Knowl Inf Syst
DOI 10.1007/s10115-011-0447-8

REGULAR PAPER

Facing the reality of data stream classification: coping with scarcity of labeled data

Mohammad M. Masud · Clay Woolam · Jing Gao · Latifur Khan · Jiawei Han · Kevin W. Hamlen · Nikunj C. Oza

Received: 6 May 2009 / Revised: 26 April 2011 / Accepted: 22 October 2011

Abstract Recent approaches for classifying data streams are mostly based on supervised learning algorithms, which can only be trained with labeled data. Manual labeling of data is both costly and time consuming. Therefore, in a real streaming environment where large volumes of data appear at a high speed, only a small fraction of the data can be labeled. Thus, only a limited number of instances will be available for training and updating the classification models, leading to poorly trained classifiers. We apply a novel technique to overcome this problem by utilizing both unlabeled and labeled instances to train and update the classification model. Each classification model is built as a collection of micro-clusters using semi-supervised clustering, and an ensemble of these models is used to classify unlabeled data. Empirical evaluation of both synthetic and real data reveals that our approach outperforms state-of-the-art stream classification algorithms that use ten times more labeled data than our approach.

Keywords Data stream classification · Semi-supervised clustering · Ensemble classification · Concept drift

1 Introduction

Data stream classification is a challenging problem because of two important properties of the stream: its infinite length and evolving nature. There are mainly two categories of evolution,

M. M. Masud (✉) · C. Woolam · L. Khan · K. W. Hamlen
Department of Computer Science, University of Texas at Dallas,
Richardson, TX 75080, USA
e-mail: mehedy@utdallas.edu

J. Gao · J. Han
Department of Computer Science, University of Illinois at Urbana Champaign,
Urbana, IL 61801, USA

N. C. Oza
Intelligent Systems Division, NASA Ames Research Center,
Moffett Field, CA 94035, USA

Published online: 20 November 2011

namely, concept drift and concept evolution. Informally, concept drift occurs when the class labels of a set of examples change over time [26]. Concept evolution occurs when a new class emerges in the stream [32]. However, both concept drift and concept evolution may also occur simultaneously. In any case, the challenge is to build a classification model that is consistent with the current concept. Most of the existing data stream classification techniques [2,10,15,17,21,25,34,38,41] are based on supervised learning algorithms. Therefore, the amount of labeled data in the stream affects the quality of the learned model. Manual labeling of data is often costly and time consuming, so in an streaming environment, where data appear at a high speed, it is not always possible to manually label all the data as soon as they arrive. Thus, in practice, only a small fraction of the stream can be labeled by human experts. Traditional stream classification algorithms have very few instances with which to update their models in such circumstances, leading to poor classifiers. Our proposed approach utilizes both labeled and unlabeled data for training and addresses this limited labeled data problem of the supervised learning techniques.

For example, suppose an organization receives flight reports as text documents from all over the world, at a rate of 1,000 reports per day. The documents have categories: "normal", "minor mechanical problem", "minor weather problem", "major mechanical problem", and so on. The system generates time sensitive warning messages that are sent to aviation authorities for proper action based on the document category. Important decision-making actions such as flight planning, resource allocation, and personnel assignment are affected by these warnings. Timely delivery of these warnings is critical to avoid both financial loss and customer dissatisfaction. The trained domain experts can only read and label 100 of the 1,000 reports each day. We would like to create an automated stream document classification system so that all 1,000 documents can be classified each day. One option is to train a purely supervised learner using our 100 labeled data points. Another option is to wait to train the supervised learner after 1,000 points are labeled. Neither of these scenarios are ideal. In the former, our learner is trained with too few points. In the latter, we have to wait 10 days for each labeled batch and may have an outdated model by that time.

To address these difficulties, we propose an algorithm that updates the existing classification model utilizing the available 100 labeled and 900 unlabeled instances, while achieving the same or better classification accuracy than a classification model that is updated using 1,000 labeled instances. Our approach uses semi-supervised learning to realize practical, realistic, and cost-effective data stream classification under these conditions. Henceforth, we refer to data streams as *partially labeled* when only a fraction of their instances are labeled, and *completely labeled* when all instances are labeled.

Semi-supervised learning of data streams is challenging for several reasons. First, the data stream is virtually infinite, and therefore, the learning must be incremental. We address this problem by following a hybrid batch-incremental technique, dividing the stream data into equal-sized chunks so that each chunk can be stored in main memory. When $P\%$ of instances in a data chunk have been labeled, we train one classifier from that chunk. We maintain an ensemble of L such classifiers to classify the unlabeled data using majority voting. Keeping the number of classifiers in the ensemble constant allows us to elegantly address the infinite length problem. Second, in order to cope with concept drift, the semi-supervised learner must adapt to changes in the stream. This is accomplished by updating the ensemble by choosing the best L models from the $L+1$ models (the previous L models plus the new model), based on their individual accuracies over the labeled training data of the new data chunk. Finally, because of concept evolution, new classes arrive in the stream, which requires updating the existing model with the new class label information. We address this problem by refining the existing models in the ensemble whenever a new class of data evolves in the stream.

