Knowl Inf Syst (2016) 46:567–597
DOI 10.1007/s10115-015-0837-4

REGULAR PAPER

An ensemble of cluster-based classifiers for semi-supervised classification of non-stationary data streams

Mohammad Javad Hosseini[1] · **Ameneh Gholipour**[1] · **Hamid Beigy**[1]

Received: 31 May 2013 / Revised: 26 January 2015 / Accepted: 15 April 2015 /
Published online: 28 April 2015
© Springer-Verlag London 2015

Abstract Recent advances in storage and processing have provided the possibility of automatic gathering of information, which in turn leads to fast and continuous flows of data. The data which are produced and stored in this way are called data streams. Data streams are produced in large size, and much dynamism and have some unique properties which make them applicable to model many real data mining applications. The main challenge of streaming data is the occurrence of concept drift. In addition, regarding the costs of labeling of instances, it is often assumed that only a small fraction of instances are labeled. In this paper, we propose an ensemble algorithm to classify instances of non-stationary data streams in a semi-supervised environment. Furthermore, this method is intended to recognize recurring concept drifts of data streams. In the proposed algorithm, a pool of classifiers is maintained by the algorithm with each classifier being representative of one single concept. At first, a batch of instances is classified by the algorithm. Thereafter, some of these instances are labeled and this partially labeled batch is used to update the classifiers in the pool. This process repeats for consecutive batches of the streams. The main advantage of the algorithm is that it uses unlabeled instances as well as labeled ones in the learning task. Experimental results show the effectiveness of the proposed algorithm over the state-of-the-art methods, in different aspects.

Keywords Semi-supervised learning · Ensemble learning · Data streams · Concept drift · Cluster assumption

✉ Hamid Beigy
beigy@sharif.edu

Mohammad Javad Hosseini
mjhosseini@ce.sharif.edu

Ameneh Gholipour
a_gholipour@ce.sharif.edu

[1] Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

1 Introduction

Streaming data are produced in the form of high-speed infinite flows. Examples include huge amount of daily information gathered by web services, financial transactions of credit cards, data stored by sensors, and network intrusions information [1]. According to three specific characteristics of streaming data, stream mining is distinguished from stationary data mining. First, the very large amount of information makes it impossible to store all of it as a whole in currently available memories. Second, data streams are produced in high speed. This means that they should be processed in real time. Thus, algorithms that require many passes on the whole data to build a classifier may be of no use in these environments. Third, data streams often experience concept drift, i.e. the distribution of data is not stationary and changes over the time. Hence, learning algorithms should be able to make their models consistent with the most recent data. It is important to note that independent and identical distribution (I.I.D) condition is not valid in the streams in which concept drift occurs, but it is rational to assume that sufficiently small size batches of data satisfy the I.I.D condition [17]. Going through a real world example helps make the problem clearer. Suppose that an e-mail service user can label each of his e-mails as *spam* or *ham*, based on his interests in reading them. However, due to large number of e-mails he receives every day, which could be referred to as a *stream* of e-mails, it is rather impossible for him to manage all of them. He labels only a few of them and needs the help of an automatic classification system to detect which e-mails are *spam* and which are *ham*. On the other hand, interests of the user might change over time. For example, during presidential elections, he loves to follow political newsletters receiving via e-mails. After the election, however, the user may not be interested in political news anymore. Hence, those political e-mails that were previously *ham* seem to be *spam* now. Here comes the problem of *concept drift*. The underlying concept based on which the instances are classified, may change for any reason. The labeling system for this e-mail service should be designed in a way that it can handle concept drifts as well.

Drift may occur in four structural forms of sudden, gradual, incremental, and recurring, as shown in Fig. 1. In sudden concept drift, the distribution of data changes instantaneously. It is the most simply detectable among other types of concept drifts. In gradual concept drift, there is a non-deterministic stage in which data are driven from two different distributions. In this non-deterministic period, the distribution of data cannot be identified. As time passes, the probability of sampling from one distribution decreases while the probability of sampling from the other one increases. There is a generalized form of gradual concept drift, also referred to as incremental or stepwise drift. In this type of drift, data are derived from more than two distributions; however, there are slight differences between them. In fact, drift is noticeable only if longer time period of observation is considered. Last type of concept drift is called recurring concept. In this type, it is potential that the distribution of data reoccurs after an unknown while [41]. Returning to the example of e-mail classification, it is probable that for next presidential elections, the user again shows an interest in reading political news. Therefore, in addition to detecting concept drifts, the learner should cope with the difficulties of detecting and learning recurring concepts as well. In overall, detecting drift is a challenging task, especially for that sometimes it cannot be distinguished from noise.

Another challenge in dealing with data streams is that in most real cases, not all of the data are labeled. In practice, only a small proportion of data is likely to be labeled while the majority is left unlabeled. This is due to the fact that labeling process is expensive and time consuming. It requires experts' supervision, special equipment or costly experiments. The great speed of coming data makes it even more difficult to label all data in a stream [37,40]. For instance, in our e-mail classification example, most of the instances are left

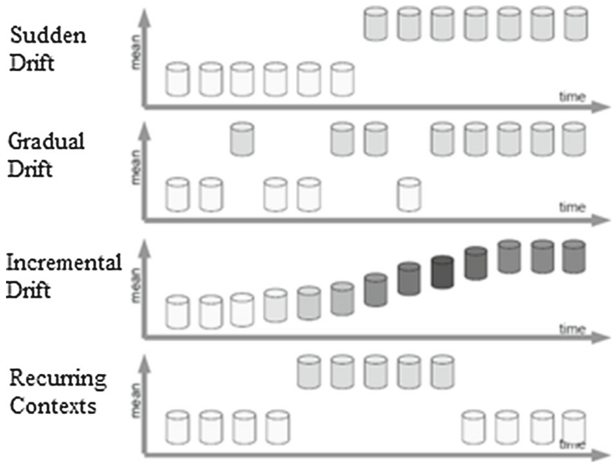

Fig. 1 Four structural types of concept drift [41]

unlabeled because of the large number of e-mails and limited time of the user. So, the task of the classification system is to predict the label of endlessly coming new e-mails based on a model that is trained on partially labeled ones. This is known as semi-supervised learning task and techniques developed for learning under such circumstances are called semi-supervised learning methods. There are only a few stream classification algorithms that are semi-supervised, i.e. capable of appropriate use of unlabeled instances along with labeled ones in the learning process [28]. Algorithms that cope with the evolutionary characteristics of data streams can be generally divided into two main groups, based on the number of classifiers they use. First group, single model classification techniques, generally works by incrementally updating the trained model upon receiving new data. Second group, namely ensemble methods, works by either iteratively constructing base classifiers, which can be produced by weak learners, and then adding them to the current ensemble or updating the existing classifiers of their ensemble. It is well known that such algorithms are performing gradient descent of an error function [3]. In stationary learning of data, ensemble learning is advantageous mostly because it is able to boost weak learners that may be only slightly better than random prediction to strong learners with very high accuracy. Generally, an ensemble method acts effectively if the base learners are diverse so that all types of data can be covered by them [39]. In streaming data, ensemble learning is more applicable since different classifiers can be used to describe different concepts of the environment. Unlike single model methods, ensemble approaches often do not require complex operations and thus have the ability to adapt to changes in speedy data streams. This is why they are highly capable of being exploited in these environments; however, dealing with numerous classifiers may decrease the speed of learning.

In this paper, we propose a new algorithm called *semi-supervised pool and accuracy-based stream classification* (SPASC), or simply SPASC, for non-stationary learning of data streams using ensemble of classifiers. As stated, the reason behind using ensemble learning approach is that ensemble methods have the ability to be updated efficiently, so they can be easily adapted to the changes in the stream [27]. The SPASC algorithm maintains a pool of classifiers in which every classifier determines a specific concept. A chunk of the stream that can be stored in memory is called a *batch*. Each instance in an upcoming batch is firstly classified by the pool of classifiers. After classifying an instance, its true label may be revealed to the algorithm. Then, the batch along with its existing true labels is exploited to update the pool. In updating process, the most similar classifier to the underlying concept of the given batch is identified. If the similarity is greater than a predefined threshold, then the classifier will be updated according to the members of the batch. Otherwise, a new classifier will be trained and added to the pool. The overall infrastructure of the algorithm is similar to PASC algorithm [19]; however, several modifications are performed to make it feasible for semi-supervised environment. In addition, a classifier named cluster-based classifier is proposed to cope with semi-supervised stream of data. In this setting, base classifiers of the pool are cluster based, i.e. each classifier maintains its concept in the form of a number of clusters. Using these classifiers instead of the base classifier of PASC helps to exploit unlabeled instances along with labeled ones in the classification task.

One major contribution of SPASC is its ability of detecting recurring concepts using ensemble of classifiers in non-stationary data streams. This type of drift can exist simultaneously with other types of drift. If we investigate recurring concepts of data streams, results can improved significantly, since adaptation to concept drift can be achieved more quickly; nevertheless, most of stream classification algorithms ignore intrinsic recurring concepts of datasets. An ensemble classifier is often a good choice while dealing with recurring concepts, because each classifier in the ensemble can be used to describe one concept of the environment. SPASC applies ensemble techniques to the problem of learning of semi-supervised time-changing data streams. Using an ensemble of classifiers, SPASC is capable of dealing with virtual and real concept drifts, since it can use a new classifier to handle a new concept, no matter the type of the drift is virtual or real. To the best of our knowledge, there is no such algorithm that exploits the detection of recurring concepts using ensemble methods for classification of non-stationary data streams. However, there exits some semi-supervised algorithms that use single classifier method to detect recurring concepts or ensemble classification of data streams. For example, the algorithm given in [25] build a decision tree for the task of detecting recurring concepts in semi-supervised datasets and the algorithm given in [28] uses ensemble techniques for semi-supervised classification of data streams; however, it is incapable of detecting recurring concepts.

Furthermore, base classifiers in SPASC are cluster based. It means that each classifier maintains a set of clusters that are formed based on received labeled and unlabeled data and will be updated continuously. Exploiting the cluster assumption for semi-supervised data [7], these clusters tend to have high intra-cluster similarity and inter-cluster dissimilarity besides being relatively pure according to the labels of their instances. They are updated by expectation maximization (EM) algorithm. The derivation of the algorithm leads to a simple and efficient updating rule, similar to that of K-means algorithm.

Last but not least, experimental results show the practicality of the work and its ability to be used in empirical cases. The algorithm is evaluated on a number of standard streaming datasets and compared to state-of-the-art methods. Comparisons show that SPASC outperforms other methods on all used datasets. Moreover, it is shown via experiments that the accuracy of SPASC does not deteriorate considerably when the ratio of unlabeled data to the whole data becomes low. This leads to the conclusion that SPASC effectively uses unlabeled data in the learning process.

The rest of the paper is organized as follows. Related works are reviewed in Sect. 2. The semi-supervised proposed algorithm including the cluster-based classifiers is described in Sect. 3. In Sect. 4, experimental results are reported. Finally, Sect. 5 concludes the discussion and introduces some future work.

2 Related work

In the context of learning data streams, some proposed algorithms are capable of dealing with gradual concept drift [23], some can handle abrupt concept drift [4,6,12,13,31], and some have the potential to cope with both types [36]. However, most of these methods are appropriate only for supervised environments in which the labels of data are fully known. Some single model classification techniques for data streams are proposed in [9,15,20,21]. Because they are building incrementally, they usually utilize only the most recent data. This means that the previous data are forgotten despite of the fact that they might still be consistent with the current or future concept. This has negative influence on accuracy of the refined model. Ensemble methods, on the other hand, have shown higher accuracy compared with single models and are discussed in [10,14,22,24,33–35,37]. All these algorithms are related to our work in that they try to build an ensemble of classifiers in order to learn data streams.

