vectors. NConDis is similar to ConDis, except that it makes some normalization on the values of the distance between different conceptual features to improve the functionality of the similarity measure even if the ranges of different attributes are not similar. NConDis between two conceptual vectors Z_A and Z_B is defined as:

$$NConDis(Z_A, Z_B) = \{dis(z_{1(A)}, z_{1(B)}) + \dots + dis(z_{n(A)}, z_{n(B)})\}^{1/2} \tag{7}$$

If f_i is nominal, $dis(z_{i(A)}, z_{i(B)})$ is defined as:

$$dis(z_{i(A)}, z_{i(B)}) = \sum_{k=1}^{size(z_i)} dis(z_{ik(A)}, z_{ik(B)})^2, \tag{8}$$

where $z_{ik(X)}$ is the k th element of $z_{i(X)}$ and $dis(z_{ik(A)}, z_{ik(B)})$ is simply the difference of these values. But if f_i is numeric, this distance is defined as:

$$dis(z_{i(A)}, z_{i(B)}) = \sum_{j=1}^m \min \left(\left(\frac{\mu_{ij(A)} - \mu_{ij(B)}}{\sigma_{ij(A)} + \sigma_{ij(B)}} \right)^2, 1 \right) \tag{9}$$

where m is the number of different classes.

3.2.2 Exact Bayesian batch assignment method

A way to find the most related concept, i.e. the most related classifier of the pool is to compute the probability that B_t and L_t correspond to the concept described by h_i :

$$P(h_i|B_t, L_t) = \frac{P(B_t, L_t|h_i) * P(h_i)}{P(B_t, L_t)}, \tag{10}$$

where the right hand side follows from the Bayes' theorem. To derive the best concept, using (10), we should find:

$$argmax_i P(h_i|B_t, L_t) = argmax_i P(B_t, L_t|h_i) * P(h_i). \tag{11}$$

The above equation is on the assumption that the best h_i is independent of the probability of B_t and L_t . As the environment is non-stationary, we cannot have any assumptions about the concepts. In fact, $P(h_i)$ can depend

on the previously seen concepts and the underlying environment. The former can vary in the context of concept drift and cannot be modeled appropriately without any assumptions. In addition, assuming different values for $P(h_i)$ considering the first kind of dependencies will lead to late detection of concept drifts, because the concepts which have appeared in the last batches of instances can get higher probabilities. The second kind of dependencies is not known to the algorithm. If these dependencies are known for some dataset, it can be used to evaluate (11) more accurately. So we consider $P(h_i)$ to be identical for all concepts. Thus it has no effect on our calculations and (11) becomes:

$$argmax_i P(h_i|B_t, L_t) = argmax_i P(B_t, L_t|h_i) \tag{12}$$

If we refer to the elements of B_t and L_t , we obtain:

$$= argmax_i P((x_{t,1}, l_{t,1}), (x_{t,2}, l_{t,2}), \dots, (x_{t,k}, l_{t,k})|h_i), \tag{13}$$

which in turn equals to:

$$= argmax_i P((x_{t,1}, l_{t,1})|h_i) * P((x_{t,2}, l_{t,2})|h_i, (x_{t,1}, l_{t,1})) * \dots * P((x_{t,k}, l_{t,k})|h_i, (x_{t,1}, l_{t,1}), \dots, (x_{t,k-1}, l_{t,k-1})). \tag{14}$$

Suppose we define h_{ij} as the hypothesis h_i which is supported by the first j labeled instances of the batch:

$$h_{i,j} = h_i(x_{t,1}, l_{t,1}), \dots, (x_{t,j}, l_{t,j}). \tag{15}$$

$$P(x_{t,j+1}, l_{t,j+1}|h_{i,j}), \quad 0 \leq j \leq k - 1. \tag{16}$$

Now the task turns out to estimate some probabilities of the form:

Equation (16) in turn equals to:

$$P(x_{t,j+1}, l_{t,j+1}|h_{i,j}) = P(l_{t,j+1}|h_{i,j}, x_{t,j+1}) * P(x_{t,j+1}|h_{i,j}), \text{ where } 1 \leq j \leq k. \tag{17}$$

The term $P(l_{t,j+1}|h_{i,j}, x_{t,j+1})$ can be estimated using the classifier describing h_i . Note that this term equals to the probability that the label of $x_{t,j+1}$ be $l_{t,j+1}$, given that the true concept is h_i and the j labeled instances of (18) are produced in the environment of h_i . So it is sufficient to update the i th classifier with the j labeled instances mentioned above and then output the posterior probability for the instance $x_{t,j+1}$. By updating the classifier in the right order, it is sufficient to update it $k-1$ times to estimate all these terms. Note that after updating the i th classifier, it should be converted to its last state. This can be done easily by making a copy of the classifiers before the updating process and then use them as the original classifiers with no change.

We estimate the second term of the right hand side of (17) by making an assumption. Assume that:

$$P(x_{t,j+1}|h_{i,j})P = (x_{t,j+1}|h_i, x_{t,1}, \dots, x_{t,j}). \quad 0 \leq j \leq k - 1. \tag{18}$$

The reason behind this assumption is that the probability of producing an unlabeled instance given that the environment is described by h_i and we know that j labeled instances have been produced in this environment, can be estimated independent of the labels of the instances. In this case, these probabilities can be estimated as follows: We create a classifier named *raw data classifier (RDC)*. The input of *RDC* is an instance and the output prediction is the most related concept. To train this classifier, after receiving the true labels of each batch B_t and determining the concept of the labeled instances of this batch, we give these the instances (and not their labels) and their corresponding concept ids as their class labels to *RDC* to update itself. To determine $P(x_{t,j+1}|h_i)$, the posterior probability of *RDC* on $x_{t,j+1}$ or $P(h_i|x_{t,j+1})$ can be used, because prior probabilities of the concept or $P(h_i)$ are assumed to be identical for all concepts. In addition, to estimate $P(x_{t,j+1}|h_i, x_{t,1}, \dots, x_{t,j})$, it is convenient to update *RDC* with j instances $x_{t,1}, \dots, x_{t,j}$ and the concept id i as the class label and then output the posterior probability.

Note that by updating *RDC*, $k-1$ updates are sufficient to estimate these probabilities for each concept in the pool. However, after estimating the probabilities for each concept, *RDC* has been changed. This is similar to the case of updating the classifiers of the pool for estimating the first term of (17) and the same solution can be applied in this case.

The following equation determines the most relevant concept to the newly arrived labeled batch B_t :

$$\begin{aligned} & \text{argmax}_i P(l_{t,1}|h_i, x_{t,1}) * P(l_{t,2}|h_{i,1}, x_{t,2}) \\ & * \dots * P(l_{t,k}|h_{i,k-1}, x_{t,k}) * P(x_{t,1}|h_i) * P(x_{t,2}|h_i, x_{t,1}) \\ & * \dots * P(x_{t,k}|h_i, x_{t,1}, \dots, x_{t,k-1}). \end{aligned} \tag{19}$$

To prevent underflow of these probabilities, the sum of *log* of the terms in (19) is used:

$$\begin{aligned} & \text{arg max}_i (\log P(l_{t,1}|h_i, x_{t,1}) + \log P(l_{t,2}|h_{i,1}, x_{t,2}) + \dots + \\ & \log P(l_{t,k}|h_{i,k-1}, x_{t,k})) + (\log P(x_{t,1}|h_i) + \log P(x_{t,2}|h_i, x_{t,1}) \\ & + \dots + \log P(x_{t,k}|h_i, x_{t,1}, \dots, x_{t,k-1})). \end{aligned} \tag{20}$$

Calculating (20) for all instances is a very time consuming action, so calculation is done on a subsample of the square root size of the batch (Procedure 5). Lines 1 and 2 of Procedure 5 select a subsample of size m (square root size of the batch) and its labels from B_t and L_t . In lines

4 and 5, Eq. (20) is calculated for all concept in the pool and the selected subsamples from B_t and L_t . Then the best concept, *bestC*, and its estimated similarity, *maxS*, to B_t and L_t are determined.