One of the main technical merits of the proposed framework lies in building a classifier using semi-supervised learning. In brief, we apply constraint-based, semi-supervised clustering to create K clusters [31] and split them into homogeneous clusters (*pure* clusters) that contain only unlabeled instances, or only labeled instances from a single class. We maintain a summary of each pure cluster (e.g., the centroid, number of data points, etc.) as a *micro-cluster* and discard all the raw data points in order to save memory and achieve faster running time. Finally, we apply a label propagation technique [39] on the micro-clusters to label the unlabeled micro-clusters. These micro-clusters act as a classification model. The power of the semi-supervised learning lies in utilizing both labeled and unlabeled data for building a model. In a typical supervised data stream learning scenario, most of the unlabeled data are discarded (not used for training) and are therefore wasted. However, semi-supervised learning can utilize most of the unlabeled data to improve the quality of the learned model. Therefore, the proposed framework is more applicable to real world data stream classification problems than are existing supervised frameworks.

The major contributions of the work can be summarized as follows. First, we propose an efficient semi-supervised clustering algorithm based on cluster-impurity measure. Second, we apply our technique to classify evolving data streams. To our knowledge, there are no stream data classification algorithms that apply semi-supervised clustering. Third, we provide a solution to the more practical problem of stream classification when labeled data are scarce. We show that our approach, using only 10% labeled training data, achieves better classification accuracy in real world data sets than other stream classification approaches that use 100% labeled training data. We believe that the proposed method provides a promising, powerful, and practical technique to the stream classification problem in general.

The rest of the paper is organized as follows: Sect. 2 discusses related work, Sect. 3 provides an overview of the classification process, Sect. 4 describes the theoretical background and the implementation of the semi-supervised clustering, Sect. 5 discusses the ensemble classification and ensemble updating process with micro-clusters, Sect. 6 discusses data set and experimental setup, and evaluation of our approach, Sect. 7 discusses our findings, and Sect. 8 concludes with suggestions for future work.

2 Related work

Our work is related to both semi-supervised clustering and stream classification techniques. We briefly discuss both approaches.

Semi-supervised clustering techniques utilize a small amount of knowledge available in the form of pairwise constraints (*must-link, cannot-link*), or class labels of the data points. Semi-supervised clustering techniques can be subdivided into two categories [6]: constraint based and distance based. Constraint-based approaches (e.g., [4,12,37]) try to cluster the data points without violating the given constraints. Distance-based techniques (e.g., [11,19,24,40]) use a specific distance metric or similarity measure (e.g., Euclidean distance), but the distance metric is parameterized so that it can be adjusted to satisfy the given constraints. Some recent approaches for semi-supervised clustering integrate the search-based and constraint-based techniques into a unified framework by applying pairwise constraints atop the unsupervised K-means clustering technique, and formulating a constrained K-means clustering problem [5,6,9]. These approaches usually apply the expectation-maximization (E-M) technique to solve the constrained clustering problem.

Our approach follows the constraint-based approach but differs from past constraint-based approaches. Most constraint-based approaches use pair-wise constraints (e.g., [9], whereas

we utilize a cluster-impurity measure based on the limited labeled data contained in each cluster [31]. If pair-wise constraints are used then the running time per E–M step is quadratic in total number of labeled points, whereas the running time is linear if impurity measures are used. Impurity measures are therefore more practical for classifying high-speed stream data. Basu et al. [4] use neither pair-wise constraints nor cluster-impurity measures. Demiriz et al. [12] use a cluster-impurity measure but apply expensive genetic algorithms and must adjust weights given to different components of the clustering objective function in order to obtain good clusters. In contrast, we apply E-M and do not need to tune parameters to get a better objective function. Furthermore, we use a compound impurity measure rather than the simple impurity measures used in [12]. To our knowledge, no other work applies a semi-supervised clustering technique to classify stream data.

Data stream mining has gained popularity in the past decade due to the overwhelming rate of data generated everyday in different applications. Two major thrusts in data stream mining are clustering and classification. There are many approaches in data stream clustering, which mainly focus on creating and updating clusters in an online fashion [42]. The micro-clustering technique for online clustering has been widely used [3,27]. However, these clustering techniques are unsupervised, but our approach applies a semi-supervised clustering technique as discussed above.

There have been many works in stream data classification. Data stream classification techniques can be categorized as single-model versus ensemble,and as supervised techniques requiring completely labeled training data versus those that require only partially labeled training data. Some single-model classification techniques incrementally update their model when new data arrives [14,21]. However, these techniques require complex operations to update the internal structure of the model and in most cases use only recent data to update the model. Thus, contributions of historical data are forgotten at a constant rate even if some of the historical data are consistent with the current concept. This can adversely affect the prediction accuracy of the refined model. Several ensemble techniques for stream data mining have been proposed [15,17,18,22,25,38]. These ensemble approaches have the advantage that they can be more efficiently built than updating a single model and they observe higher accuracy than their single-model counterparts [35]. Our approach is an ensemble approach.

Supervised techniques for data stream classification require that the training data be completely labeled. Many approaches (e.g., [21,25,38,45]) assume that the true labels of data points become available immediately after classification. In other words, they require that all instances in the stream be labeled eventually. However, some other approaches (e.g., [1,2]) assume two logical streams, where one stream contains only labeled instances and the other contains only unlabeled instances. The labeled stream is used to update the existing model, and the model is used to classify the unlabeled stream. Therefore, although they are supervised approaches, they do not require that all the instances in the stream be labeled. Some other supervised approaches (e.g., [28]) consider the labeling delay by analyzing IBk classifier performance under different delay values and classifier update strategies.