As pointed out previously, most real cases of data streams have only a small portion of their data labeled, while the majority is unlabeled. A diverse range of methods, known as semi-supervised learning, have been suggested to make the most of unlabeled data as well as exploiting labeled ones [27,38]. Two techniques have been widely used in semi-supervised classification of non-stationary data streams: EM and predicting the labels of unlabeled instances. In the former, when a concept drift is detected, using an EM algorithm, labels of unlabeled instances are tuned in the expectation step and the model is updated in the Maximization step. In the latter, labels of unlabeled instances are estimated first and then, they are used in updating the model along with labeled ones. One group of methods in this category is clustering algorithms that identify label of an instance based on the cluster in which it falls.The method proposed in [11] is an example of a semi-supervised method for drifting data streams using a decision tree. When a new batch is received, the amount of loss of the tree on the batch is calculated. If the loss is greater than an experimentally set threshold, the tree is updated to decrease the loss amount. A subset of unlabeled instances then is selected and their label is estimated. These instances are used to update the leaves of the tree. Updating process may lead either the leaf to extend or its class distributions to change.

In [27], a method called “semi-supervised stream clustering” denoted by “SmSCluster” is proposed. The streaming data are split into chunks and a classification model is formed for each chunk. A semi-supervised clustering algorithm is introduced to create K clusters from the partially labeled training data. The clustering algorithm uses EM to produce clusters with both minimizing intra-cluster dispersion and at the same time the impurity of each cluster regarding its labels. A summary of the statistics of the instances belonging to each cluster is saved as a “micro-cluster”. The micro-clusters are served as a classification model. Classification is performed with K-nearest neighbor algorithm, which identifies Q-nearest clusters. The most frequent label in these clusters will be the predicted label of an instance. In order to cope with the stream evolution, an ensemble of L such models is used. Whenever a new model is built from a new data chunk, we update the ensemble by choosing the best L models from the $L + 1$ models (previous L models and the new model), based on their individual accuracies on the labeled training data of the new data chunk.

In [28], the ReaSC method is proposed. We have used this method for our comparison. It is an extension of SmSCluster with some improvements. In SmSCluster, it is assumed that labeled instances are randomly distributed in the stream, while ReaSC does not require this assumption. In both methods, an ensemble of models is used to label instances. When at least P % of instances are classified, they are used to generate K clusters using a constraintbased clustering algorithm. After the clustering procedure is done, instances are discarded and information about clusters is stored in micro-cluster format. Finally, labeled micro-clusters are used to label others. This new model is added to the ensemble to amend it. The number of classifiers in the pool is a constant number in order to elegantly address the infinite length problem. This algorithm and SPASC are similar in that they both use cluster-based classifiers and use EM algorithm to exploit unlabeled instances along with labeled ones. They also both try to produce clusters that are pure according to their labels. This is done with different formulations and assumptions. In ReaSC, K-means formulation is extended to have a term for cluster impurity, while our method starts from assuming Gaussian distribution for data and also some assumptions about the labels of each cluster. However, the most important difference is that in SPASC, the recurring concepts are taken into account. The reason that we maintain and use previously identified concepts is that in many real applications, maintaining models just based on their performance on the last chunk of data will have two disadvantages: (a) It is sensitive to noise in data, (b) it will gradually forget previously seen data and cannot cope with concepts that reoccur over time.

Also in [8,29], semi-supervised classification of streams is done using ensemble methods. In these approaches, labeled data are employed to build classifiers and unlabeled data are used to determine classifiers voting weights. In [2], an ensemble learning method is proposed. The majority vote of the classifiers is used to label the unlabeled instances. It is shown that their classifier is PAC learnable, even if the labeling process is noisy.

While some forms of concept drift such as gradual or abrupt drifts have been extensively explored, research on recurring concepts has gained attention only recently and much of the work is done for supervised classification of data streams. A window-based framework, called conceptual clustering and prediction (CCP), for detecting recurring concepts is proposed in [22]. It extracts a conceptual vector for each window using a transformation function. The algorithm preserves a classifier for each concept in a pool which is updated as time passes. A clustering algorithm is used to detect recurring concepts. Clustering is done on conceptual vectors such that if the distance of a new conceptual vector to the conceptual vector of its nearest concept available in the pool is less than a threshold or the pool is full and does not have additional space, the classifier corresponding to that concept will be updated; otherwise, a new classifier and concept will be created. Euclidean distance between the conceptual vectors is used as the difference measure. One major shortcoming of this framework, however, is how to determine the threshold, since it is a problem specific parameter and should be tuned through trial and error. PASC [17,19] is an extension of the CCP framework method. It introduces some new similarity measures. It also decreases the dependency on the threshold of the CCP framework method and handles larger data sets. PMRCD also uses similar approach to CCP. This method maintains a pool of classifiers and manages it by merge and split operations so that there is no need to set the threshold parameter for this method. One other research [32] inspires context space model to extract concepts for each classifier [16]. N-tuple of form $R = \left(a_1^R, a_2^R, \ldots, a_N^R\right)$ is a context space, where a_i^R shows the acceptable regions of feature ai. The classifier and its corresponding context space are maintained in the pool. However, all these ensemble methods to detect recurring concepts are only applicable to fully labeled data streams and cannot handle semi-supervised datasets.

REDLLA [25] is an algorithm for semi-supervised classification of data streams with recurring concept drifts and limited labeled data. It grows a decision tree of which each leaf uses a clustering algorithm based on K-means to produce concept clusters. Then, the majority-class method is used to label unlabeled data. All concept clusters generated over the previous data chunks are maintained in a set of history concept clusters. In order to detect recurring concepts, the deviation between two concept clusters is measured based on their radius and distance. If $r_{\text{new}}, r_{\text{last}}$ show the radius of new cluster and previous one, respectively, and dist(a, b) denotes the distance between their centers, then condition dist $(a, b) > r_a + r_b$ shows a concept drift. If the deviation between this new cluster and the clusters in history is less than a threshold, it is a recurring concept. One problem with REDLLA is that it only considers numerical attributes and cannot handle categorical or mixed attributes.

3 The proposed semi-supervised learning algorithm

In this section, a semi-supervised classification algorithm for classification of data streams is proposed. The proposed algorithm is based on ensemble classification approach and is called SPASC. The goal of SPASC is to classify non-stationary data streams when only a small portion of data is labeled. It is based on detection of recurring concepts and tries to exploit the information about the previously known concepts if they are repeated throughout the stream. The algorithm keeps a limited number of classifiers in a set, referred to as pool, for its classification task. In SPASC algorithm, it is assumed that instances of data are received and classified in the form of a stream. For instance, in the e-mail classification example, which was provided in the introduction, e-mails are received and classified continuously. After classification of each instance, a user may reveal its true label to the algorithm. In the e-mail example, a user may send a feedback after reading an e-mail and specify whether it is spam or ham. For example, after reading a spam e-mail, the user may state that this instance has been classified incorrectly.

The pseudo-code of SPASC is given in Procedure 1.The overall framework of SPASC algorithm is as follows: instances of data arrive continuously in the form of batches. It is assumed that the stream is divided into equal and fixed size batches of data. The algorithm consists of two phases. In phase one (line 4 of the pseudo-code and will be discussed in Sect. 3.1), each instance in the received batch is classified by using the pool of classifiers. We use an adaptive weighted scheme for classification. Once the classification of one instance is done, an expert user probably reveals its true label to the algorithm. Those labels are used for adapting the weights of classifiers and also will form a partially labeled batch as an input to the second phase. Phase two (line 5 of the pseudo-code and will be discussed in Sect. 3.2) involves updating the pool using this partially labeled batch. This update is done either by updating one of the pre-existing classifiers or adding a new classifier, which is built based on the new batch.

In the proposed method, we need a classifier that can handle partially labeled data. In particular, the classifier needs to be updatable based on partially labeled instances. We have proposed a classifier for our purpose and will be discussed in Sect. 3.3. The proposed classifier maintains a number of clusters. Each cluster is consisted of labeled and unlabeled data. These clusters tend to be pure as possible, i.e. containing instances with the same label. Also they will be formed such that they have the highest intra-cluster similarity and inter-cluster dissimilarity. Clusters of the classifier are built and updated using EM algorithm.

Concretely, the algorithm is provided by $B_t = \left(x_1^t, x_2^t, \ldots, x_l^t\right)$ as labeled instances such that x_i^t $(1 \leq i \leq l)$ is ith instance in the tth batch with $L_t = \left(l_1^t, l_2^t, \ldots, l_l^t\right)$ showing their labels, respectively, and $B_t' = (x_{l+1}^t, x_{l+2}^t, \ldots, x_k^t)$ as unlabeled instances, where k is the size of batch. In order to simplify notations, it is assumed that all labeled instances appear prior to unlabeled ones. This assumption has no effect on accuracy of the algorithm and can be removed in practice.

```
   Input: an infinite stream of instances: $B_t$ , $B'_t$ , $L_t$
   Output: Predicted labels of instances $x_i^t$ $(1 \le i \le k, t \ge 1)$
1  Initialization: Pool = ∅; $B_1$, $B'_1$, $L_1$ = read_next_batch(); $C_1$ = make_classifier($B_1$ , $B'_1$ , $L_1$); $w_1$ = 1; Pool = Pool ∪ {$C_1$};
2  while (true)
3      $B_t$, $B'_t$, $L_t$ = read_next_batch(); // each $l_i^t$ for labeled instances will be revealed just after classification
4      Pool.classify($B_t$ , $B'_t$);        //The output of algorithm
5      Pool.update($B_t$ , $B'_t$ , $L_t$);
6  end while
```

Procedure 1 – The main framework of SPASC

The algorithm starts with some initializations: an empty pool is created; the first batch of data is received; a cluster-based classifier, called C_1 is built with the first batch; the weight of this classifier, denoted by w_1, is set to 1; C_1 is added to the empty pool. As mentioned earlier, there is an iterative process at the heart of the algorithm containing two major phases. Lines 4 and 5 show these phases. The two phases of the algorithm are discussed in Sects. 3.1 and 3.2. While going through these phases, the details of cluster based classifiers are not required. Hence, we postpone the discussion about this classifier to Sect. 3.3. Finally, in Sect. 3.4, we bring a toy example of the execution of the proposed method.

3.1 Phase 1: classifying the batch

Classification of a batch is done by classifying its instances one after another. For classification task, a classifier from the pool should be selected. In order to be able to adjust to potential concept drifts, an adaptive approach is suggested to select the most suitable classifier from the pool. In this approach, a positive weight in interval (0,1] is assigned to each classifier. These weights are computed according to the performance of the classifiers on each instance of a batch and being used for classification of the next instance in that batch. Furthermore, at the end of each iteration, these weights are updated in order to be used in the next iteration (updating weights at the end of iterations will be discussed in Sect. 3.2). For the first iteration, the weight of the first classifier of the pool is initialized to 1.

For ith instance in the tth batch denoted by x_i^t, the classifier with the maximum associated weight is selected and classification task is done. If x_i^t is an unlabeled instance, it would not have any role in updating classifiers weights and the algorithm continues with classifying the next instance, x_{i+1}^t, but if x_i^t is a labeled instance, its true label is revealed right after classification and the instances along with their true labels is used to update classifiers weights. For this task, x_i^t is given not only to the classifier that has classified it but also to all other classifiers of the pool to see whether they classify it correctly or not. Based on the accuracy of each classifier on instance x_i^t, its weight will be updated. The updated weights are used for classifying the next instance denoted by x_{i+1}^t.