1	$S_t = \text{sub_sample}(B_t, m);$
2	$SL_t = \text{sub_sample}(L_t, m);$
3	<i>/* stores the labels of S_t*/</i>
4	$(\text{maxS}, \text{bestC}) = (\text{max}, \text{argmax})_{c \in 1.. \text{PoolSize}()}$
5	$\text{exBayesianSimilarity}(\text{Pool}, \text{RDC}, c, (S_t, SL_t)) \text{ /*Equation (20)*/}$

Procedure 5. Exact Bayesian batch assignment method.

3.2.3 Bayesian batch assignment method

This method is similar to exact Bayesian method. In both methods, it is tried to find the value of right hand side of (12) for the concepts described by the classifiers pool. In fact the difference between these two methods is how to deal with the expression. In Bayesian method, (12) is estimated as:

$$\begin{aligned} \text{argmax}_i P(h_i|B_t, L_t) &= \text{argmax}_i P(B_t, L_t|h_i) \\ &= \text{argmax}_i P(B_t|h_i) * P(L_t|B_t, h_i). \end{aligned} \tag{21}$$

So we have to calculate the terms $P(L_t|B_t, h_i)$ and $P(B_t|h_i)$. $P(L_t|B_t, h_i)$ is the conditional probability that the predicted labels of $(x_{t,1}, x_{t,2}, \dots, x_{t,k})$ are $(l_{t,1}, l_{t,2}, \dots, l_{t,k})$ using classifier h_i . $P(B_t|h_i)$ is the conditional probability that the batch, B_t , be produced in an environment described by the i th concept. As mentioned earlier, if there is a concept drift, the I.I.D condition will not hold anymore. To assure I.I.D condition holds between the instances of a batch, its size should be small enough. So we assume all elements of each batch are I.I.D and so (22) holds:

$$P(L_t|B_t, h_i) = \prod_{j=1}^{j=k} P(l_{t,j}|x_{t,j}, h_i). \tag{22}$$

The $P(l_{t,j}|x_{t,j}, h_i)$ terms on the right hand side is calculated directly from the i th classifier. Again as we assumed the I.I.D condition holds within the batch, B_t , we obtain:

$$P(B_t|h_i) = \prod_{j=1}^{j=k} P(x_{t,j}|h_i). \tag{23}$$

We propose the following method for calculating $P(x_{t,j}|h_i)$, i.e. the probability that j th instance of t th batch comes from the i th concept. We create a classifier *RDC* similar to the one used in exact Bayesian method. Thus the input to this classifier is one of the instances and it predicts the related concept as output. To train this classifier, after receiving the true labels of B_t , we predict the concept of elements of the batch. Thus to update the *RDC* we give it

the batch instances and their corresponding concept ids. To determine the relative concept of the batch, one way is to calculate $P(x_{t,j}|h_i)$ for all the elements (similar to exact Bayesian method and assuming equal prior probabilities for all concepts); however, this is time consuming and inefficient. Instead we use a statistic of batch elements X_t to train the *RDC*. We simply define $X_t = \sum x_{t,j}$ i.e. sum of all elements of the t th batch.

After receiving B_t , X_t is built and given to *RDC* to calculate the probability for all concepts. Assume that *RDC* gives with probability p that the t th batch elements belongs to the i th concept. Then we obtain:

$$P(B_t|h_i) \propto p_i^k \tag{24}$$

Therefore, to determine the best classifier, (21) turns into:

$$\arg \max_i P(h_i|B_t, L_t) = \arg \max_i p_i^k * \prod_{j=1}^{j=k} P(l_{t,j}|x_{t,j}, h_i) \tag{25}$$

To prevent underflow of the products, we use (26) Instead of (25) to find the best concept:

$$\begin{aligned} \arg \max_i P(h_i|B_t, L_t) &= \arg \max_i k * \log p_i \\ &+ \sum_{j=1}^{j=k} \log P(l_{t,j}|x_{t,j}, h_i). \end{aligned} \tag{26}$$

Still this equation leads to inefficient algorithm because calculating the posterior for all $x_{t,j}$ is time consuming. Considering that I.I.D condition holds within a batch, we can resolve this problem by using a batch subset of smaller size. A subsample of the square root of the pool size is used to estimate (26). The pseudocode of this method is shown in Procedure 6. Lines 5 and 6 find the best describing classifier of the batch according to Bayesian method.

```

1   $X_t = \text{sum\_data}(B_t);$ 
2   $S_t = \text{sub\_sample}(B_t, m);$ 
3   $SL_t = \text{sub\_sample}(L_t, m);$ 
4  /* stores the labels of  $S_t$  */
5   $(\text{maxS}, \text{bestC}) = (\text{max}, \text{argmax})_{c:1 \dots \text{Pool.size()}}$ 
6   $\text{BayesianSimilarity}(\text{Pool}, \text{RDC}, c, S_t, SL_t, X_t) /* \text{Equation (26)} */$ 

```

Procedure 6. Bayesian batch assignment method.

3.2.4 Heuristic batch assignment method

In this method, to find the best concept that describes B_t and L_t , the accuracy of all classifiers are calculated on B_t and the most accurate one is updated. Therefore, the similarity measure is the accuracy of the classifiers on B_t . The intuition behind this approach is that when a classifier is more accurate on the current batch, it's probably more relevant to it. The pseudocode of this method is shown in

Procedure 7. Lines 4 and 5 find the best classifier describing the batch according to Heuristic method.

```

1   $S_t = \text{sub\_sample}(B_t, m);$ 
2   $SL_t = \text{sub\_sample}(L_t, m);$ 
3  /* stores the labels of  $S_t$  */
4   $(\text{maxS}, \text{bestC}) = (\text{max}, \text{argmax})_{c:1 \dots \text{Pool.size()}}$ 
5   $(\text{pool}[c].\text{accuracy}(S_t, SL_t));$ 

```

Procedure 7. Heuristic batch assignment method.

3.3 Phase 3: determining the active classifier or classifiers' weights

After these two phases, some settings should be done to get ready for the next iteration. If the active method for classifying the batches is used, the next active classifier should be determined. Active classifier is the one that is updated in the current iteration (i.e. *bestC* in Procedure 1).