Since data labeling is costly and time consuming, some techniques have been proposed that are not based on supervised learning, and that try to minimize the labeling cost by utilizing small amount of labeled data. There are different approaches that fall into this category, such as active learning [16,46], positive unlabeled (PU) learning [30,44], and semi-supervised learning [29,31]. Active learning techniques selectively choose a small number of unlabeled data for labeling and use those labels to update the existing classification model. Active learning techniques are useful since they require few labeled instances for training. However, choosing a proper threshold for the labeling decision (i.e., whether an instance should be labeled) is a major challenge for these techniques. Furthermore, these techniques cannot

use unlabeled data for training, whereas semi-supervised learning approaches can. PU learning approaches use only labeled, positive instances along with some unlabeled instances to update the classifier. These approaches are only applicable to binary classification problems. Semi-supervised learning techniques capitalize on a few labeled instances and use a large amount of unlabeled instances to train a classification model.

Our proposed approach is a semi-supervised approach. It is applicable to multi-class classification problems. The main difference between these techniques and the active learning approaches is that the semi-supervised approaches do not require selective labeling, and therefore, do not need a threshold for deciding whether an instance should be labeled. In the technique proposed here, the data points that need to be labeled may be chosen randomly. Furthermore, semi-supervised approaches can also utilize unlabeled training data for improving the quality of the learned model.

Our current work is an extension of two previous works [31,39]. In this paper, we describe our technique more elaborately and provide detailed understanding and theoretical analysis of the proposed framework. We have also enriched the experimental results by adding three more datasets beyond those reported in [31]. Furthermore, we run more rigorous experiments, such as parameter sensitivity, running time analysis, as well as contribution of different components of the proposed technique, and report in-depth analysis and justification of the results.

3 Top level description

We begin by informally defining the data stream classification problem. We assume that data arrive in chunks, as follows:

$$\begin{aligned} D_1 &= x_1, \ldots, x_S \\ D_2 &= x_{S+1}, \ldots, x_{2S} \\ &\vdots \\ D_n &= x_{(n-1)S+1}, \ldots, x_{nS} \end{aligned}$$

where x_i is the ith instance in the stream, S is the chunk size, D_i is the ith data chunk, and D_n is the latest data chunk. Assuming that the class labels of all the instances in D_n are unknown, the problem is to predict their class labels. Let y_i and $\hat{y}_i$ be the actual and predicted class labels of x_i, respectively. If $\hat{y}_i = y_i$, then the prediction is correct; otherwise it is incorrect. The goal is to minimize the prediction error.

Table 1 explains the terms and symbols that are used throughout the paper. The high-level architecture of our Realistic Stream Classifier (ReaSC) is depicted in Fig. 1.

We train a classification model from a data chunk D_i as soon as $P\%$ ($P \ll 100$) randomly chosen instances from the chunk have been labeled by an independent labeling mechanism (e.g., human experts). This assumption is less strict than other stream classification techniques, such as [38], which assumes that all the instances of D_i are labeled before it can be used to train a model. We build the initial ensemble M from L models = $\{M^1, ..., M^L\}$ using the first L data chunks, where M^i is trained from chunk D_i. Algorithm 1 (below) is then applied to each new incoming chunk.

The main steps are as follows:

1. ***Classification***: The existing ensemble is used to predict the labels of each instance in D_n using a classification function and majority voting (Sect. 5.1). As soon as D_n has been partially labeled, the following steps are performed.

Table 1 Symbols and terms

S:	Chunk size	L:	Ensemble size
C:	Number of classes in the stream	D_i:	A data chunk
K:	Number of initial clusters per chunk (before splitting)		
$\hat{K}$:	Number of micro-clusters per model ($\hat{K} \leq (C+1)K$)		
P:	Percentage of labeled data in each chunk		
M:	The ensemble of models $\{M^1,...,M^L\}$		
M^i:	The ith model in the ensemble = the set of micro-clusters $\{\psi_1^i,...,\psi_{\hat{K}}^i\}$		
ψ_j^i:	The jth micro-cluster in the ith model		

Labeled instance: An instance that has been correctly labeled by an independent labeling mechanism (e.g., a domain expert)
Partially labeled chunk: A data chunk having at least $P\%$ labeled instances
Completely labeled chunk: A data chunk having 100% labeled instances

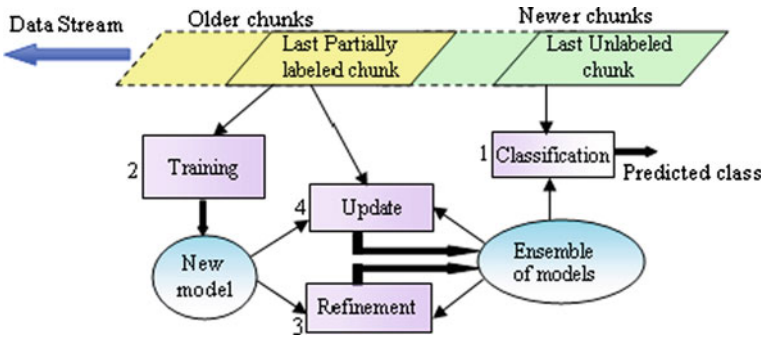


Fig. 1 Overview of ReaSC

Algorithm 1 ReaSC

Input: D_n: Latest data chunk
M: current ensemble of L models $\{M^1, ..., M^L\}$
Output: Updated ensemble M
1: **for all** $x_i \in D_n$ **do** $\hat{y}_i \leftarrow$ **Classify**(M,x_i) (Sect. 5.1)
2: When $P\%$ instances in D_n are labeled
3: $M' \leftarrow$ **Train**(D_n) (Sect. 4) /* */ Build a new model M'
4: $M \leftarrow$ **Refine-Ensemble**(M, M') (Sect. 5.2)
5: $M \leftarrow$ **Update-Ensemble**(M, M', D_n) (Sect. 5.3)
6: **return** M

2. ***Training***: Training occurs by applying semi-supervised clustering on the partially labeled training data to build K clusters (Sect. 4). The semi-supervised clustering is based on the E-M algorithm, which locally minimizes an objective function. The clusters are then split into *pure* clusters, where a pure cluster contains only unlabeled datapoints or labeled datapoints of only one class. The summary of each pure cluster is then saved as a *micro-cluster*. We combine this set of micro-clusters with the *labeled* micro-clusters (micro-clusters that correspond to manually labeled instances) from the last r contiguous chunks. Finally, we apply a label propagation technique to propagate labels from the labeled micro-clusters to the unlabeled micro-clusters. This yields a new classification model M' that can be used to classify unlabeled data.