Figure 2 shows a sample scenario of classifying a batch. Upon receiving instance x_i^t, selector selects the classifier in the pool that has the maximum weight. The SPASC classifier predicts the label of the received instance. Besides, this predicted label along with the true label of x_i^t (if any), which is given by the expert user, is used to update the weights of classifiers. The updating rule aims to increase the chance of selection of classifiers that predict the labels of labeled instances correctly. These classifiers tend to describe the current concepts of the environment from which new instances are produced. The rule for updating the weights is:

$$w'_j = F\left(w_j, M\left(j, x_i^t\right)\right), \tag{1}$$

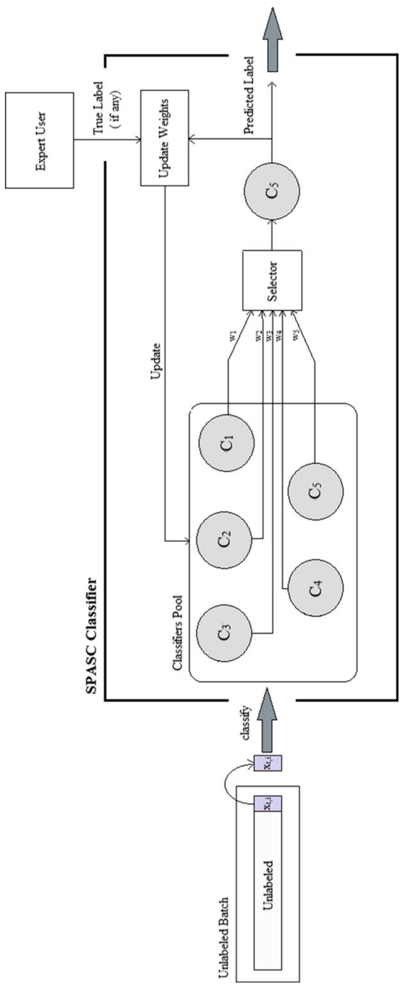

Fig. 2 Phase 1 of SPASC : classifying a new batch of data

where w_j and w'_j are the current and new weights of the jth classifier, respectively, and $M(j, x_i^t)$ will be 0 if jth classifier classifies the instance x_i^t correctly and will be 1 otherwise. Function F could be any function that outputs higher values if the classifier classifies the instance correctly than the case of incorrect output. We have used the following function in our experiments:

$$F\left(w_j, M\left(j, x_i^t\right)\right) = w_j \times \beta^{M\left(j, x_i^t\right)}, \tag{2}$$

where β is a constant parameter in interval [0,1). It is shown that this function will produce near optimal classification results [17]. The algorithm for updating weights of classifiers is shown in Procedure 2.

```
for i=1 to k do
  predicted_label$_i^t$ = Pool.classify(x$_i^t$);
  if x$_i^t$ is labeled then
    for j=1 to Pool.size() do
        M(j, x$_i^t$) = 1(predicted_label$_i^t$= true_label$_i^t$)
        wj = F(wj, M(j, x$_i^t$)) ;
    end for
  end if
end for
```

Procedure 2 - Updating classifiers weights

3.2 Phase 2: updating the pool

When classification of a batch is finished and the labels of all labeled instances are received, the partially labeled batch will be used to update the pool. The pool contains a number of classifiers, each of them describing one single concept. The number of allowed classifiers in the pool are limited and controlled by parameter *maxC*. The newly arrived batch is either driven from a concept of one of the pre-existing classifiers or is described by a new concept. Hence, as Fig. 3 shows, updating the pool is done either by (a) updating one of the pre-existing classifiers or (b) adding a new classifier built based on the new batch. In order to determine the classifier that will be updated in case (a), a method called batch assignment is proposed

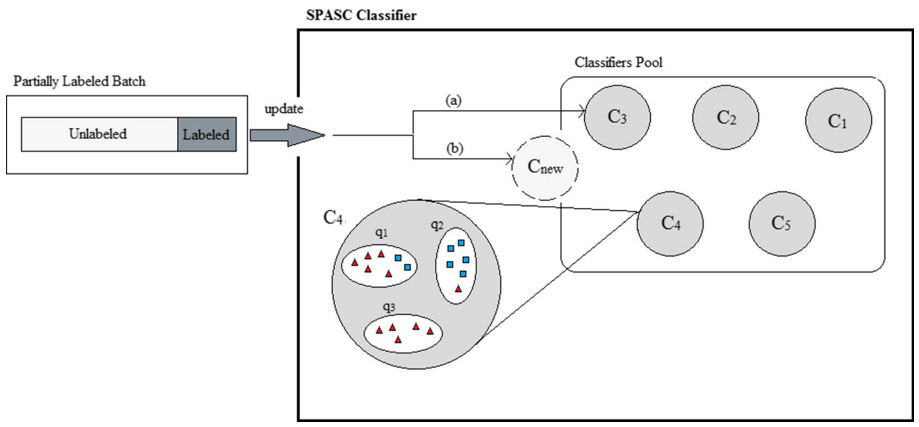

Fig. 3 Phase 2 of SPASC: updating the classifiers pool

which will be discussed later in this section. In the proposed batch assignment method, a similarity measure is defined between a batch of instances and an existing classifier. The classifier that has the highest similarity to the batch is selected with the batch assignment method.

The overall algorithm for updating the pool is given in Procedure 3. As a new batch arrives, the most similar classifier of the pool, namely *best_classifier*, is identified in line 1. If the similarity between the most similar classifier to the current batch, denoted by *max_similarity*, is higher than a given threshold θ_1 or the pool is full, the classifier will be updated in line 3. Otherwise, a new classifier is built and added to the pool as shown in lines 5 and 6. We propose two semi-supervised batch assignment methods: *semi-Bayesian* and *semi-heuristic*. These batch assignment methods are explained in Sects. 3.2.1 and 3.2.2, respectively. Finally, after updating an existing classifier or adding a new one, the weights of the classifier should be updated before performing the next iteration. This is done in lines 8–11 and is discussed in Sect. 3.2.3.

```
1   (max_similarity, best_classifier) = Pool.assign_batch($B_t$, $B'_t$, $L_t$);
2   if max_similarity > $\theta_1$ or Pool.size() = maxC then
3       best_classifier.update($B_t$, $B'_t$, $L_t$);
4   else
5       C = make_classifier($B_t$, $B'_t$, $L_t$);
6       Pool = Pool ∪ {C};
7   end if
8   for j=1 to Pool.size() do
9       classifier_error= Pool[j].error($B_t$, $L_t$);
10      $w_j$= $\beta^{2^{classifier\_error}}$
11  end for
```

Procedure 3 - Semi-supervised method for updating classifiers pool

3.2.1 Semi-supervised Bayesian method for batch assignment

In this batch assignment method, the similarity measure between a batch and the concept of ith classifier, C_i, is defined as the probability that the concept from which the batch is derived is the same as the concept of that classifier, called h_i. In other words, it is the probability that h_i describes B_t, B'_t and L_t:

$$P\left(h_i|B_t, B'_t, L_t\right)$$

We compute this probability with two simplifying assumptions:

Assumption 1 The probability that an instance x takes a label l given x is drawn from an arbitrary concept h_i is independent of other instances driven from h_i and their labels. This means that

$$P\left(l|x, B, B', L, h_i\right) = p(l|x, h_i), \tag{3}$$

where B, B' and L refers to an arbitrary partially labeled batch.

Assumption 2 The probability that an instance x is driven from an arbitrary concept h_i is independent of other instances that are driven from h_i.

$$P\left(x|B, h_i\right) = P\left(x|h_i\right), \tag{4}$$

where B refers to an arbitrary batch of instances.

According to the Bayes Theorem, the probability that hypothesis h_i describes the concept of B_t, B_t' and L_t is:

$$P\left(h_i \mid B_t,\ B_t',\ L_t\right) = \frac{P\left(B_t,\ B_t',\ L_t \mid h_i\right)\ \times P\left(h_i\right)}{P\left(B_t,\ B_t',\ L_t\right)} \tag{5}$$

So the best concept, which we call it h^*, is the one that maximizes the numerator of the right-hand side, because the best concept is independent of the probabilities of B_t, B_t' and L_t. Thus we have

$$h^* = \text{argmax}_i P\left(h_i \mid B_t,\ B_t',\ L_t\right) = \text{argmax}_i \left(P\left(B_t,\ B_t',\ L_t \mid h_i\right) \times P\left(h_i\right)\right). \tag{6}$$

Since we do not have any prior knowledge about different concepts, we use the uniform distribution for h_i, hence $P(h_i) = P(h_j)$ for all i and j and hence h^* will become:

$$h^* = \text{argmax}_i P\left(h_i \mid B_t,\ B_t',\ L_t\right) = \text{argmax}_i P\left(B_t,\ B_t',\ L_t \mid h_i\right) \tag{7}$$

So the goal is to calculate $P\left(B_t,\ B_t',\ L_t \mid h_i\right)$ which is equal to the following equation:

$$P\left(B_t,\ B_t',\ L_t \mid h_i\right) = P(L_t \mid B_t,\ B_t', h_i) \times P\left(B_t,\ B_t' \mid h_i\right) \tag{8}$$

Using Assumption 1, we have:

$$P(L_t \mid B_t,\ B_t', h_i) = P(L_t \mid B_t, h_i) \tag{9}$$

and thus, Eq. (8) can be rewritten as:

$$P\left(B_t,\ B_t',\ L_t \mid h_i\right) = P(L_t \mid B_t, h_i)\ \times P\left(B_t,\ B_t' \mid h_i\right). \tag{10}$$

As a result, the two probabilities on the right-hand side of the Eq. (10) should be estimated. Following the Assumption 1, the first term can be estimated easily:

$$P(L_t|B_t, h_i) = \prod_{j=1}^{l} P(l_j^t|x_j^t, h_i) \tag{11}$$

The term $P(l_j^t|x_j^t, h_i)$ shows the probability that the label of instance x_j^t equals to l_j^t and can be calculated using the ith classifier. In the rest of this section, we will show how to estimate the second term of equation (10), i.e., $P\left(B_t,\ B_t' \mid h_i\right)$. Considering Assumption 2, it can be inferred that:

$$P(B_t,\ B_t'|\ h_i) = \prod_{j=1}^{k} P(x_j^t|\ h_i). \tag{12}$$

$P(x_j^t|h_i)$ can be rewritten as:

$$P(x_j^t|h_i) = \frac{P\left(h_i|x_j^t\right) \times P\left(x_j^t\right)}{P\left(h_i\right)}. \tag{13}$$

The goal of this estimation is to determine the best concept h^* which is independent of $P\left(x_j^t\right)$ $(1 \leq j \leq k)$. In addition, $P\left(h_i\right)$ is assumed to be equal for all concepts. Therefore, we can use Eq. (14) instead of Eq. (12):

$$P(B_t,\ B_t'|\ h_i) \propto \prod_{j=1}^{k} P\left(h_i|x_j^t\right). \tag{14}$$

In order to calculate the posterior probabilities of concepts h_i given instances, we create and use a classifier named *raw data classifier* (RDC). RDC predicts that with what probability one batch of instances is associated with one concept.