If the weighted classifier method is used, the weights of the classifiers should be set for the next iteration. It is worth noticing that if we use the weights computed in this iteration for the next step, this method would work similar to the active classifier method or even worse, because one of the classifiers weights may get too high, as a result of not having concept drift for a long time. Then it will take so long that this weight decreases and the performance decreases significantly. We use:

$$w_0(i) = \beta^{(2^{err(i)})}, \tag{27}$$

where $err(i)$ is the error of the i th classifier on a subsample of the square root size of the batch. So, the more inaccurate classifier, the less initial weight it will have for the next iteration. Some kind of locality assumption is used in (27) for setting the initial weights which does not work properly when sudden concept drift occurs. As mentioned in phase one, such concept drift is handled by updating the weights during batch processing (Procedure 8).

```

1   $S_t = \text{sub\_sample}(B_t, m);$ 
2   $SL_t = \text{sub\_sample}(L_t, m);$ 
3  for  $j=1$  to  $\text{Pool.size}()$  do
4   $c\_error = \text{Pool}[j].\text{error}(S_t, SL_t);$ 
5   $W_j = \beta^{(2^c \cdot c\_error)}$ ;
6  end for

```

Procedure 8. Determine classifier weights.

4 Experimental results

In order to evaluate performance of the proposed algorithm, computer experiments on some standard datasets are conducted. In this section, we first introduce the datasets

Table 1 Emailing list dataset (Elist) (Katakis et al. 2009)

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

which contain recurring concepts in 4.1. These datasets are used in the experiments. Then in 4.2 we discuss the parameter tuning of the proposed methods and compare it with the parameter tuning of CCP framework, one of the most promising frameworks developed in the tracking of recurring concepts. Finally, the proposed methods are compared with each other and CCP framework method in 4.3. The experiments show the effectiveness of the proposed method. It can be seen that weighted classification method outperforms active classification method in datasets containing sudden concept drift. The proposed batch assignment methods outperform CCP batch assignment method in datasets with arbitrary attributes. Finally, the effectiveness of the merging process is shown for large datasets and sensitivity of the algorithm to its parameters is studied.

4.1 Datasets

Three real datasets and one artificial dataset are chosen for the experiments given in this section. The artificial dataset is moving hyperplanes and contains sudden concept drift. Real datasets are emailing list (Katakis et al. 2009), spam filtering and sensor data. Emailing list and spam filtering are high dimensional datasets and sensor data is a very large real dataset. Emailing list and hyperplane datasets contain sudden concept drift and spam filtering and sensor data contain gradual drift.

4.1.1 Emailing list dataset

In this dataset, a stream of email from different topics are collected and labeled as interesting or junk with respect to the user's interest (Katakis et al. 2009). Usenet posts data (Frank 2010) which contains 20 newsgroups collection is used to construct this dataset and three of its topics are selected. In each time interval (concept), the user is interested in one or two topics and labels the emails according to his/her interest. User interests may vary time to time, so the dataset contains recurring concepts and sudden drifts (Table 1). The labels and the existing drifts of this dataset are artificial. So elist is not a pure real dataset, but its instances are real. The dataset has 1,500 instances and 913 attributes and is divided into 5 time intervals of equal length of data (Elwell and Polikar 2011).

4.1.2 Spam filtering dataset

Spam filtering dataset is extracted from Spam Assassin¹ collection. It contains 9,324 email messages with 500 attributes and two possible labels. This dataset consists of gradual concept drift (Elwell and Polikar 2011).

4.1.3 Hyperplane dataset

The aim of this dataset is to predict the class of a rotating hyperplane. A hyperplane decision surface is represented by equation $g(\vec{X}) = \vec{W} \cdot \vec{X} = 0$, where \vec{W} is an n-dimensional vector showing the orientation of the surface and \vec{X} is the instance. If for an instance we have $(g(\vec{X}) > 0)$, we classify it as 1, otherwise it is classified as 0. The hyperplane is moving through the time and simulates sudden concept drift. We generated 8,000 instances with 30 numeric attributes. After receiving 2,000 instances concept drift occurs suddenly. To simulate recurring concept problem there is a reappearance of concepts after the first 4,000 instances.

4.1.4 Sensor dataset

Sensor dataset is a real dataset which consists of the information collected from 54 sensors deployed in Intel Berkeley Research laboratory in a two-month period (Zhu and 2010). The class label is the sensor ID, so there are 54 classes, 5 attributes and 2,219,803 instances. The type and place of concept drift is not specified in the dataset but it is obvious that there are some drifts. For example, lighting or the temperature of some specific sensors during the working hours is much stronger than nights or weekends.

4.2 Parameter tuning

The proposed learning algorithm is designed in such a way that most of its parameters can be set simply. In fact, they can be set according to general properties of the datasets as it will be discussed later or they can be chosen from a specified range of values. This is not the case in CCP framework method and so it is an advantage of our algorithm. CCP framework method has a parameter θ with similar effect to our parameters θ_1 , θ_2 and θ_3 . If this parameter is set wrongly in CCP framework method, the accuracy of the classification will decrease significantly. For example, θ should be set to 4 for elist and 2 for hyperplane datasets. However, if we set θ to 4 instead of 2 for hyperplane dataset, its accuracy will decrease 10 % (from 78 % to 68 %). Besides, there is no knowledge for

¹ The Apache SpamAssassin Project-<http://spamassassin.apache.org/>.

the proper range of this parameter in CCP framework method and low or high values of this parameter will lead to low performances, but we explain how to set the threshold parameters of our methods. On the other hand, parameters θ_i of the proposed methods are equally set for all four datasets and acceptable results are obtained.

The parameters of the performed experiments are set as what follows, except specified otherwise:

If weighted classifiers method is used for phase 1, a parameter β will be required in the updating process of the classifiers' weights. This parameter is by definition in the interval (0,1). The smaller the parameter β is, the faster the updating of the weights of the classifiers is done in response to the potential concept drift. So if the dataset contains sudden concept drift, this parameter should be low. Otherwise, the higher values lead to more robustness to noise. This parameter is set to 0.1 to obtain fast responses to concept drift. Another parameter is the maximum classifier number ($maxC$) which determines the maximum expected number of possible concepts. To set this parameter, it should be considered that how many classifiers will be enough to describe the different concepts of the datasets. This knowledge may be apparent for some datasets. Otherwise, it can be tuned by trial and error technique. $maxC$ is set to 10 for all datasets meaning that 10 different concepts are enough to describe each of the four datasets. The parameter θ_3 used in Heuristic batch assignment method is a threshold on the accuracy of a classifier on a newly arrived batch. If the accuracy is more than θ_3 , it is assumed that the classifier describes the concept of the batch correctly. This parameter is set to 0.95 for the first three datasets and to 0.8 for sensor dataset. This means that the method expects a classifier of the pool to have accuracy higher than these values to be appropriate to describe the last batch of instances. The values are chosen by trial and error so that the overall accuracy of the method will almost be the highest. The parameters θ_1 and θ_2 used as the described thresholds of Bayesian and exact Bayesian methods are set to $2 * \log(0.65) * m$ and $2 * \log(0.75) * m$ and, respectively, where m is the number of instances used to determine the relevancy of a concept to a new batch. This is because we believe that if none of the probabilities in the relevancy equations is less than 0.75 for Bayesian and 0.65 for exact Bayesian method, the concept with this property is relevant to the newly arrived batch and its labels. Besides, these parameters can be set by changing the values in the logarithms (0.75 and 0.65) between 0 and 1 and noting that very high or low values will not lead to good results. The batch size parameters are set to 500 for hyperplane dataset and 50 for the other datasets. These batch sizes have the required properties of the batch size parameter: They are not very large for the datasets so that the probability of drift inside a batch is low and not very

small so that the batch would have enough instances to describe a concept appropriately.