Table 2 An example of ReaSC actions with stream progression

Arrival of chunk	Action(s)
Stream progression	
D_1	—
D_2	$M^1 \leftarrow \text{Train}(D_1)$
⋮	⋮
D_{L+1}	$M^L \leftarrow \text{Train}(D_L)$, Initial model $M = \{M^1, ..., M^L\}$ $\forall x_j \in D_{L+1}\ \hat{y}_j \leftarrow \text{Classifiy}(M, x_j)$
D_{L+2}	$M' \leftarrow \text{Train}(D_{L+1})$ $M \leftarrow \text{Refine-Ensemble}(M, M')$ $M \leftarrow \text{Update-Ensemble}(M, M', D_{L+1})$ $\forall x_j \in D_{L+2}\ \hat{y}_j \leftarrow \text{Classifiy}(M, x_j)$
⋮	⋮
D_{L+i}	$M' \leftarrow \text{Train}(D_{L+i-1})$ $M \leftarrow \text{Refine-Ensemble}(M, M')$ $M \leftarrow \text{Update-Ensemble}(M, M', D_{L+i-1})$ $\forall x_j \in D_{L+i}\ \hat{y}_j \leftarrow \text{Classifiy}(M, x_j)$
⋮	⋮

3. ***Ensemble refinement***: In this step, M' is used to refine the existing ensemble of models (Sect. 5.2), if required. Refinement is required if M' contains some data of a particular class c, but no model in the ensemble M contains any data of that class. This situation may occur because of concept evolution. In this case, the existing ensemble M does not have any knowledge of class c, so it must be refined to classify instances of this class. Refinement occurs by injecting micro-clusters of M', which contains labeled instances of class c, into the existing models of the ensemble.
4. ***Ensemble update***: In this step, we select the best L models from the $L+1$ models of $M \cup \{M'\}$ based on their accuracies on the labeled instances of D_n (Sect. 5.3). These L best models comprise the new ensemble. The ensemble technique helps the system to cope with concept drift.

Table 2 illustrates a schematic example of ReaSC. In this example, we assume that $P\%$ of data chunk D_i is labeled by the time chunk D_{i+1} arrives. The initial ensemble is built with the first L chunks. Then, the ensemble is used to classify the latest chunk (D_{L+1}). Arrival of each following chunk D_{L+i} ($i > 1$) evokes the following sequence of operations:

(i) Train a new model M' using chunk D_{L+i-1}, which been partially labeled by the arrival of chunk D_{L+i}.
(ii) Refine the existing ensemble M using the new model M'.
(iii) Update the ensemble M by choosing the best L models from $M \cup \{M'\}$.
(iv) Classify each instance in D_{L+i} using ensemble M.

4 Training with limited labeled data

As mentioned earlier, we train a classification model from each partially labeled data chunk. The classification model is a collection of K micro-clusters obtained using semi-supervised

clustering and label propagation. Training consists of four basic steps: semi-supervised clustering, cluster splitting, label propagation, and storing the cluster summaries as micro-clusters.

4.1 Semi-supervised clustering

In the semi-supervised clustering setting, we are given a set of m data points $\mathcal{X} = \{\mathbf{x_1}, \ldots, \mathbf{x_l}, \mathbf{x_{l+1}}, \ldots, \mathbf{x_m}\}$, where the first l instances are labeled and the remaining instances are unlabeled. We assign the class label $y_i = 0$ for each unlabeled instance $\mathbf{x_i}$, $i > l$. We are to create K clusters, maintaining the constraint that all points in the same cluster have the same class label. We restrict the value of parameter K to be greater than the number of classes C, since intuitively there should be at least one cluster for each class of data. We will first re-examine the unsupervised K-means clustering in Sect. 4.1.1 and then propose a new semi-supervised clustering technique using cluster-impurity minimization in Sect. 4.1.2.

4.1.1 Unsupervised K-means clustering

The unsupervised K-means clustering creates K-partitions of the data points based on the only available information—the similarity/dispersion measure among the data points. The objective is to minimize the sum of dispersion between each data point and its corresponding cluster centroid (i.e., intra-cluster dispersion). Given m unlabeled data points $\mathcal{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$, K-means creates K-partitions $\{\mathcal{X}_1, ..., \mathcal{X}_K\}$ of $\mathcal{X}$, minimizing objective function

$$\mathcal{O}_{Kmeans} = \sum_{i=1}^{K} \sum_{\boldsymbol{x} \in \mathcal{X}_i} ||\boldsymbol{x} - \boldsymbol{u_i}||^2 \tag{1}$$

where $\boldsymbol{u_i}$ is the centroid of cluster i, and $||\boldsymbol{x} - \boldsymbol{u_i}||$ is the Euclidean distance between $\boldsymbol{x}$ and $\boldsymbol{u_i}$.