Each batch of data is assumed to be one unit and serves as one training instance for this classifier. The label of each instance (i.e. one batch) is the index of the most related concept to that instance. RDC receives all instances of a batch, whether or not they are labeled, along with the index of the concept that describes that batch. It is assumed that all members of a batch are derived from one single concept, i.e. the concept drift does not occur in batches. This describing concept has been determined through the batch assignment method, which assigns the batch to the most similar classifier (equally, the most similar concept) in the pool. As we know each classifier is associated with exactly one concept, the concept related to that classifier is the one describing the batch. For example, if a batch containing B_t and B'_t is assigned to h_i, *RDC* should be trained by all instances x^t_j, $1 \le j \le k$ with index i as their label. For this purpose, it is assumed that $P\left(h_i|x^t_j\right)$ is equal for all instances x^t_j and we call it p^t_i. Then, the batch is converted into a vector and this vector is given to *RDC* as training instance with the index i as its label. The equality assumption for $P\left(h_i|x^t_j\right)$ and also using one vector to describe the whole batch is done in order to increase the efficiency of the method. We use the average value of the features of all instances in B_t and B'_t as the desired vector describing the whole batch. Finally, the posterior probability of class i of *RDC* is used to estimatep^t_i. Therefore, we can use the following as estimation for the second term of Eq. (10).

$$P\left(B_t, B'_t|h_i\right) \propto P\left(h_i|B_t, B'_t\right) \propto {p^t_i}^k \tag{15}$$

Substituting (11) and (15) into (10) gives:

$$h^* = \text{argmax}_i P\left(h_i|\ B_t,\ B'_t,\ L_t\right) = \text{argmax}_i \left(\prod_{j=1}^{l} P(l_{t,j}|x_{t,j}, h_i) \times {p^t_i}^k\right) \tag{16}$$

It is straightforward to conclude that:

$$h^* = \text{argmax}_i \left(\sum_{j=1}^{l} \log P\left(l^t_j|x^t_j, h_i\right) + \left(k \times \log p^t_i\right)\right) \tag{17}$$

Finally, the best classifier can be obtained by (10) and algorithm of semi-Bayesian method for updating the pool, using a new batch $(B_t,\ B'_t,\ L_t)$, can be seen in Procedure 4.

```
$X_t$ = avg_vector ($B_t$, $B'_t$);
for $i$=1 to Pool.size() do
    $probability_i$ = RDC.predict($X_t$, Pool[$i$]);
    $similarity_i$ = semi_Bayesian_similarity($probability_i$, $B_t$, $L_t$);
end for
assigned_classifier = $\text{argmax}_{i:1..\text{Pool.size()}}$ ($similarity_i$);
RDC.update ($X_t$, assigned_classifier);
```

Procedure 4 – Semi-supervised Bayesian method for batch assignment

3.2.2 Semi-supervised heuristic method for batch assignment

In this approach, the similarity measure between a batch of instances and a classifier of the pool is defined as the accuracy of that classifier on the labeled instances of that batch.

Therefore, the accuracy of all classifiers in the pool on B_t and L_t is calculated and the classifier with the highest accuracy is determined. If the accuracy of this classifier is more than a predefined threshold, denoted by θ_2, then it is selected for update; otherwise, a new classifier will be built and added to the pool. The reason is that the higher the accuracy of a classifier on a batch is, the more probable the classifier is related to that batch. In other words, intuitively a classifier or concept that describes a batch will have high accuracy on it. It is important to note that the accuracy can be computed only for the labeled data, so the unlabeled portion of the batch, B'_t, is not used in this method.

In order to have a comparison between the heuristic method and the Bayesian method described in the previous subsection, we should note that the heuristic method is similar to the first term of Eq. (17), i.e. the summation on the logarithm of the posterior probabilities of classification. However, it does not use the logarithm and the probabilities are either 0 or 1. Therefore, the main difference is that the Bayesian method uses the posterior probabilities instead of just the classification results and it also uses the unlabeled instances to assign the batch

3.2.3 Updating the classifiers weights

As it is aforementioned, the algorithm is iterative. At the end of iterations, the weights of the classifiers are required to be tuned for the next iteration. While classifying a new batch, if all classifiers have equal weights, then proper classifier may not be chosen for the first instances of that batch until some classifiers make enough mistakes so that their weights are decreased. On the other hand, if no tuning is done for the next batch of instances, weights could be very low for some classifiers as they may not be the correct classifier for a long time. This way, reaction to concept drifts will not be as fast as required. Hence, we suggest a tuning process to alleviate this problem. For this purpose, labeled instances, B_t, L_t, are used to tune the weights according to last four lines of Procedure 3. As stated in Sect. 3.1, β is a constant parameter in the interval [0,1). The reason behind the formula in line 10 of this procedure is that classifiers with higher errors should be given lower weights at the beginning of the next iteration. We use this equation in our experiments; nevertheless, any other equation with similar properties can be used as well.

3.3 Cluster-based classifiers

In this section, we propose a method for building the base classifiers of the pool, namely cluster-based classifiers. Each of these classifiers is firstly trained by a single batch of instances. Afterward, they are updated given new batches of data. In the following subsections, the overall structure of the classifiers, the approach for constructing a cluster-based classifier from a single batch of instances, incrementally updating the classifier according to new batches of instances, and finally classifying instances and finding posterior probability distributions for them are discussed.

3.3.1 The overall structure of cluster-based classifier

Each classifier is consisted of Q clusters, where Q is a parameter of the algorithm. Partially labeled batches of instances are used to build and update these classifiers. In each cluster, there are both labeled and unlabeled data. The final purpose of constructing these clusters is to obtain clusters that are pure according to labels of their instances. This means that it is preferred that most instances of each cluster have the same label. Unlabeled instances are exploited to shape clusters more accurately. Constructing and updating of clusters of a classifier are done based on the two following assumptions regarding clusters and their instances:

Assumption 3 (*Distribution of data within a cluster*) The data which fall into the qth cluster are assumed to be derived from a Normal distribution with the mean of μ_q and covariance matrix of $\sum_q = \varepsilon I_n$, where I_n is an n-dimensional identity matrix and $\varepsilon \in \mathbb{R}^+$ and $\varepsilon \to 0$. This condition on ε leads to intra-cluster similarity and inter-cluster dissimilarity. This assumption is what is exactly done in K-means algorithm [26].

Assumption 4 (*Distribution of labels*) The probability that an unlabeled instance, x_i in cluster q, gets label c_j is given by:

$$P(l_i = c_j | x_i \in q) = \begin{cases} p_{i,j,q} & \text{if } c_j = \text{argmax}_{c'_j} \left(NL_{q,c'_j} \right) \\ \delta \in \mathbb{R}^+ & \text{otherwise} \end{cases}, \tag{18}$$

such that $\delta \ll \varepsilon$ and $p_{i,j,q}$ is a positive real number. NL_{q,c_j} shows the number of instances in cluster q that have the label c_j. The sum of the probabilities $P(l_i = c_j | x_i \in q)$ for all possible values of c_j should be equal to 1. Therefore from the condition that $\delta \to 0$, it can be inferred that $p_{i,j,q} \to 1$. The supportive reason behind this assumption is to obtain pure clusters during constructing and updating clusters.

3.3.2 The approach for constructing a cluster-based classifier from a single batch of instances

In what follows, we discuss how cluster-based classifiers are trained with a single batch of instances. Training a classifier means building and updating its data clusters. Suppose a batch of data containing B_t, B'_t and L_t is received. The aim of our training algorithm is to build Q clusters based on B_t, B'_t and L_t. In order to build clusters, we need to first determine the cluster that each instance belongs to and then compute the means of all clusters; however, neither the assignment of instances to clusters nor the means of clusters are known. Hence, the problem is to determine the means of Q clusters based on some known variables, i.e. B_t, B'_t and L_t, and some unknown variables, i.e. assignment of instances to clusters. We use EM algorithm to solve this problem. As it was discussed, the assignment of instances to clusters, either labeled or unlabeled, are the unknown variables of the problem. We define the set of unknown variables as:

$$Z = \{Z_i | 1 \leq i \leq k\}, \tag{19}$$

where each Z_i is a binary vector with length Q indicating the assignment of the ith instance to each cluster. Thus, Z_i can be defined as:

$$Z_i = \left\{Z_{i,q} \in \{0, 1\} | 1 \leq q \leq Q\right\}, \tag{20}$$

such that:

$$Z_{i,q} = \begin{cases} 1 & \text{if } x_i \in q \\ 0 & \text{otherwise} \end{cases}. \tag{21}$$

In order to assign instances to clusters and find their means, it is now sufficient to specify the steps of the EM algorithm [30]. Considering $Z_{i,q}$as latent variables, B_t, B'_t and L_t as observed data, the set of means of clusters, $\{\mu_q | 1 \leq q \leq Q\}$, is the parameter to be estimated. Initialization, expectation and maximization steps, and the stopping criterion of the EM algorithm are discussed in the following. After the initializing step, expectation and maximization steps are performed iteratively until a stopping criterion is satisfied.

Initializations Instances are assigned to clusters randomly and for each cluster q, μ_q will be the mean of its members.

Expectation step In the expectation step, the probability of unknown variables given known variables and the means of clusters estimated in the previous iteration should be calculated. This probability can be stated as $P(Z|B_t, B_t', L_t, \ \mu^{\text{old}})$, where μ^{old} denotes the set of mean values of clusters in the previous iteration of the algorithm. We calculate this probability using a simplifying I.I.D assumption between individual assignment probabilities for different instances to clusters. Consequently:

$$P\left(Z|B_t, B_t', L_t, \ \mu^{\text{old}}\right) = \prod\nolimits_{i=1}^{l} P\left(Z_i|x_i^t, l_i^t, \mu^{\text{old}}\right) \times \prod\nolimits_{i=l+1}^{k} P\left(Z_i|x_i^t, \mu^{\text{old}}\right). \quad (22)$$

Now the task is to compute the two probabilities $P\left(Z_i|x_i^t, \mu^{\text{old}}\right)$ for unlabeled instances and $P\left(Z_i|x_i^t, l_i^t, \mu^{\text{old}}\right)$ for labeled ones. The only possible assignment of instances to clusters is that each instance x_i^t is assigned to exactly one cluster q. Therefore, it is sufficient to find the two sets of probabilities in Eq. (22) for $Z_i = I_q = (0, 0, \ldots, 1, 0, \ldots, 0)$, i.e. a binary vector with value 1 only in the qth element. Then, we have the following equation for unlabeled instances:

$$P\left(Z_i = I_q|x_i^t, \mu^{\text{old}}\right) = \frac{P\left(x_i^t, \mu^{\text{old}}|Z_i = I_q\right) *P\left(Z_i = I_q\right)}{\sum_{j=1}^{Q} P\left(x_j^t, \mu^{\text{old}}|Z_j = I_q\right) *P\left(Z_i = I_q\right)}. \quad (23)$$

Substituting the right-hand side probability values from Assumption 3, which states that members of a cluster form a Normal distribution, we obtain:

$$P\left(Z_i = I_q|x_i^t, \mu^{\text{old}}\right) = \frac{\pi_q \exp\left\{\frac{-\|x_i^t - \mu_q^{\text{old}}\|^2}{2\varepsilon}\right\}}{\sum_{j=1}^{Q} \pi_j \exp\left\{\frac{-\|x_i^t - \mu_j^{\text{old}}\|^2}{2\varepsilon}\right\}}, \quad (24)$$

where π_j shows the prior probability of assigning an instance to cluster j and $\|x_i^t - \mu_j^{\text{old}}\|^2$ denotes the Euclidean distance between instance x_i^t and center of cluster j, i.e. μ_j^{old}. Since we do not have any prior knowledge about clusters, π_j is assumed to be equal for all clusters and thus can be omitted from the equation. According to the assumption of $\varepsilon \to 0$, we obtain

$$P\left(Z_i = I_q|x_i^t, \mu^{\text{old}}\right) = \begin{cases} 1 \text{ if } q = \text{argmin}_{j=1..\text{Q}} \|x_{t,i} - \mu_q^{\text{old}}\|^2 \\ 0 \text{ otherwise} \end{cases}. \quad (25)$$

The relation states that an unlabeled instance belongs to a cluster with probability 1 provided that the cluster mean has the minimum distance to it, and this probability will be 0 for other clusters. In order to calculate $P\left(Z_i = I_q|x_i^t, l_i^t, \mu^{\text{old}}\right)$, we first state Lemma 1 regarding the assignment of instances to clusters and then calculate the probability according to this lemma.