The CCP framework method has two parameters, namely $maxC$ and θ . In order to provide a fair comparison between the proposed method and CCP framework method, $maxC$ is set to 10 for all datasets, i.e. the same value as the proposed method and the parameter θ is chosen by trial and error so that the highest accuracy will be obtained. θ is set to 4, 2.5, 0.1 and 2 for the elist, spam filtering, hyperplane and sensor datasets, respectively.

As a result, the parameter tuning process of our method is more straightforward than CCP framework method; because they can be set according to the general properties of the datasets or they can be chosen from a range of values by trial and error. Moreover, almost the same parameters are shown to work well on the various datasets we have chosen. These datasets have different attributes and natures, but the applied methods only care about the correctness of the classification of the classifiers of the pool and do not depend on the different natures of the datasets. So the lower dependency of the parameters to the datasets is expected. The only parameter that differs in different methods is the batch size parameter. This is a concern of CCP framework method, too.

4.3 Results and discussion

We have performed three experiments in order to evaluate PASC method. The first experiment which is presented in the next subsection compares the performance of the different variations of PASC with each other and with CCP framework method. The second experiment is designed to evaluate the effect of the merging process and sensitivity to the threshold parameters. The third experiment is designed to evaluate the effect of the parameter β of PASC. The two following subsections discuss the results obtained from the first experiment and the two last subsections discuss the second and third experiments.

In the first experiment, we compared the different variations of PASC with CCP framework method in terms of accuracy, precision, recall, f-measure running time. The parameters used in these experiments are discussed in 4.2. The average accuracy values of the methods on the consecutive batches of instances (except the first batch which is ignored in all methods) of elist, spam filtering, hyperplane and sensor datasets are shown in Figs. 1 and 2. The figures consist of four parts each showing the plot of the accuracies of using the four batch assignment methods and active classifier or weighted classifiers methods on a given dataset. So for each dataset, two plots are shown for the two classification methods. In addition, the accuracies, precisions, recalls and running times of the different methods are shown in Tables 2 through 5.

Fig. 1 Results of active classifier method on all datasets

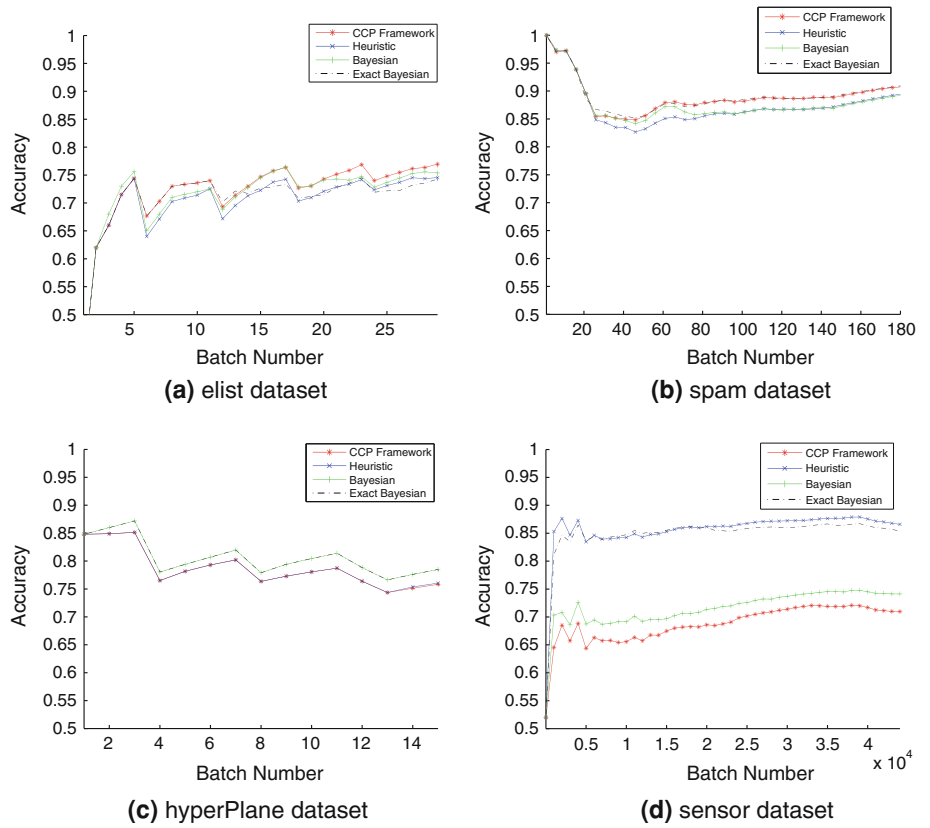


Fig. 2 Results of weighted classifiers method on all datasets

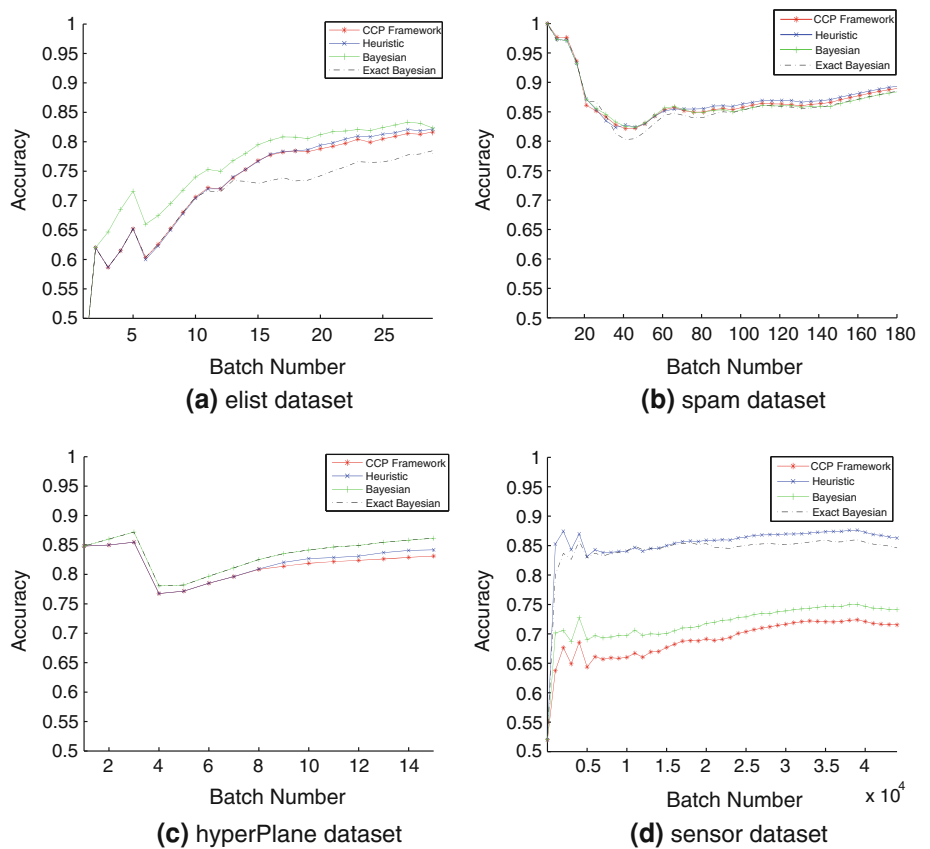


Table 2 Results of all methods on elist dataset

Batch assign	Classification method	Acc.	P	R	F-measure	Time
CCP	Active	0.77	0.73	0.81	0.77	1,004
	Weighted	0.82	0.79	0.83	0.81	1,274
Heuristic	Active	0.75	0.71	0.77	0.74	1,816
	Weighted	0.82	0.8	0.83	0.81	1,843
Bayes	Active	0.75	0.71	0.8	0.75	2,089
	Weighted	0.82	0.8	0.84	0.82	2,462
Exact Bayes	Active	0.74	0.73	0.71	0.72	32,039
	Weighted	0.78	0.78	0.76	0.77	32,339