4.1.2 K-means clustering with cluster-impurity minimization

Given a limited amount of labeled data, the goal of K-means with minimization of cluster impurity (MCI-Kmeans) is to minimize the intra-cluster dispersion (same as unsupervised K-means) while minimizing the impurity of each cluster. A cluster is completely pure if it contains only unlabeled instances or labeled instances from only one class. Thus, the objective function should penalize each cluster for impurity. The general form of the objective function is as follows:

$$\mathcal{O}_{MCIKmeans} = \sum_{i=1}^{K} \sum_{\boldsymbol{x} \in \mathcal{X}_i} ||\boldsymbol{x} - \boldsymbol{u_i}||^2 + \sum_{i=1}^{K} \mathcal{W}_i Imp_i \tag{2}$$

where $\mathcal{W}_i$ is the weight associated with cluster i and Imp_i is the impurity of cluster i. In order to ensure that both the intra-cluster dispersion and cluster impurity are given the same importance, the weight associated with each cluster should be adjusted properly. In addition, we wish to penalize each data point that contributes to the impurity of the cluster. So, the weight associated with each cluster is chosen to be

$$\mathcal{W}_i = |\mathcal{X}_i| \bar{D}_{\mathcal{X}_i} \tag{3}$$

where $\mathcal{X}_i$ is the set of data points in cluster i and $\bar{D}_{\mathcal{X}_i}$ is the average dispersion of each of these points from the cluster centroid. Thus, each instance contributes to the total penalty,

which equals the cluster impurity multiplied by the average dispersion of the data points from the centroid. We observe that Eq. 3 is equivalent to the sum of dispersions of all the instances from the cluster centroid. That is, we may rewrite Eq. 3 as:

$$\mathcal{W}_i = \sum_{\boldsymbol{x} \in \mathcal{X}_i} ||\boldsymbol{x} - \boldsymbol{u_i}||^2$$

Substituting this formula for $\mathcal{W}_i$ into Eq. 2 yields

$$\begin{aligned} \mathcal{O}_{MCIKmeans} &= \sum_{i=1}^{K} \sum_{\boldsymbol{x} \in \mathcal{X}_i} ||\boldsymbol{x} - \boldsymbol{u_i}||^2 + \sum_{i=1}^{K} \sum_{\boldsymbol{x} \in \mathcal{X}_i} ||\boldsymbol{x} - \boldsymbol{u_i}||^2 Imp_i \\ &= \sum_{i=1}^{K} \sum_{\boldsymbol{x} \in \mathcal{X}_i} ||\boldsymbol{x} - \boldsymbol{u_i}||^2 (1 + Imp_i) \end{aligned} \tag{4}$$

Impurity measures: Equation 4 should be applicable to any impurity measure in general. Entropy and Gini index are most commonly used impurity measures. We use the following impurity measure: $Imp_i = \mathcal{ADC}_i Ent_i$, where $\mathcal{ADC}_i$ is the *aggregated dissimilarity count* of cluster i and Ent_i is the entropy of cluster i. The reason for using this impurity measure will be explained shortly. In order to understand $\mathcal{ADC}_i$, we must first define the *dissimilarity count*.

Definition 1 (*Dissimilarity count*) Let $\boldsymbol{x}$ be a data point having class label y, and $\boldsymbol{x}$ belongs to cluster i. The dissimilarity count $\mathcal{DC}_i(\boldsymbol{x}, y)$ of $\boldsymbol{x}$ is the total number of instances in that cluster having class label other than y. Formally, we define

$$\mathcal{DC}_i(\boldsymbol{x}, y) = |\mathcal{X}_i| - |\mathcal{X}_i(y)| \tag{5}$$

where $\mathcal{X}_i(y)$ is the set of instances in cluster i having class label y.

Recall that unlabeled instances are assumed to have class label 0. Note that $\mathcal{DC}_i(\boldsymbol{x}, y)$ can be computed in constant time if we maintain an integer vector to store the counts $|\mathcal{X}_i(c)|$, $c \in \{0, 1, .., C\}$. The aggregated dissimilarity count $\mathcal{ADC}_i$ is the sum of the dissimilarity counts of all the points in cluster i:

$$\mathcal{ADC}_i = \sum_{x \in \mathcal{X}_i} \mathcal{DC}_i(\boldsymbol{x}, y). \tag{6}$$

The entropy of a cluster i is computed as

$$Ent_i = \sum_{c=0}^{C} (-p_c^i \log(p_c^i))$$

where p_c^i is the prior probability of class c:

$$p_c^i = \frac{|\mathcal{X}_i(c)|}{|\mathcal{X}_i|}. \tag{7}$$

The use of Ent_i in the objective function ensures that clusters with higher entropy are penalized more. However, if only Ent_i had been used as the impurity measure, then each point in the same cluster would have received the same penalty. But we want to favor the points belonging to the majority class in a cluster and disfavor the points belonging to the minority classes. This punishment/reward scheme forces more points of the majority class to be moved into the cluster, and more points of the minority classes to be moved out of the cluster.