Lemma 1 *Considering Assumptions 3 and 4 and assuming that each cluster center has a distance from $x_{t,i}$ which is different from that of other means of clusters, there exists only one target cluster, namely q, for each labeled instance to be assigned to. This means that $P\left(Z_i = I_q|x_i^t, l_i^t, \mu^{\text{old}}\right) = 1$ and $P\left(Z_i = I_{q'}|\text{x}_i^t, l_i^t, \mu^{\text{old}}\right) = 0$ for all $q' \neq q$.*

Proof First of all, we show that for each two different clusters $q1$ and $q2$, either

$$P\left(Z_i = I_{q2}|x_i^t, \mu_{q2}^{\text{old}}\right)/P(Z_i = I_{q1}|x_i^t, \mu_{q1}^{\text{old}}) \to 0 \tag{26}$$

or

$$P\left(Z_i = I_{q1}|x_i^t, \mu_{q1}^{\text{old}}\right)/P(Z_i = I_{q2}|x_i^t, \mu_{q2}^{\text{old}}) \to 0. \tag{27}$$

For a labeled instance, $P\left(Z_i = I_q|x_i^t, l_i^t, \mu^{\text{old}}\right)$ could be written as:

$$P\left(Z_i = I_q|x_i^t, l_i^t, \mu^{\text{old}}\right) = P\left(l_i^t|Z_i = I_q, x_i^t, \mu^{\text{old}}\right) \times P\left(Z_i = I_q|x_i^t, \mu^{\text{old}}\right). \tag{28}$$

It is obvious that the sum of (28) over different possible values of Z_i equals to 1, because instance x_i^t will be assigned to only one of the clusters. Having an instance x_i^t and two clusters $q1$ and $q2$, without loss of generality, assume that $q1$ is nearer to x_i^t than $q2$. This means that

$$P\left(Z_i = I_{q2}|x_i^t, \mu_{q2}^{\text{old}}\right)/P(Z_i = I_{q1}|x_i^t, \mu_{q1}^{\text{old}}) \to 0. \tag{29}$$

Now, we will prove the lemma statement. Based on the majority label of the instances in these two clusters, three situations are likely to happen:

1) Major label of $q1$ is l_i^t. In this case, according to equations (28) and (29) and Assumption 4, whatever the major label of $q2$ is, we have

$$P\left(Z_i = I_{q2}|x_i^t, l_i^t, \mu_{q2}^{\text{old}}\right)/P(Z_i = I_{q1}|x_i^t, l_i^t, \mu_{q1}^{\text{old}}) \to 0. \tag{30}$$

2) Major label of $q1$ is not l_i^t while major label of $q2$ is l_i^t. Thus,

$$P\left(l_i^t|x_i^t, \mu_{q1}^{\text{old}}\right)/P(l_i^t|x_i^t, \mu_{q2}^{\text{old}}) \to 0. \tag{31}$$

Therefore, we have

$$\frac{P\left(Z_i = I_{q1}|x_i^t, l_i^t, \mu^{\text{old}}\right)}{P\left(Z_i = I_{q2}|x_i^t, l_i^t, \mu^{\text{old}}\right)} = \frac{P\left(l_i^t|x_i^t, \mu_{q1}^{\text{old}}\right)}{P(l_i^t|x_i^t, \mu_{q2}^{\text{old}})} \times \frac{P(Z_i = I_{q1}|x_i^t, \mu_{q1}^{\text{old}})}{P\left(Z_i = I_{q2}|x_i^t, \mu_{q2}^{\text{old}}\right)}. \tag{32}$$

The computation of ratio given in Eq. (32) might not seem simple at the first glance as the first term in right-hand side converges to 0, while the second term goes to infinity. However, recalling Assumption 4, we have $\delta \ll \varepsilon$ which means that first term predominates the second one and consequently, the probability that $q1$ is the target cluster for the instance is much less than that of $q2$. In other words, we have

$$P\left(Z_i = I_{q2}|x_i^t, l_i^t, \mu_{q2}^{\text{old}}\right)/P(Z_i = I_{q1}|x_i^t, l_i^t, \mu_{q1}^{\text{old}}) \to 0. \tag{33}$$

3) None of $q1$ and $q2$ has l_i^t as their major label. In this case, similar to the first one, q_1 has more chance to be assigned to the instance, which means Eq. (30) holds again.

Finally, generalizing this argument for all clusters brings us to a total ordering among clusters regarding the assignment of each instance x_i^t to them. Therefore, the probability of assigning x_i^t to only one of the clusters, namely q, equals to 1. This probability for other clusters will be 0, which completes the proof of this lemma. □

From lemma 1, the assignment probability can be formulated by defining a set L, consisting of clusters with label $l_{t,i}$:

$$L = \{1 \leq q \leq Q | \mathrm{argmax}_{c'_j} \left(NL_{q,c'_j} \right) = l_{t,i}\} \tag{34}$$

Here, $NL_{q,c_j'}$ shows the number of instances that have label c'_j in cluster q. concretely, set L contains all clusters in which label $l_{t,i}$ is the majority. With the help of L, the above arguments can be summarized as:

$$P\left(Z_i = I_q | x_{t,i}, l_{t,i}, \mu^{\text{old}}\right) = \begin{cases} 1 & \text{if} \left(L = \emptyset \text{ and} q = \mathrm{argmin}_{j=1..Q} \mathrm{dist}\left(x_{t,i}, \mu_j^{\text{old}}\right)\right) \\ & \text{or} \left(L \neq \emptyset \text{ and} q = \mathrm{argmin}_{j \in L} \mathrm{dist}\left(x_{t,i}, \mu_j^{\text{old}}\right)\right) \\ 0 & \text{otherwise} \end{cases} \tag{35}$$

Equation (35) is interpreted as follows: with probability 1, an instance will be assigned to a cluster if that cluster has the minimum distance to the instance among all of the clusters that have the majority label equal to the label of that instance. If no cluster has the majority label equal to the instance's label, which shows the set L is empty, then the instance will be assigned to the nearest cluster with probability 1. The probability of assigning to all other clusters is 0.

Maximization step In this phase, the means of clusters are to be calculated for the next iteration of the algorithm. Regarding the maximization step of the EM algorithm, we have:

$$\mu^{\text{new}} = \mathrm{argmax}_{\mu} \phi\left(\mu, \mu^{\text{old}}\right), \tag{36}$$

where $\phi\left(\mu, \mu^{\text{old}}\right)$ is defined as:

$$\begin{aligned} \phi\left(\mu, \mu^{\text{old}}\right) &= E_{Z|\theta^{\text{old}}, B_t, B'_t, L_t}\left[\ln\left(P\left(B_t, B'_t, L_t, Z|\mu\right)\right)\right] \\ &= \sum_Z P\left(Z|B_t, B'_t, L_t, \mu^{\text{old}}\right) \ln\left(P\left(B_t, B'_t, L_t, Z|\mu\right)\right) \\ &= \sum_Z P\left(Z|B_t, B'_t, L_t, \mu^{\text{old}}\right) \ln\left[\left(P\left(B_t, B'_t, Z|\mu\right)\right) \times P\left(L_t|B_t, B'_t, Z, \mu\right)\right] \\ &= \sum_Z P\left(Z|B_t, B'_t, L_t, \mu^{\text{old}}\right) \ln\left(P\left(B_t, B'_t, Z|\mu\right)\right) \\ &\quad + \sum_Z P\left(Z|B_t, B'_t, L_t, \mu^{\text{old}}\right) \ln\left(P\left(L_t|B_t, B'_t, Z, \mu\right)\right) \end{aligned} \tag{37}$$

It can be assumed that $P\left(L_t|B_t, B'_t, Z, \mu\right)$ is independent of the means of clusters. Indeed, based on Assumption 4 and assuming independence among the labels of different instances, the information of assigning instances to clusters is sufficient for calculating the probability of their labels. Thus to maximize $\phi\left(\mu, \mu^{\text{old}}\right)$, the second term of Eq. (37) could be discarded. That is:

$$\mathrm{argmax}_{\mu} \phi\left(\mu, \mu^{\text{old}}\right) = \mathrm{argmax}_{\mu} \sum_Z P\left(Z|B_t, B'_t\right) \tag{38}$$

Assuming independence assumption for different instances, it can be shown that:

$$\mathrm{argmax}_{\mu} \phi\left(\mu, \mu^{\text{old}}\right) = \mathrm{argmax}_{\mu} \sum_{i=1}^{k} \sum_{q=1}^{Q} Z_{i,q} \ln \mathrm{P}\left(x_{t,i}, Z_{i,q}|\mu_q\right). \tag{39}$$

The probability of one instance belonging to a cluster q is a Normal distribution with the mean μ_q and covariance matrix εI_n. So, we have

$$\operatorname{argmax}_{\mu}\phi\left(\mu,\mu^{\text{old}}\right)=\operatorname{argmax}_{\mu}\frac{-1}{2}\left(\sum_{i=1}^{k}\sum_{q=1}^{Q}Z_{i,q}\left\|x_{t,i}-\mu_q\right\|^2\right). \tag{40}$$

Consequently, $\phi\left(\mu,\mu^{\text{old}}\right)$ takes its maximum value if:

$$\mu_q=\frac{1}{N_q}\sum_{i=1}^{k}Z_{i,q}x_{t,i}, \tag{41}$$

where N_q denotes the number of instances (either labeled or unlabeled) of cluster q, respectively. The relation simply states that the mean of a cluster is the arithmetic average of its members. The stopping criterion for algorithm is whether it converges to a solution or it has been run a specified number of iterations.

3.3.3 Incremental updating of cluster-based classifiers

In the previous subsection, the cluster formation and updating based on a new batch of data were discussed. Here, we focus on incremental updating of the classifiers as it is one of the crucial characteristics for classifiers of the pool. In the incremental version, the expectation and maximization steps are performed iteratively to update the clusters. The assignment of the instances is done similar to the initial construction of clusters which was discussed in Sect. 3.3.2 and according to the means of the existing clusters and previous and newly labeled instances of the clusters. After assigning phase is completed, new means for clusters are recalculated based on these new members. This process is repeated until the stopping criterion, which is the same as given before, is reached.

3.3.4 Instance classification

In order to classify a new instance, the nearest cluster to it has to be identified, firstly. In that cluster then, the majority of labels of members will be the predicted label for the instance. In addition, the posterior probabilities are calculated by finding the percentage of labels in the nearest cluster that are identical to the desired label.