Table 3 Results of all methods on spam dataset

Batch assign	Classification method	Acc.	P	R	F-measure	Time
CCP	Active	0.91	0.91	0.84	0.94	2,217
	Weighted	0.89	0.92	0.87	0.93	2,820
Heuristic	Active	0.89	0.91	0.84	0.93	3,942
	Weighted	0.89	0.92	0.89	0.93	4,112
Bayes	Active	0.89	0.9	0.86	0.93	4,537
	Weighted	0.88	0.91	0.91	0.92	5,405
Exact Bayes	Active	0.91	0.92	0.86	0.94	109,857
	Weighted	0.89	0.91	0.91	0.92	112,266

4.3.1 Comparison of methods’ accuracies, precisions and recalls

Parts (a) through (c) of Figs. 1 and 2 show that the four batch assignment methods are almost similar in the performance on the first three datasets. But the last parts of these figures which contain the results on sensor dataset show that CCP and Bayesian batch assignment methods has lower performances (between 9 % and 15 % of accuracy) than Heuristic and exact Bayesian methods. This means that CCP framework and Bayesian methods have some problems in determining the true concept of a batch in sensor dataset. One problem with CCP framework method is that it uses the Euclidean distance as the measure of similarity of a batch to a concept. ConDis, the distance measure used in CCP, is dependent on the magnitude of the attribute values and an attribute with large values can reduce the effects of the other attributes in the distance calculation. The problem of Bayesian method could be possibly the I.I.D assumptions made in it, as it can be seen that exact Bayesian method does not suffer from this problem. However, Bayesian method still outperforms CCP framework method (about 3 %).

Considering the plots of Figs. 1 and 2, it is deduced that although there are some differences in the accuracies of the

Table 4 Results of all methods on hyperplane dataset

Batch assign	Classification method	Acc.	P	R	F-measure	Time
CCP	Active	0.76	0.72	0.81	0.78	868
	Weighted	0.83	0.81	0.83	0.84	947
Heuristic	Active	0.76	0.73	0.77	0.78	974
	Weighted	0.84	0.81	0.83	0.85	970
Bayes	Active	0.78	0.75	0.8	0.8	876
	Weighted	0.86	0.83	0.84	0.87	899
Exact Bayes	Active	0.78	0.75	0.71	0.8	1,135
	Weighted	0.86	0.83	0.76	0.87	1,178

Table 5 Results of all methods on sensor dataset

Batch assignment	Classification method	Accuracy	Time
CCP	Active	0.71	370,560
	Weighted	0.71	813,398
Heuristic	Active	0.87	929,289
	Weighted	0.86	846,226
Bayes	Active	0.74	883,682
	Weighted	0.74	1,299,652
Exact Bayes	Active	0.84	1,596,031
	Weighted	0.83	2,184,393

different batch assignment methods, the variations in the accuracy of all of these methods are similar. This means that all batch assignment methods are affected by the drifts of the datasets in the same way.

The other important note that can be interpreted from Figs. 1 and 2 is that weighted classifiers method outperforms (5 % to 8 %) active classifier method when the concept drift of the dataset is sudden. This can be seen in the part (a) of Figs. 1 and 2 for elist dataset and in the part (c) of hyperplane dataset. However, when the concept drift is gradual, the two classification methods work almost the same. This is reasonable because when a sudden concept drift occurs while processing a batch (especially in the beginning of a batch), weighted classifiers method quickly adapts the weights according to the drift and so the performance does not decrease significantly. This can be seen by comparing part (a) (or part (c)) of Figs. 1 and 2. In part (a) of Fig. 1, the classification accuracy decreases in some points (when a sudden concept drift occurs) and it takes some reasonable time for this method to regain its past accuracy. But in the same points in part (a) of Fig. 2, the accuracy will not decrease as much as in the last case and the time taken to regain the past accuracy is much less. However, the advantage of weighted classifiers method over active classifier method cannot be seen in datasets containing gradual concept drift. In this case, the both

methods work almost the same as each other. Besides, weighted classifiers method needs that after classification of each instance (in the selected subsample), the true label becomes revealed and this condition is not needed for active classification method.

Tables 2, 3, 4 and 5 show the average accuracies, precisions, recalls and overall running times of the different methods on the four datasets (For sensor dataset, precisions, recalls and f-measures are not computed since it has more than two classes). The minor differences in the accuracies can be seen in these tables. The running times of the different methods are listed in the tables which will be discussed in the next subsection. The other point that can be seen in these tables is that the precisions and recalls of the different methods are proportional to their accuracies in most of the cases. So instead of comparing the f-measures of the different methods, their accuracies can be compared.

4.3.2 Comparison of run times of different methods

We could see the run time of each method in the last columns of Tables 2 through 5. In this comparison, first we compare the four batch assignment methods. Then active and weighted classifiers methods are compared.

Training and testing of classifiers are the most time consuming parts of the algorithms. One other time consuming part in CCP framework is constructing conceptual vectors and clustering them. Time complexity for updating the most similar classifier by the current batch is linear in the number of instances and it is the same for all methods. In addition, in the classification task of all batch assignment methods, each data is classified once. However, the number of classifications and measurements of the posterior probabilities distributions and updates done for an instance is different for batch assignment methods. The running time of the methods can be expressed by four terms: T_0 , T_1 , T_2 , and T_{RDC} , where T_0 is the time expended to classify an instance, T_1 is the time taken to find the posterior probabilities for it, T_2 is the time taken to update a classifier and T_{RDC} is the time taken while using and updating raw data classifier (*RDC*) in Bayesian method. To find the most similar classifier to the current batch, a subsample of size m from data in the recent window is used in our three batch assignment methods. Using all of the classifiers in the pool, each of m instances is classified once in Heuristic method, and its posterior probabilities estimation is measured in Bayesian method. T_{RDC} , another term in Bayesian method's run time and the computation time in updating and using *RDC* in Bayesian method is a constant time for each batch. This time includes the time taken to construct X_i for the batch B_i , and finding the posterior probabilities for X_i using *RDC* and also updating *RDC* with X_i and the corresponding classifier number.