To improve the purity of the clusters in this way, we introduce $\mathcal{ADC}_i$ into the equation. The combination of $\mathcal{ADC}_i$ and Ent_i is now called the *compound impurity measure* since it can be shown that $\mathcal{ADC}_i$ is proportional to the Gini index of cluster i. Following Eq. 6, we obtain

$$\begin{aligned}
\mathcal{ADC}_i &= \sum_{x \in \mathcal{X}_i} \mathcal{DC}_i(\boldsymbol{x}, y) = \sum_{c=0}^{C} \sum_{x \in \mathcal{X}_i(c)} DC_i(\boldsymbol{x}, y) \\
&= \sum_{c=0}^{C} \sum_{x \in \mathcal{X}_i(c)} (|\mathcal{X}_i| - |\mathcal{X}_i(c)|) \quad \text{(using Eq. 5)} \\
&= \sum_{c=0}^{C} |\mathcal{X}_i(c)| \, (|\mathcal{X}_i| - |\mathcal{X}_i(c)|) = |\mathcal{X}_i|^2 \sum_{c=0}^{C} \left(\frac{\mathcal{X}_i(c)}{\mathcal{X}_i} \right) \left(1 - \frac{\mathcal{X}_i(c)}{\mathcal{X}_i} \right) \\
&= |\mathcal{X}_i|^2 \sum_{c=0}^{C} p_c^i (1 - p_c^i) \quad \text{(using Eq. 7)} \\
&= |\mathcal{X}_i|^2 \left(1 - \sum_{c=0}^{C} (p_c^i)^2 \right) = |\mathcal{X}_i|^2 Gini_i
\end{aligned}$$

where $Gini_i$ is the Gini index of cluster i.

Optimizing the objective function with E-M. The problem of minimizing Eq. 4 is an *incomplete-data problem* because the cluster labels and the centroids are all unknown. The common solution to this problem is to apply E-M [13]. The E-M algorithm consists of three basic steps: initialization, E-step, and M-step. Each of them is discussed below.

Initialization with proportionate cluster distribution. For each class c appearing in the data, we initialize $k_c \leq K$ centroids by choosing k_c points from the labeled data of class c. The ratio of k_c to K is chosen to be equal to the ratio of the number of labeled points having class label c to the total number of labeled points in the dataset. That is, $k_c = K \frac{|\mathcal{L}(c)|}{|\mathcal{L}|}$, $c \in \{1, ..., C\}$, where $\mathcal{L}$ is the set of all labeled points in $\mathcal{X}$, and $\mathcal{L}(c)$ is the subset of points in $\mathcal{L}$ belonging to class c. We observed in our experiments that this initialization works better than initializing equal number of centroids of each class. This is because if we initialize the same number of centroids from each class, then larger classes (i.e., classes having more instances) tend to create larger and sparser clusters, which leads to poorer classification model.

Let there be η_c labeled points of class c in the dataset. If $\eta_c > k_c$, then we choose k_c centroids from η_c points using the farthest-first traversal heuristic [20]. To apply this heuristic, we first initialize a *visited set* of points with a randomly chosen point having class label c. At each iteration, we find a point x_j of class c that maximizes the minimum distance from all points in the visited set and add it to the visited set. This process continues until we have k_c points in the set. If $\eta_c < k_c$, then we choose remaining centroids randomly from the unlabeled points. After initialization, the E-Step and M-Step are iterated until the convergence condition is fulfilled.

E-Step. In the E-Step, we assign each data point $\boldsymbol{x}$ to a cluster i such that its contribution to the global objective function, $\mathcal{O}_{MCIKeans}(\boldsymbol{x})$, is minimized:

$$\mathcal{O}_{MCIKeans}(\boldsymbol{x}) = ||\boldsymbol{x} - \boldsymbol{u_i}||^2 (1 + Ent_i \, \mathcal{DC}_i(\boldsymbol{x}, y)) \tag{8}$$

Note that the value of the global objective function $\mathcal{O}_{MCIKeans}$ depends on the order in which the labeled points are assigned to clusters. It is computationally intractable to try all possible orderings and choose the best one. However, there are some heuristic approaches that approximate the optimal solution. We follow the *iterative conditional mode* (ICM) algorithm [8]. This is implemented as follows: At each iteration of ICM, we first randomly order the points. Then, we assign the points (in that order) to the cluster i that minimizes $\mathcal{O}_{MCIKeans}(\boldsymbol{x})$. This is continued until no point changes its cluster in successive iterations, which indicates convergence. ICM is guaranteed to converge [8]. The E-step completes after termination of ICM, and the algorithm moves to the M-step.

M-Step. In the M-Step, we re-compute each cluster centroid by averaging all the points in that cluster:

$$\boldsymbol{u_i} = \frac{\sum_{\boldsymbol{x} \in \mathcal{X}_i} \boldsymbol{x}}{|\mathcal{X}_i|} \tag{9}$$

After performing this step, the convergence condition is checked. If fulfilled, the procedure terminates, otherwise another iteration of E-Step and M-Step is performed.

4.2 Splitting clusters into pure clusters

Although most of the clusters constructed in the previous step are made as pure as possible, some of them may contain instances from a mixture of classes or may contain a mixture of labeled and unlabeled instances. The clusters are therefore split into pure clusters so that each pure cluster contains only unlabeled instances or only labeled instances from a single class. This is done by creating $C+1$ groups of instances of the cluster—one group per class plus one group for the unlabeled instances—and considering each group as a pure cluster. At this point, the reader may ask whether we could create the pure clusters in one step using K-means clustering separately for each class and for the unlabeled data in a supervised fashion rather than creating them in two steps—i.e., via semi-supervised clustering and splitting. The reason for this two-step process is that when limited amount of labels are available, semi-supervision is usually more useful than full supervision. It is likely that supervised K-means would create low quality, less dense, or more scattered clusters than semi-supervised clustering. The resulting cluster representatives (micro-clusters) would have less precision in representing the corresponding data points. As a result, the label propagation may also perform poorly.

Splitting is done as follows. Suppose $\mathcal{H}_i$ is a cluster.