3.4 Example of execution of the proposed method

In this subsection, we bring an example of execution of the algorithm, which will help understand different parts of the proposed method. In our e-mail classification example, assume that the batch size is 100. The system is working for a user, named Alice, and she receives 100 e-mails per week. The e-mails are represented with bag-of-word features such as tf-idf weights. The label of e-mails can be spam or ham. After receiving any e-mail, Alice may reveal its true label to the system. In the first week, there is no learnt concept and the system will classify all e-mails as ham. Alice receives three spam e-mails with the content: "congratulations! YOU HAVE WON £1,000,000.00. Kindly read the attachment". She tags these e-mails as spam. She also responds to 20 of her e-mails, which we interpret as a signal that they are ham. Out of these 20 e-mails, five of them are from different dealerships offering deals on cars. This is because Alice is buying a car and has given her e-mail to dealerships. Therefore, 23 e-mails are considered as labeled instances and the rest are unlabeled. At the end of the first week, a cluster-based classifier will be built based on 100 partially labeled e-mails. Assume that $Q = 5$, i.e., there are five clusters. The clusters are found based on partially labeled instances and each of the e-mails will fall into one cluster. Note that the cluster-based classifiers are designed to have clusters that are as pure as possible. Therefore, some of these clusters will have e-mails that are mostly spam and the others will have mostly ham e-mails. Finally, RDC is built using the average of the feature vectors of the first batch and label equal to the index of the first concept, which is one. In the second week, the system classifies e-mails. The distance of each e-mail to mean of each cluster is computed. The e-mail will be labeled as the majority label of the nearest cluster. Alice responds to 25 e-mails that will be considered as ham. She also buys her car at the end of the second week. The similarity of the second partially labeled batch to the only classifier of the pool is computed. Based on Eq. (17), the similarity is computed using RDC and also the cluster-based classifiers of the pool. In this iteration, there is only one cluster-based classifier in the pool. Therefore, it will be the optimal classifier for the second batch. It turns out that the similarity is higher than the predefined threshold. Therefore, the cluster-based classifier will be updated based on the second batch. RDC will also be updated similar to the previous batch. In the third week, Alice receives 100 e-mails and the system predicts their labels. She responds only to 15 of them. Among these e-mails, 10 of them are from dealerships. Since she is not interested anymore in buying cars, she tags four of them as spam. This will provide 19 labeled e-mails. At the end of the third week, the similarity of the third partially labeled batch to the only classifier of the pool is computed. This similarity is lower than the predefined threshold. Therefore, a new cluster-based classifier will be built and added to the classifier's pool. RDC will also be updated with the average of the feature vectors of the third batch and label equal to two. The weights of these classifiers will also be estimated based on their performance on the last batch, and the system is ready to process e-mails of the fourth week.

4 Experimental results

In order to evaluate the performance of the proposed algorithm, the computer experiments are conducted. These experiments on a number of standard datasets are conducted in order to evaluate the performance of the proposed method. In the following sections, datasets on which the experiments were performed are introduced, firstly. All datasets contain concept drifts or recurring concepts. Parameter tunings are discussed next. Finally, the performance of SPASC is compared to that of ReaSC [30]. The comparison of this method to our previous works is not practical, since they are all supervised algorithms. Being founded on a theoretically powerful idea, ReaSC is one of the most successful methods proposed in the field of semi-supervised classification of non-stationary data streams. ReaSC is similar to SPASC in several ways, including the use of ensemble classifiers and the cluster assumption for data; however, it has some differences that are explained in Sect. 2. According to experimental results, SPASC shows higher performance over ReaSC.

4.1 Datasets

Two real and one artificial datasets are used in experiments:

KDDCup99 This real dataset contains TCP connection records from two weeks of network traffic in MIT Lincoln laboratory. Each instance is labeled as either normal, or an attack, with exactly one of 22 identified attack types. Therefore, there are possibly 23 labels in the dataset. Each instance consists of 41 attributes. A 10 % subset of all data is extracted for our experiments, which has the total size of 490,000 instances. This dataset contains gradual or sudden concept drifts and has been used in several researches including [28].

Spam filtering This real dataset is the base that our e-mail classification example was derived from and is obtained from e-mail messages of Spam Assassin Collection [28]. Each e-mail is classified as either spam or ham. Only 20 % of instances are spam. This dataset contains 9,324 instances with 500 attributes for each. Attributes are words derived after applying feature selection with χ^2 measure. The characteristics of spam messages in this dataset gradually change as time passes, that is they show gradual concept drift [22].

Moving hyperplanes This synthetic dataset is used for modeling the problem of predicting the label of a rotating hyperplane. A decision boundary of a hyperplane in an n-dimensional space is denoted by $g(x) = \vec{w}.\vec{x} = 0$, where $\vec{w}$ is the direction of the hyperplane and $\vec{x}$ is an instance. Instances for which $g(x) > 0$ are labeled as positive and instances for which $g(x) < 0$ are labeled as negative. For simulating time-changing concepts, the orientation and position of the hyperplane could be changed in a smooth manner by changing the relative size of the weights [20]. This artificial dataset is generated in MOA environment [5] and consists of 8000 instances and 30 attributes. After each 2,000 instances, a drift occurs. Moreover, there are two concepts from which these instances are derived. They recur after the first 4,000 instances. Thus, both sudden concept drifts and recurring concepts are embedded in this dataset.

4.2 Parameter tuning

We used the following parameter setting in the experiments. The number of clusters, Q, is set to 5 for all datasets. In addition, the percent of labeled instances varies from 20 % of the whole data in all datasets. Window size is set to 50 for spam filtering, 500 for hyperplanes, and 1,000 for KDD99 datasets. The window size parameter was set by trial and error method and according to the properties of the datasets. This size should not be very small, as the batch may not contain sufficient instances to describe a concept. Very large values for this parameter will also lead to batches which may contain concept drifts. The number of concepts is set to 10 for all experiments. Thresholds, θ_1 and θ_2 are $(k+l)\log(0.65)$ and 0.95, respectively where k is the window size and l is the number of labeled instances. As stated in Sects. 3.2.1 and 3.2.2, these are threshold parameters used in *batch assignment* methods. Threshold $\theta_1$controls the similarity of a batch to a classifier in semi-Bayesian approach. The idea behind setting θ_1 to $(k+l)\log(0.65)$ is that Eq. (17), which calculates the similarity measure between a batch of instances and a concept for semi-Bayesian approach contains logarithm of $(k+l)$ probability values. Using this threshold, if none of these probabilities is more than 0.65, the similarity between the batch and the corresponding concept will not be high enough so that the batch can be assigned to that concept. Threshold θ_2 controls the accuracy of a classifier on a batch in semi-heuristic approach. Setting θ_2 to 0.95 means that if the accuracy of a classifier on a batch is more than 0.95, the similarity of the concept described by the classifier to the batch will be high enough to assign the batch to that classifier. Finally, β is set to 0.1 for all datasets using trial and error method. According to our experiments, value of β does not have much effect on the result of the algorithm.

To make a fair comparison, parameters of ReaSC are tuned either similar to those of SPASC or according to described configurations of [28]. Specifically, the number of clusters, the ratio of labeled instances to whole data and the number of basic concepts (models) are set equal to those of SPASC. ReaSC algorithm includes one extra parameter, called injection probability which is set to 20 %.

4.3 Accuracy of algorithms

In this section, the results of running both algorithms on all datasets will be compared. Figures 4, 5 and 6 demonstrate the performance of SPASC and ReaSC over each dataset. In each figure, diagrams of overall accuracy of SPASC with semi-Bayesian batch assigning method, SPASC with semi-heuristic batch assigning method, and ReaSC are shown. For all of the three datasets, SPASC performs more accurately than ReaSC, disregarding the method of batch assignment. Accuracy of semi-Bayesian, semi-heuristic and ReaSC algorithms on spam filtering dataset are 89.8, 0.9 and 81.7, respectively, which show an improvement of more than 8 % in accuracy for SPASC compared to ReaSC.

Over KDD99 dataset, these accuracies are 97.1, 94 and 93.8, respectively. Semi-Bayesian method shows the highest accuracy. For hyperplanes dataset, finally, accuracies are seen as 72.2, 71.9 and 71. The better performance of SPASC is mainly because it is capable of

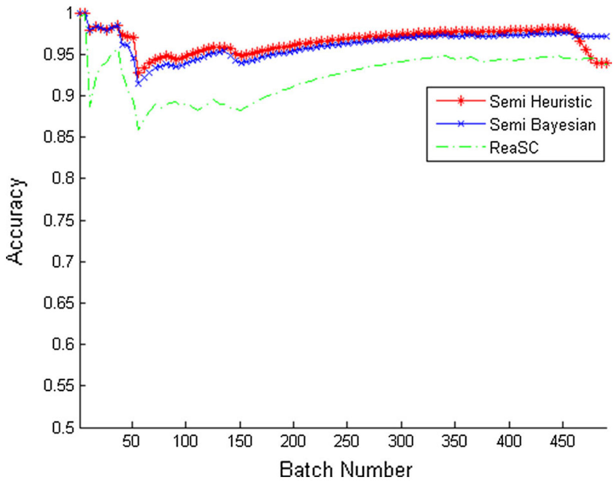

Fig. 4 Comparison of accuracy on KDD99 dataset

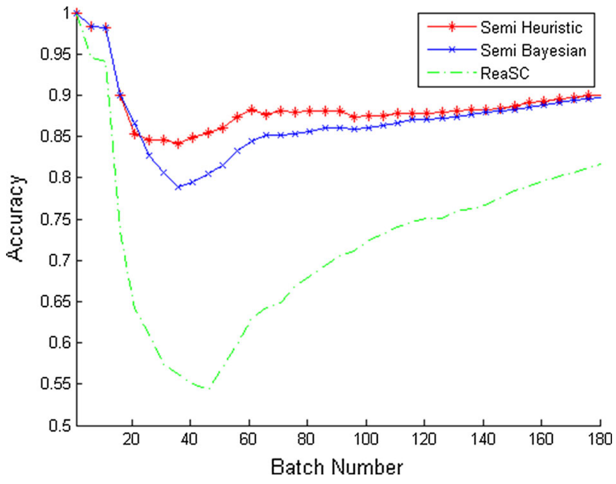

Fig. 5 Comparison of accuracy on spam filtering dataset

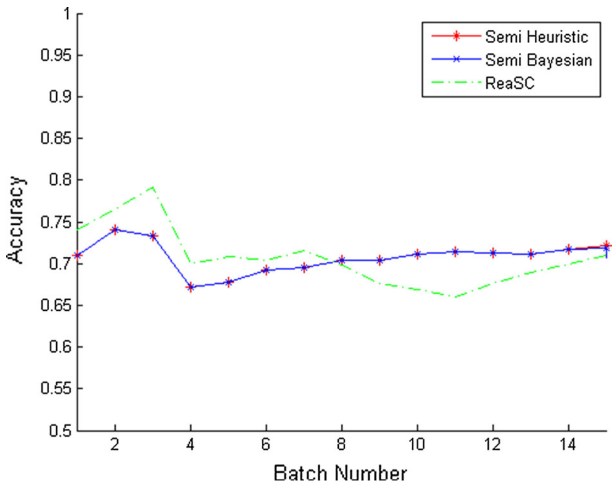

Fig. 6 Comparison of accuracy on hyperplanes dataset

detecting recurring concepts and thus it exploits previously learned concepts in the case that they reoccur. This significantly reduces the cost of learning a repeated concept from scratch. Therefore, SPASC will be more robust on datasets containing recurring concepts. ReaSC, however, cannot handle recurring concepts and will show lower accuracy in such cases. As described in Sect. 2, ReaSC will maintain an ensemble of L models. After getting a new chunk of partially labeled data, a new model is created. The best L models out of the $L + 1$ models will be selected based on their accuracies on the last seen chunk of data. The number of labeled instances is limited for each batch, so a good model may be discarded because of a few labeled examples. This means that the method is sensitive to noise. In addition, after a concept drift in L successive chunks, the whole models may be replaced. This implies that recurring concepts will not be handled using this method. Hence, if a concept reoccurs, ReaSC needs to learn models from scratch for that concept.