Exact Bayesian method needs an update between each two posterior probabilities distribution computations in addition to the computation of the posterior probabilities in Bayesian method. Besides, the same computation time for updating and posterior probabilities distribution computations is needed for *RDC*. So the most required time for Heuristic method is $m * maxC * T_0$, for Bayesian method is $m * maxC * T_1 + T_{RDC}$ and for exact Bayesian method is $2 * (m * maxC * T_1 + (m - 1) * maxC * T_2)$, where $maxC$ is the maximum number of classifiers of the pool.

As it is expected, exact Bayesian method takes the most time among all. It is obvious from definition that T_1 and T_0 are almost the same for Naïve Bayes classifier used in our experiments. Consequently, Bayesian method is expected to take more time than Heuristic method. Results in Tables 2 through 5 uphold this fact; however, there are some minor inconsistencies for this rule in some of the methods and datasets. The reason for these inconsistencies could be the differences because of parameter settings and the number of classifiers in the pool for different methods. In addition, it is important to note that the above expressions are written using the maximum number of classifiers and in fact they only determine the upper bounds. It is notable that CCP framework method takes the least time among all while using each of the classification methods for classification, but the time taken in Heuristic and weighted classifiers methods is not far more than the running time of CCP framework and weighted classifiers methods.

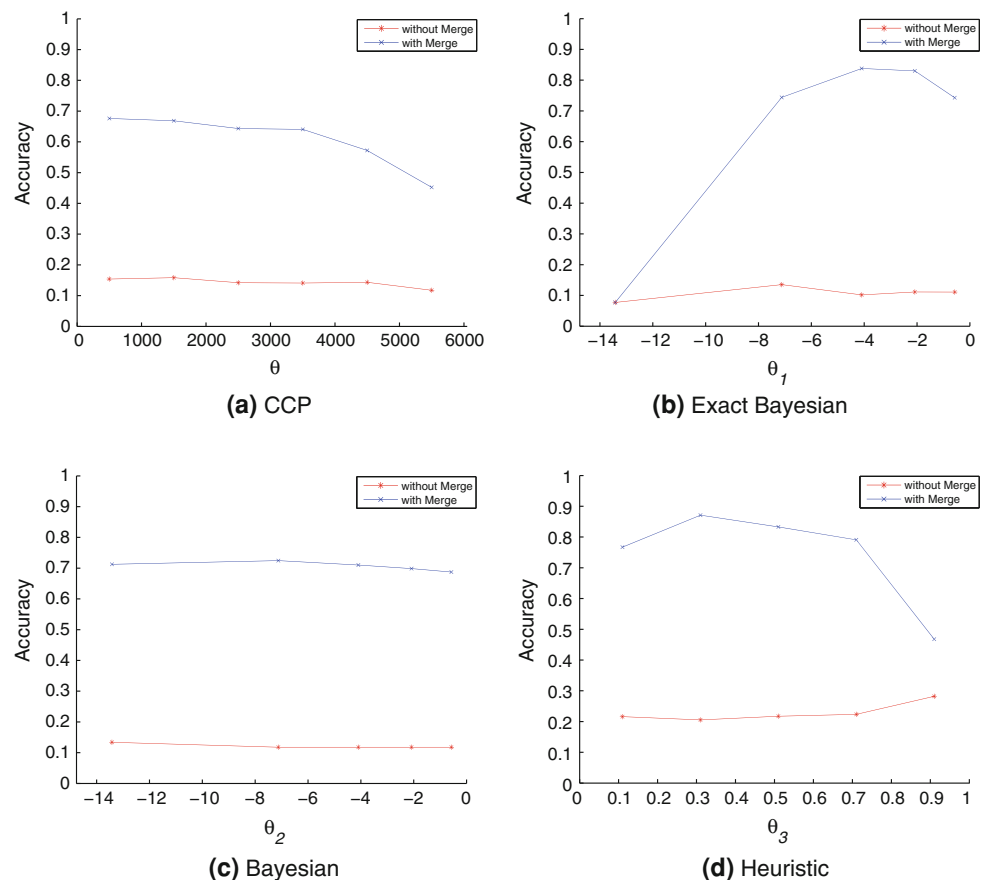
As weighted classifiers method needs updating the classifiers' weights which in turn requires classification of some instances, it is expected to take more time than active method. This can be seen for all batch assignment methods, except for Heuristic method, because to obtain time saving in Heuristic and weighted classifiers methods, we used the same subsample for both tasks of updating the classifiers and determining their weights. By this optimization task, the running time of Heuristic method is not far more than CCP framework method for weighted classifiers method.

Finally, in weighted classifiers method, exact Bayesian and Bayesian methods take the most time. Heuristic and CCP framework method are almost the same for some datasets, but Heuristic takes more time for the others. However, the overall accuracies of Heuristic method are higher than CCP framework method.

4.3.3 Impact of the merging process and sensitivity of the methods to the threshold parameters

In this section, the impact of the merging process and the sensitivity of the batch assignment methods to the threshold parameters are studied. Sensor dataset is used to evaluate the merging process because it is large enough to

Fig. 3 Effect of the merging process on sensor dataset for all batch assignment methods



make the impact of the different parameters clear. In addition, the effect of the merging process becomes clearer for large datasets, because this process resolves the problem of assigning more than one classifier to one concept and this possibility can be more expected for higher number of batches of instances. Parts (a) through (d) of Fig. 3 show the accuracies of the four batch assignment methods for different values of the threshold parameters. In each part, the accuracies of the corresponding method in conjunction with the merging process are compared with the accuracies obtained when the merging process is not used. The parameters used in these experiments are the same as those used in the previous sections for the sensor stream dataset, except that the parameter $maxC$ is set to 40 instead of 10. This change is made so that when the methods are used without the merging process and as a result the probability of assigning more than one classifier to a concept is high, the number of available classifiers would be higher to reduce the impact of this problem.

Using a wide range of values for the threshold parameters, it can be seen that the dependency of the methods to the threshold parameter is low. This is the case both when the merging process is used or not, but this does not mean that the true setting for these parameters is not crucial. Indeed, using very high or low values for the threshold

parameters is not recommended. For example, part (a) of Fig. 3 which shows the accuracies for CCP framework batch assignment method, determines that the method does not work well when the threshold parameter θ is very high. It is obviously expected because all the batches of instances will be assigned to one classifier in this case and no use of recurring concepts is made. In fact, only an incremental learning will be done which is not expected to work well on the concept drifting datasets. Besides, note that although low dependency to the threshold parameter can be seen for CCP batch assignment, this method will have very low accuracy without using the merging process.