Case I: $\mathcal{H}_i$ contains only unlabeled instances or only labeled instances from a single class. No splitting is necessary since the cluster is already pure.

Case II: $\mathcal{H}_i$ contains both labeled and unlabeled instances, and/or labeled instances from more than one class. For each class, we create a cluster with the instances of that class. If $\mathcal{H}_i$ contains unlabeled instances, then another cluster is created with those unlabeled instances.

The resulting pure clusters contain only unlabeled instances or labeled instances from a single class. The total number of pure clusters will be at most $(C+1)K = \hat{K}$, which is a constant. However, in practice, we find that $\hat{K}$ is almost the same as K. This is because in practice most of the macro-clusters are purely homogeneous and need not be split.

Splitting pure, unlabeled clusters. The pure, unlabeled clusters may be further split into smaller clusters. This is because, if the instances of a pure unlabeled cluster actually come

from different classes, then this cluster will have a negative effect on the label propagation. However, there is no way to accurately know the real labels of the unlabeled instances. So, we use the predicted labels of those instances that were obtained when the instances were classified using the ensemble. Therefore, the pure unlabeled micro-clusters are split into smaller clusters based on the predicted labels of the unlabeled instances. Note that cluster splitting may create clusters of different sizes. However, this does not affect classification quality since the classification function (Sect. 5.1) considers cluster size; i.e., smaller clusters have less weight and vice versa.

4.3 Storing the cluster summaries

After building and splitting the clusters, we create a summary of the statistics of the data points belonging to each cluster. The summary contains (i) the total number of points N, (ii) the total number of labeled points Lt, (iii) the class label $y \in \{1, ..., C\}$ of the instances in the cluster or 0 if the instances are unlabeled, and (iv) the centroid μ of the cluster. If ψ_i represents a micro-cluster then $\psi_i(N)$ represents the number of instances in the micro-cluster, $\psi_i(\mu)$ represents the centroid, and so on. This summary is hence referred to as a *micro-cluster*. After creating the micro-clusters, we discard the raw data points.

4.4 Label propagation

We combine all the labeled micro-clusters from the last r contiguous chunks with the micro-clusters obtained using the technique discussed above. The default value of r is 3 [39]. Since most micro-clusters are unlabeled, label propagation is applied to provide appropriate labels to all the micro-clusters. We follow the label spreading technique of Zhou et al. [43]. This approach first creates an affinity matrix based on some similarity metric from the given data points.

$$\boldsymbol{W}_{i,j} = e^{-\left(\frac{||x_i - x_j||}{2\sigma^2}\right)} \tag{10}$$

where x_i and x_j are two data points in the dataset. The affinity matrix constructs a fully connected graph where each data point represents a vertex and $\boldsymbol{W}_{i,j}$ represents the weight of the edge between the vertices x_i and x_j. We modify the Gaussian kernel affinity equation for micro-clusters as follows:

$$\boldsymbol{W}_{i,j} = e^{-\left(\frac{||\psi_i(\mu) - \psi_j(\mu)||}{2\sigma^2}\right)} \psi_j(N) \tag{11}$$

where ψ_i and ψ_j are two micro-clusters, $\psi_i(\mu)$ is the centroid of ψ_i, and $\psi_j(N)$ is the number of data points in ψ_j. The intuition behind this modification is as follows. Weight $\boldsymbol{W}_{i,j}$ is the edge weight of the *directed edge* between micro-clusters ψ_i and ψ_j. During label propagation, the weights of the neighbors of a micro-cluster partially determine the label of that micro-cluster. Thus, the larger the weight, the greater the influence of its neighbors in determining its label. In case of a single data point, as opposed to micro-clusters, the affinity matrix only depends on the similarity between the two points. In case of micro-clusters, statistic $\psi(N)$ should also be considered for each micro-cluster ψ, because each micro-cluster ψ_j actually represents $\psi_j(N)$ single data points. The influence of ψ_j on ψ_i should therefore be $\psi_j(N)$ times the influence of a single data point, and vice versa. It should be noted that the diagonal entries of the affinity matrix $\boldsymbol{W}_{i,i}$ should be zero.

After constructing the affinity matrix, the label spreading algorithm of Zhou et al. [43] is applied. First, the vector of labels is initialized:

$$Y^{(0)} = (y_1, \ldots, y_l, 0, 0, \ldots, 0) \tag{12}$$

where y_i is the class label of the ith micro-cluster and 0 is the class label for the unlabeled micro-clusters. In each iteration of the algorithm, the label of a micro-cluster is determined using a linear combination of its initial label, its current label, and the labels of its neighbors:

$$Y^{(t+1)} = \alpha \mathcal{L} Y^{(t)} + (1 - \alpha) Y^{(0)} \tag{13}$$

where α is a constant between 0 and 1, and $\mathcal{L}$ is the normalized graph Laplacian of $\boldsymbol{W}$. We use $\sigma = 0.25$, and $\alpha = 0.99$ in our experiments, which is the suggested value according to [39]. The algorithm terminates when no micro-cluster changes its label in successive iterations. For binary classification problems, the label value y_i associated with each labeled micro-cluster ψ_i is initialized with either -1 or $+1$ and that of each unlabeled micro-cluster is initialized with 0. When the algorithm terminates, the label of a micro-cluster is obtained from the sign of y. For multi-class problems, $\boldsymbol{y}_i$ is a vector having one value in interval [0, 1] for each class. For unlabeled ψ_i, the vector initially contains all zeros. For a labeled ψ_i belonging to class c, $\boldsymbol{y}_i[c]$ is set to 1 and $\boldsymbol{y}_i[j]$ is set to 0 for all other classes $j \neq c$. When the algorithm terminates, the label of a micro-cluster is chosen to be the class having the maximum value in the vector $\boldsymbol{y}_i$. These $\hat{K}$ micro-clusters are used as a classification model to classify future data. The next section discusses the ensemble updating and classification techniques.