Finally, Comparing the semi-Bayesian and semi-heuristic method shows that they have relatively similar performance on two datasets, while semi-Bayesian approach performs slightly better (about 3 %) on KDD99 dataset. As discussed in Sect. 3.2.2, the main theoretical difference of these two methods is that in semi-Bayesian, the unlabeled data are also used to assign a batch of instances to a concept. Therefore, we expect that the semi-heuristic method has failed to assign the last batches to correct concepts. This may be handled in semi-Bayesian method by using unlabeled instances for batch assignment.

Moreover, it could be inferred from above figures that all algorithms decline in accuracy when facing concept drift; however, the amount of decrease varies among different algorithms.

4.4 Effect of percentage of labeled instances on performance

In this section, the accuracy of the algorithm with two batch assignment methods is evaluated for different proportions of labeled instances to the whole. For this purpose, we change percentage of labeled instances from 10 % of the whole data to 100 % with the ascending steps of 10 % and test the accuracy of methods. Figures 7, 8 and 9 show the results of these experiments. Ignoring a few exceptions, increasing the proportion of labeled instances

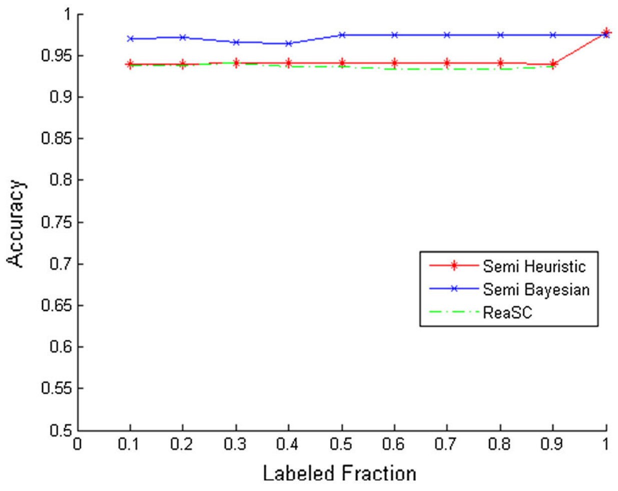

Fig. 7 Effect of percentage of labeled instances on KDD99 dataset

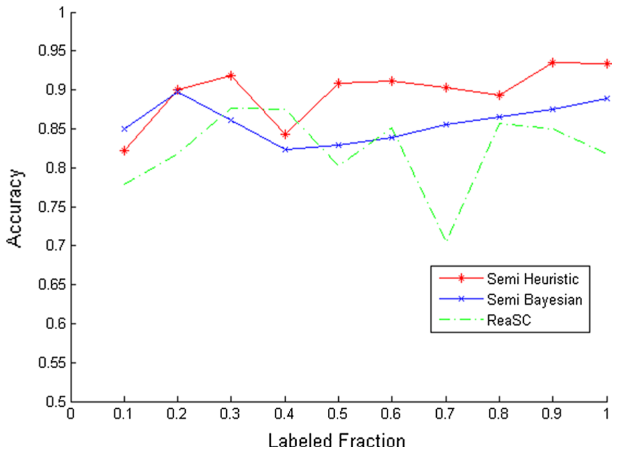

Fig. 8 Effect of percentage of labeled instances on spam filtering dataset

generally results in an increase in accuracy of SPASC. A notable point to mention, however, is that the amount of increase is very slight for percentage of labeled instances more than 20 %. This comes to the important conclusion that SPASC algorithm is able to use unlabeled instances efficiently to learn concepts. In addition, except for a few cases, SPASC outperforms ReaSC for all datasets and all percentages of labeled instances. Finally, it is expected that by increasing the percentage of labeled instances, the accuracy of learners increase as well. However, ReaSC does not show this behavior on spam filtering and hyperplanes datasets. The base models that ReaSC makes should intuitively improve by increasing the number of labeled instances. However, when the number of labeled instances increases, the accuracy of the mentioned $L + 1$ models will be measured on different samples. This may lead to different selection of models. The new selection may not be always better while testing on more labeled samples and this may add some randomness to the results while increasing the ratio of labeled instances.

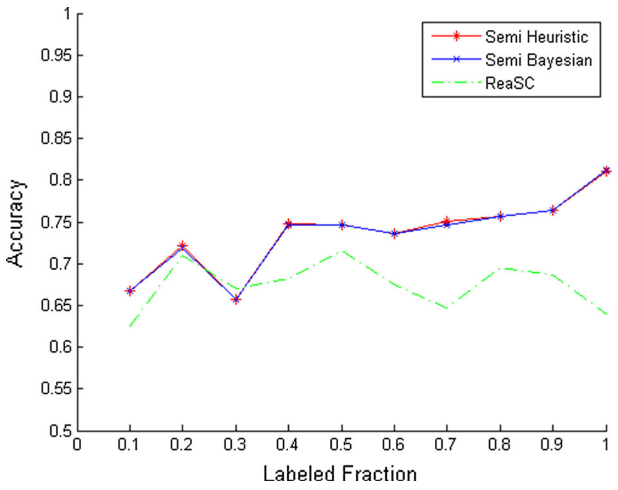

Fig. 9 Effect of percentage of labeled instances on hyperplanes dataset

4.5 Sensitivity to parameters

Besides the percentage of labeled instances, there are some other important parameters that should be analyzed. In this section, the effects of the number of clusters and the number of concepts on accuracy are analyzed. These two parameters are common between SPASC and ReaSC. As a result, the sensitivity to these parameters can be compared for the two algorithms. The effect of number of clusters on accuracy is shown in Figs. 10, 11 and 12. For all datasets, semi-Bayesian method outperforms others for sufficiently large number of clusters. For KDD99 and hyperplanes datasets, semi-heuristic also outperforms or at least functions as accurate as ReaSC. In spam filtering dataset, however, semi-heuristic fluctuates between 0.75 and 0.9 accuracy. In addition, it can be concluded that semi-Bayesian is more robust against changes in number of clusters than semi-heuristic method.

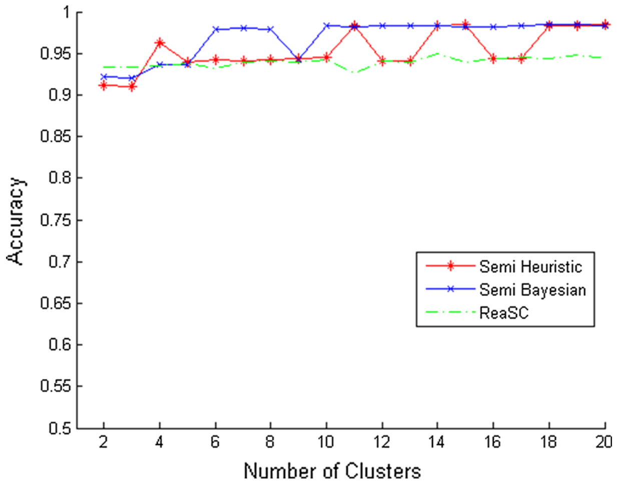

Fig. 10 Effect of number of clusters on KDD99 dataset

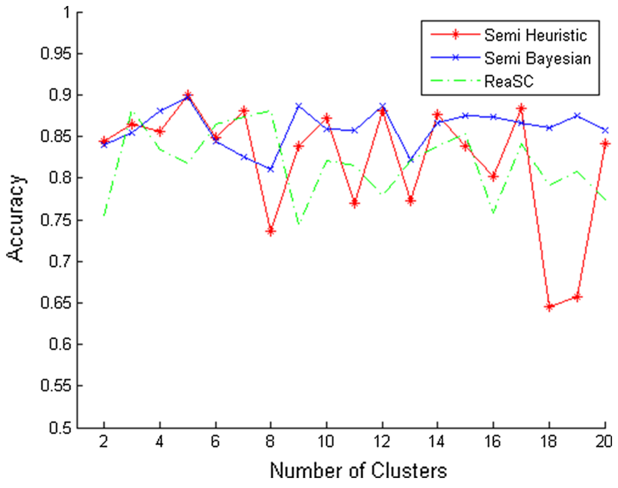

Fig. 11 Effect of number of clusters on spam filtering dataset

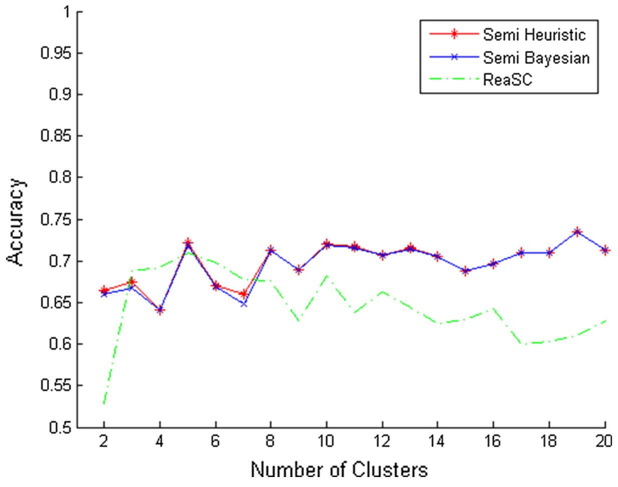

Fig. 12 Effect of number of clusters on hyperplanes dataset

Another interesting result than can be inferred from this experiment is that when the number of clusters increases, ReaSC shows a decline in accuracy while SPASC remains highly accurate. This is more noticeable for spam filtering and hyperplanes datasets. It is due to the recurring of concepts in these datasets. SPASC retains drifted instances in order to use them later if that concept recurs; conversely, ReaSC uses a mechanism for forgetting old data. Thus, as time passes, SPASC forms clusters with more instances, while these clusters are sparse in ReaSC, since it exploits fewer number of instances to form its clusters in comparison with SPASC that forms its clusters using all received instances from the beginning.