The other important result that can be extracted from these figures is that the merging process increases the accuracy substantially for the sensor dataset. Therefore, we suggest that the merging process can be important for other large datasets, too. This result can be obtained by comparing the two curves of the parts (a) through (d) of Fig. 3. The accuracies of the methods with merging process is more than about 60 % ~ 70 % better in comparison with the methods without merging process. In addition, for reasonable threshold parameters (not very low or high) no value of the threshold parameters leads to an accuracy which is even competitive to the accuracies of the methods with the merging process. This means that using the

Fig. 4 Sensitivity of the different methods to the threshold parameters. **a** Pure CCP method. **b–d** The proposed batch assignment methods in conjunction with the merging process and the weighted classifiers method

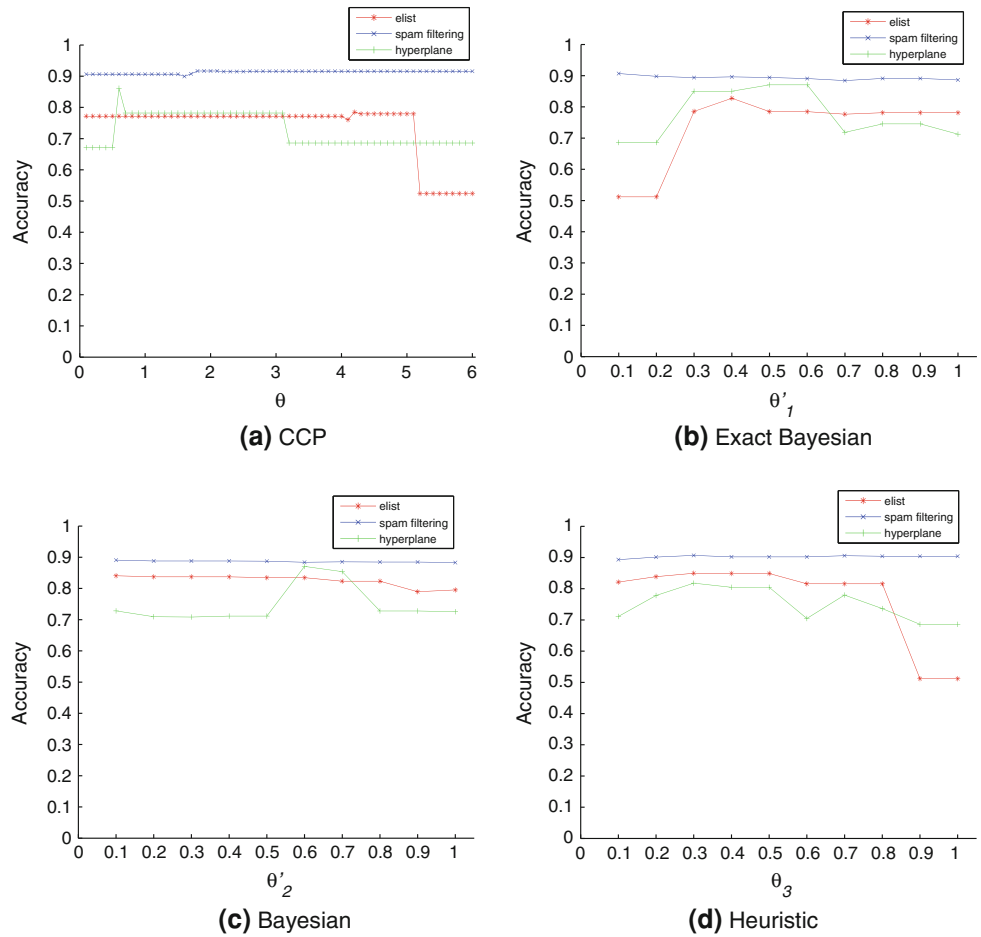
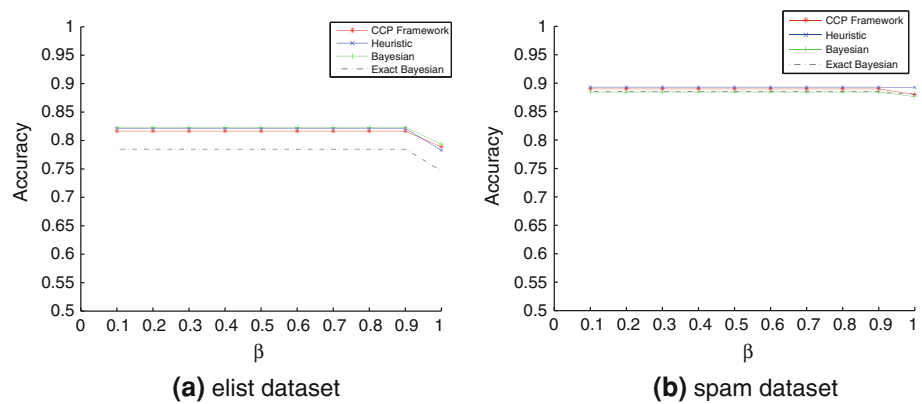


Fig. 5 Accuracies of weighted classifiers method for different values of β for **a** elist and **b** spam datasets



merging process is crucial even if the parameters are tuned very well.

The sensitivity of the threshold parameters for the other datasets can be seen in Fig. 4. Parts (a) through (d) of Fig. 4 shows the accuracies of the different batch assignment methods in conjunction with the merging process and weighted classifiers method for classification. For exact Bayesian and Bayesian methods, θ'_1 and θ'_2 are included instead of θ_1 and θ_2 , where $\theta_i = 2 * \log(\theta'_i) * m$ and m is the batch size. The sensitivity is more than that of the sensor

dataset. The parameters should not be set to very high or low values. However, the parameters of the proposed method can be chosen from a specified range of values.

4.3.4 Sensitivity of weighted classifiers method to parameter β

Parts (a) and (b) of Fig. 5 show the accuracies of weighted classifiers method for different values of the parameter β and on the elist and spam filtering datasets. The

sensitivities of the other two datasets are similar and so they are not discussed. The results are shown for the four batch assignment methods. As it can be seen, the sensitivity of weighted classifiers method to β is very low for both datasets. In fact, only when this parameter is set to 1, the accuracy will be different, but the interval consists of the values in (0,1). The results of weighted classifiers method for the elist dataset which contains sudden concept drift outperform active classifier method, for all values of β . When β is less than 1, the results are even better. For spam filtering dataset which contains gradual concept drift, the results of weighted classifiers method for all values of β and active classifier method are almost the same.

5 Conclusion and future works

We have proposed a method with some variations for data stream classification in the presence of recurring concepts. A pool of classifiers was used in the method which was updated according to the consecutive batches of instances. Each classifier in the pool was representative of a concept. This pool was also used for the classification task. To update the pool, one of its classifiers was selected and updated with each newly arrived batch in a batch assignment method or a new classifier was added to the pool. One new classification method and three batch assignment methods was introduced and compared with the existing methods. In addition, a merging process to merge the classifiers of the pool was introduced. The most similar method to ours is CCP framework method. In order to evaluate our method, we compared it with this method on four datasets with different natures. The experiments showed that our batch assignment, classification and merging methods improve the result on datasets with challenging properties of having arbitrary attributes, containing sudden concept drift and being of large size. In addition, the parameter setting of our method was shown to be simple according to the general properties of the datasets or a specified range of values and almost the same parameters (especially for threshold and β parameters) can be set for different datasets.

Some future research directions related to this work might include the followings: First, more management activities rather than the merging process can be done on the classifiers of the pool to obtain better results even in more dynamic environments and under the constraints of the problem. Second, the batch size parameter can be tuned during the execution of the algorithm according to the tradeoff between larger batch size and the locality assumption for a batch. In addition, it is not necessary to have the same batch size for both classification and updating the pool. This can help us to handle the drift in the

current window more carefully. Third, other similarity measures with lower time complexities or higher performances can be used for the batch assignment task or the merging process.

Acknowledgments The authors should acknowledge from Pooya Samangouei for taking part in the editing process of the paper. This work was supported by Iran Telecommunication Research Center.