The effect of number of concepts (models) on the accuracy can be seen in Figs. 13, 14 and 15. Semi-Bayesian and semi-heuristic methods outperform ReaSC for sufficient number of concepts, i.e. 7 for KDD99, 3 for spam filtering and 5 for hyperplanes datasets. For smaller number of concepts, however, ReaSC gives more accurate results. This means that

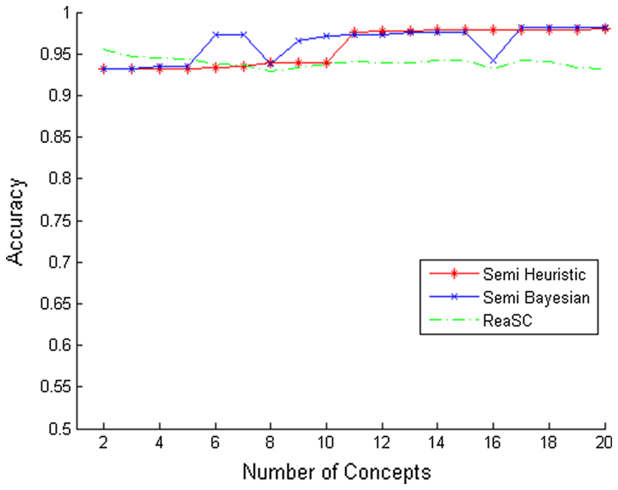

Fig. 13 Effect of number of concepts on KDD99 dataset

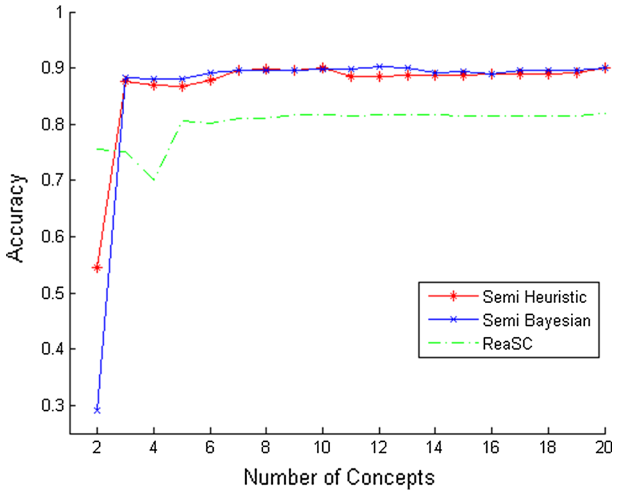

Fig. 14 Effect of number of concepts on spam filtering dataset

SPASC requires a minimum number of concepts to perform well, although this number may not be equal to the actual number of concepts describing instances. The problem is that SPASC method may assign more than one classifier to a single actual concept as its threshold parameter for the used batch assignment method may not be set properly or even no unique appropriate threshold parameter exists for all concepts. For example, hyperplanes dataset contains only two concepts, but SPASC needs at least five concepts. Similar problem exists for the supervised version of SPASC, i.e. PASC [19] algorithm. This problem has been addressed in our previous work, PMRCD [18], by managing the pool of classifiers via merge and split operations on the classifiers of the pool. Similar approach can be used for SPASC; however, it can already be seen that the current method has led to highly accurate results using sufficient number of classifiers.

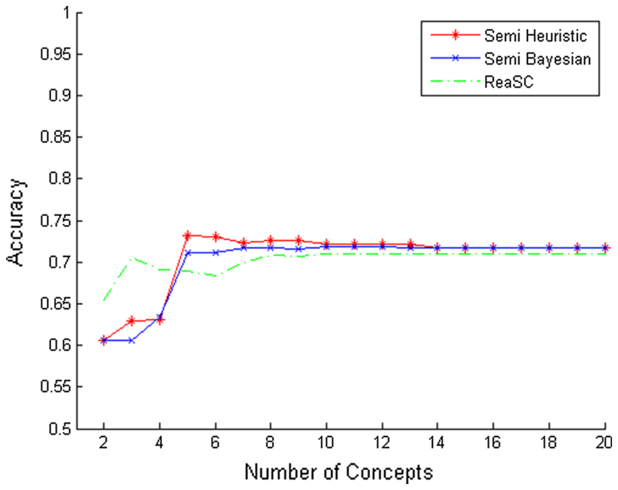

Fig. 15 Effect of number of concepts on hyperplanes dataset

5 Conclusion and future works

In this paper, we proposed a semi-supervised algorithm for classification of non-stationary data streams using ensemble learning methods. In this algorithm, called SPASC, a pool of classifiers is maintained and updated upon processing consecutive batches of data. Each classifier in the pool is associated with one single concept. When a new batch arrives, labels are predicted by the algorithm at first. Then, labels of some of them are revealed and used for updating the pool. Basic classifiers of the pool are of the cluster-based type. In the process of updating, using both labeled and unlabeled data, each classifier forms clusters of instances such that two conditions are fulfilled: (1) there would be the most intra-cluster similarity and inter-cluster dissimilarity and (2) each cluster is as pure in labels as possible. In literature, one of the most similar methods is ReaSC [28] since it uses ensemble classifiers which work based on clustering the data. This algorithm is one of the most successful methods proposed in the field of semi-supervised classification of non-stationary data streams; however it is incapable of handling recurring concepts. This is because it forgets learned base models based on recent batches of data. It is also discussed that it may be sensitive to noise in the stream while discarding the base models. In order to evaluate SPASC, it is mainly compared to ReaSC on three standard datasets. Considering experimental results, SPASC outperforms ReaSC in terms of accuracy. It is also proved via experiments that having a low percentage of labeled instances, achieving the performance near that of the situation where all labels are present are still possible. In some datasets, it was not the case for ReaSC, again because it does not take recurring concepts into account. The proposed method is open to improvement. Some future works in this direction include exploiting a number of pool management methods. These methods try to increase the accuracy of the pool by pursuing different strategies such as splitting one classifier into two new classifiers or merging a classifier with a similar one. Moreover, using variable sized batches of data could be considered. Length of a batch can be determined by monitoring some characteristics of input data during the running of the algorithm. Also, it is not necessary to apply classification and updating procedures on equal length batches. For instance, classification can be performed on smaller batches of data while updating can be done using larger batches. Furthermore, other measures for finding the similarity between a batch of data and a classifier, in batch assignment method, can be tested. In addition, the formulation has done based on some assumptions. These assumptions can be reconsidered to be more realistic. This may lead to more complicated methodology since optimization without those assumptions may be generally harder. Finally, conducting experiments on more and larger datasets is another direction to follow in future.

Acknowledgments The authors would like to thank the anonymous reviewers for their valuable comments and suggestions which improved the paper.

References

1. Aggarwal CC (2006) Data streams: models and algorithms. Springer-Verlag New York Inc, New York
2. Ahmadi Z, Beigy H (2012) Semi-supervised ensemble learning of data streams in the presence of concept drift. In: Proceedings of the 7th International Conference on Hybrid Artificial Intelligent Systems. Salamanca. Springer, Spain, pp 526–537
3. Bennett KP, Demiriz A, Maclin R (2002) Exploiting unlabeled data in ensemble methods. In: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp 289–296
4. Bifet A, Gavaldà R (2007) Learning from time-changing data with adaptive windowing. In: Proceedings of SIAM International Conference on Data Mining (SDM). Minneapolis, Minnesota, United States
5. Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) MOA: massive online analysis. J Mach Learn Res 99(9):1601–1604
6. Castillo G (2008) Adaptive learning algorithms for Bayesian network classifiers. AI Commun 21(1):87–88
7. Chapelle O, Schalkopf B, Zien A (2006) Semi-supervised learning. MIT press, Cambridge
8. Ditzler G, Polikar R (2011) Semi-supervised learning in nonstationary environments. In: Proceeding of the International Joint Conference on Neural Networks (IJCNN). IEEE, pp 2741–2748
9. 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, Massachusetts, United States, pp 71–80
10. Fan W (2004) Systematic data selection to mine concept-drifting data streams. In: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM. Seattle. WA, United States, pp 128–137
11. Fan W, Huang Y, Yu PS (2004) Decision tree evolution using limited number of labeled data items from drifting data streams. In: Proceedings of the 4th IEEE International Conference on Data Mining. IEEE Computer Society, pp 379–382
12. Gama J, Fernandes R, Rocha R (2006) Decision trees for mining data streams. Intell Data Anal 10(1):23–45
13. Gama J, Medas P, Rocha R (2004) Forest trees for on-line data. In: Proceedings of the ACM Symposium on Applied Computing. ACM, pp 632–636
14. Gao J, Fan W, Han J (2007) On appropriate assumptions to mine data streams: analysis and practice. In: Proceedings of the 7th IEEE International Conference on Data Mining. IEEE Computer Society
15. Gholipour A, Hosseini MJ, Beigy H (2013) An adaptive regression tree for non-stationary data streams. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM. Coimbra, Portugal, pp 815–817
16. Gomes JB, Menasalvas E, Sousa P (2010) Tracking recurrent concepts using context. In: Proceedings of the 7th International Conference on Rough Sets and Current Trends in Computing. Springer-Verlag, Warsaw, Poland, pp 168–177
17. 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 IEEE International Conference on Data Mining Workshops. IEEE. Vancouver, Canada, pp 588–595
18. Hosseini MJ, Ahmadi Z, Beigy H (2012) New management operations on classifiers pool to track recurring concepts. In: Proceedings of the 14th international conference on data warehousing and knowledge discovery. Springer, Vienna, Austria, pp 327–339
19. Hosseini MJ, Ahmadi Z, Beigy H (2013) Using a classifier pool in accuracy based tracking of recurring concepts in data stream classification. Evol Syst 4(1):1–18

20. 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, California, United States, pp 97–106
21. Karimi Z, Abolhassani H, Beigy H (2012) A new method of mining data streams using harmony search. J Intell Inf Syst 39(2):491–511
22. 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
23. Klinkenberg R (2004) Learning drifting concepts: example selection vs. example weighting. Intell Data Anal 8(3):281–300
24. 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, Germany, pp 449–456
25. Li P, Wu X, Hu X (2010) Mining recurring concept drifts with limited labeled streaming data. In: Proceeding of the 2nd Asian Conference on Machine Learning (JMLR), Tokyo, Japan, pp 241–252
26. MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, California, United States, pp 281–297
27. Masud MM, Gao J, Khan L, Han J, Thuraisingham B (2008) A practical approach to classify evolving data streams: training with limited amount of labeled data. In: Proceedings of the 8th IEEE International Conference on Data Mining, pp 929–934
28. Masud MM, Woolam C, Gao J, Khan L, Han J, Hamlen KW, Oza NC (2012) Facing the reality of data stream classification: coping with scarcity of labeled data. Knowl Inf Syst 33(1):213–244
29. Minku LL (2011) Online ensemble learning in the presence of concept drift. University of Birmingham, Birmingham
30. Moon TK (1996) The expectation-maximization algorithm. Signal Process Mag IEEE 13(6):47–60
31. Nishida K (2008) Learning and detecting concept drift. Information science and technology. Hokkaido University, Hokkaido
32. 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. IEEE Computer Society, pp 38–42
33. Scholz M, Klinlenberg 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
34. 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, California, United States, pp 377–382
35. Wang H, Fan W, Yu PS, Han J (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, DC, United States, pp 531–540
36. Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. Mach Learn 23(1):69–101
37. 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-Verlag, Prague, Czech Republic, pp 552–562
38. Wu X, Li P, Hu X (2012) Learning from concept drifting data streams with unlabeled data. Neurocomputing. Elsevier, Amsterdam
39. Zhou ZH (2011) When semi-supervised learning meets ensemble learning. Front Electr Electron Eng China 6(1):6–16
40. Zhu X, Goldberg AB (2009) Introduction to semi-supervised learning. Synth Lect Artif Intell Mach learn 3(1):1–130
41. Zliobaite I (2009) Learning under concept drift: an overview. Vilnius University, Technical Report

Mohammad Javad Hosseini is currently a Ph.D. student in Computing Science and Engineering Department at University of Washington, Seattle, US. He received his B.Sc. degree in 2010 and M.Sc. degree in 2012 from Sharif University of Technology, Tehran, Iran. His research interests focus on large-scale machine learning and data mining.

Ameneh Gholipour is a M.Sc. student in department of Computing Science at University of Alberta, Edmonton, Canada. She is also member of Alberta Innovates Center for Machine Learning (AICML). She received her B.Sc. degree in 2011 and her M.Sc. degree in 2013 both from Sharif University of Technology, Tehran, Iran. Her research interests mainly include machine learning and data mining, specifically text mining.

Hamid Beigy received the B.Sc. and M.Sc. degrees in Computer Engineering from the Shiraz University in Iran, in 1992 and 1995, respectively. He also received the Ph.D. degree in Computer Engineering from the Amirkabir University of Technology, Iran, in 2004. Currently, he is an Associate Professor in Department of Computer Engineering at the Sharif University of Technology, Tehran, Iran. His research interests include machine learning and high performance computing.