References

- Aha DW, Kibler D, Albert MK (1991) Instance-based learning algorithms. *Mach Learn* 6(1):37–66
- Baena-García M et al (2006) Early drift detection method. In: *ECML PKDD workshop on knowledge discovery from data streams*, pp 77–86
- Bifet A et al (2010) Accurate ensembles for data streams: combining restricted Hoeffding trees using stacking. In: *2nd Asian conference on machine learning*, Tokyo, Japan: *JMLR*, pp 225–240
- Bifet A (2009) Adaptive learning and mining for data streams and frequent patterns, PhD Thesis in Departament de Llenguatges i Sistemes Informatics. Universitat Politècnica de Catalunya
- Bifet A, Gavaldà R (2007) Learning from time-changing data with adaptive windowing. In: *SIAM international conference on data mining (SDM'07)*. Minneapolis, Minnesota, pp 443–448
- Castillo G (2006) Adaptive learning algorithms for Bayesian network classifiers, PhD thesis in mathematics. Aveiro University
- Domingos P, Hulten G (2000), Mining high-speed data streams. In: *Proceedings of the 6th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, Boston, pp 71–80
- Elwell R, Polikar R (2011) Incremental learning of concept drift in nonstationary environments. *IEEE Trans Neural Netw* 99: 1517–1531
- Frank A, Asuncion A (2010) UCI machine learning repository. <http://archive.ics.uci.edu/ml>. Accessed Jan 2012
- Freund Y, Schapire RE (1996) Game theory, on-line prediction and boosting. In: *Proceedings of the ninth annual conference on Computational learning theory*. ACM, New York, pp 325–332
- Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. *J Comput Syst Sci* 55(1):119–139
- Gama J, Castillo G (2006) Learning with local drift detection. In: *The proceedings of the advanced data mining and applications*. Springer, Berlin, pp 42–55
- Gama J, Kosina P (2009) Tracking recurring concepts with meta-learners. In: *Proceedings of the 14th Portuguese conference on artificial intelligence: progress in artificial intelligence*. Springer, Aveiro, pp 423–434
- Gama J, Medas P, Rocha R (2004) Forest trees for on-line data. In: *Proceedings of the 2004 ACM symposium on applied computing*. ACM, Nicosia, pp 632–636
- Gama J, Fernandes R, Rocha R (2006) Decision trees for mining data streams. *Intell Data Anal* 10(1):23–45
- Gao J, Fan W, Han J (2007) On appropriate assumptions to mine data streams: analysis and practice. In: *Proceedings of the 2007 seventh IEEE international conference on data mining*, pp 143–159
- Gao J et al (2008) Classifying data streams with skewed class distributions and concept drifts. *IEEE Internet Comput* 12(6): 37–49
- Garnett R (2010) learning from data streams with concept drift, PhD Thesis in engineering science. University of Oxford

- Gomes JB, Menasalvas E, Sousa PAC (2011) Learning recurring concepts from data streams with a context-aware ensemble. In: Proceedings of the 2011 ACM symposium on applied computing, pp 994–999
- Hosseini MJ, Ahmadi Z, Beigy H (2011) Pool and accuracy based stream classification: a new ensemble algorithm on data stream classification using recurring concepts detection. In: Proceedings of the 2011 IEEE eleventh international conference on data mining workshops (ICDMW). IEEE, Vancouver, pp 588–595
- Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: Proceedings of the 7th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, San Francisco, pp 97–106
- Ikonomovska E, Gama J, S. Deroski., (2011) Learning model trees from evolving data streams. *Data Mining Knowl Discov* 23(1): 128–168
- Katakis I, Tsoumakas G, Vlahavas I (2009) Tracking recurring contexts using ensemble classifiers: an application to email filtering. *Knowl Inf Syst* 22(3):371–391
- Klinkenberg R (2004) Learning drifting concepts: example selection vs. example weighting. *Intell Data Anal* 8(3):281–300
- Kolter JZ, Maloof MA (2005) Using additive expert ensembles to cope with concept drift. In: Proceedings of the 22nd international conference on Machine learning. ACM, Bonn, pp 449–456
- Kolter JZ, Maloof MA (2007) Dynamic weighted majority: an ensemble method for drifting concepts. *J Mach Learn Res* 8:2755–2790
- Kuncheva LI, Žliobaitė I (2009) On the window size for classification in changing environments. *Intell Data Anal* 13(6):861–872
- Lazarescu MM (2005) A multi-resolution learning approach to tracking concept drift and recurrent concepts. In: 5th IAPR workshop on pattern recognition in information systems (PRIS). Miami, USA, pp 52–61
- Littlestone N (1988) Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Mach Learn* 2(4):285–318
- Littlestone N, Warmuth MK (1994) The weighted majority algorithm. *Inf Comput* 108(2):212–261
- Minku L, Yao X (2012) DDD: a new ensemble approach for dealing with concept drift. *IEEE Trans Knowl Data Eng* 99:619–633
- Morshedlou H, Barforoush AA (2009) A new history based method to handle the recurring concept shifts in data streams. *World Acad Sci Eng Technol* 58:917–922
- Nishida K (2008) Learning and detecting concept drift. PhD Thesis in information science and technology. Hokkaido University, Hokkaido
- Padovitz A, Loke SW, Zaslavsky A (2004) Towards a theory of context spaces. In: Proceedings of the 2nd IEEE annual conference on pervasive computing and communications workshops, pp 38–42
- Ramamurthy S, Bhatnagar R (2007) Tracking recurrent concept drift in streaming data using ensemble classifiers. In: Proceedings of the sixth international conference on machine learning and applications, pp 404–409
- Schlimmer JC, Richard J, Granger H (1986) Incremental learning from noisy data. *Mach Learn* 1(3):317–354
- Scholz M, Klinkenberg R (2005) An ensemble classifier for drifting concepts. In: Proceedings of the 2nd international workshop on knowledge discovery in data streams, Porto, Portugal, pp 53–64
- Stanley KO (2003) Learning concept drift with a committee of decision trees, Technical Report UT-AI-TR-03-302, Computer Sciences Department, University of Texas
- Street WN, Kim Y (2001) A streaming ensemble algorithm (SEA) for large-scale classification. In: Proceedings of the 7th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, San Francisco, pp 377–382
- Tsymbol A (2004) The problem of concept drift: definitions and related work, Technical Report TCD-CS-2004-15, Computer Science Department, Trinity College Dublin
- Wang H et al (2003) Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the 9th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, Washington, pp 226–235
- Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. *Mach Learn* 23(1):69–101
- Widmer G, Kubat M (1998) Special issue on context sensitivity and concept drift—introduction. *Mach Learn* 32(2):83–84
- Woolam C, Masud MM, Khan L (2009) Lacking labels in the stream: classifying evolving stream data with few labels. In: Proceedings of the 18th international symposium on foundations of intelligent systems. Springer, Prague, pp 552–562
- Zhu X (2010) Stream data mining repository. <http://www.cse.fau.edu/~xqzhu/stream.html>. Accessed Jan 2012
- Zliobaite I (2010) Learning under concept drift: an overview. Vilnius University, Technical report
- Zliobaite I (2010) Adaptive training set formation, PhD Thesis in physical sciences and informatics, Vilnius University