5 Ensemble classification

The ensemble consists of L models, where each model is trained with a partially labeled data chunk according to Sect. 4. The initial ensemble consists of the first L models trained with the first L chunks in the stream. The ensemble is used to classify future unlabeled instances. The ensemble additionally undergoes several modifications in each successive chunks to keep it current with the most recent concept.

5.1 Classification

The classification function uses an inductive label propagation technique. In order to classify an unlabeled data point $\boldsymbol{x}$ with a model M^i, we use the following formula [7]

$$\hat{y} = \frac{\sum_j \boldsymbol{W}_\Psi(x, \psi_j)\hat{y_j}}{\sum_j \boldsymbol{W}_\Psi(x, \psi_j) + \epsilon} \tag{14}$$

where x is the test point, the ψ_j's are the pseudo-points in the model, $\boldsymbol{W}_\Psi$ is the function that generated matrix $\boldsymbol{W}$ on $\Psi = \{\psi_1, ..., \psi_{\hat{K}}\}$, and ϵ is a small smoothing constant to prevent the denominator from being zero.

This equation is designed for binary classification. It can also be applied to multi-class classification if $\hat{\boldsymbol{y}}$ is considered as a vector of C values in interval [0, 1], where C is the total number of classes. The predicted class will be the class whose corresponding value is maximal in $\hat{\boldsymbol{y}}$. The inductive rule was designed to classify a test instance using a single model. In order to classify a test instance using the ensemble, we compute $\hat{\boldsymbol{y}}$ for each model $M_i \in M$ separately and then sum all of the $\hat{\boldsymbol{y}}$ vectors. The maximum value in this vector sum becomes the predicted class.

5.2 Ensemble refinement

After a new model M' has been trained with a partially labeled data chunk, the existing ensemble M is refined with this model (Algorithm 1, line 3). Refinement is done if the latest partially labeled data chunk D_n contains a class c that is absent in all models of the ensemble M. This is possible if either a completely new class appears in the stream or an old class re-appears that has been absent in the stream for a long time. Both of these happen because of concept evolution, and the class c is therefore termed an *evolved class*. Note that there may be more than one evolved classes in the stream. If there is any evolved class, M must be refined so that it can correctly classify future instances of that class. Algorithm 1 describes how the existing model is refined.

Algorithm 2 Refine-Ensemble

Input: M: current ensemble of L models $\{M^1, ..., M^L\}$
M': the new model built from the new data chunk D_n
Output: Refined ensemble M
1: **if Need-to-refine**$(M) =$ *false* **then return** M
2: **for** each labeled micro-cluster $\psi_j \in M'$ **do**
3: **if** ψ_j(y) (the class label of ψ_j) is an evolved class **then**
4: **for** each model $M^i \in M$ **do**
5: $Q \leftarrow$ the closest pair of micro-clusters in M^i having the same class label
6: **if** $Q \neq$ *null* **and** $|M^i| \geq K$ **then** Merge the pair of micro-clusters in Q
7: $M^i \leftarrow M^i \cup \psi_j$ /* Injection */
8: **end for**
9: **end if**
10: **end for**
11: **return** M

The Refine-Ensemble algorithm. Algorithm 1 starts (line 1) by checking whether ensemble refinement is needed. This can be done in constant time by keeping a boolean vector $\boldsymbol{V}$ of size C per model and setting $V[c] \leftarrow$ *true* during training if there is any labeled training instance from class c. Function **Need-to-refine**(M) checks whether there is any class c such that $V[c]$ is false for all models $M^i \in M$, but true for M'. If there is such a class c, then c is an evolved class so refinement is necessary. In that case, the algorithm looks into each micro-cluster ψ_j of the new model M' (line 2). If the class label of ψ_j is an evolved class (line 3), then we do the following.

For each model $M^i \in M$, we inject micro-cluster ψ_j into M^i (line 7). Injection means adding ψ_j to model M^i. Before injecting ψ_j, we try to merge the closest pair of micro-clusters in M^i having the same class label (line 6). This is done to keep the number of micro-clusters in each model constant. However, merging is done only if both the following conditions are satisfied: (i) there exist a pair of micro-clusters having the same class label, and (ii) the total number of micro-clusters $|M^i|$ is greater than or equal to K. Note that condition (i) fails if $|M^i| \leq C$. This happens if either $|M^i| < K$ or $K \leq C$. Condition (ii) fails if $|M^i| < K$. Thus, merging is not performed if $|M^i| < K$ or $K \leq C$. The first special case $|M^i| < K$ occurs because of deleting micro-clusters (see the last paragraph of this subsection on micro-cluster deletion). We allow $|M^i|$ to be incremented after injection in this case. The second special case $K \leq C$ occurs because of concept evolution, i.e., as a result of new classes appearing in the stream. This cannot be allowed since $K > C$ must be maintained (see Sect. 4). In

this case, we not only increment $|M^i|$ with injection but also increment K (not shown in the algorithm) to maintain the $K > C$ relationship.

The reasoning behind the refinement is as follows. Since no model in ensemble M has knowledge of an evolved class c, the models will certainly misclassify any data belonging to the class. By injecting micro-clusters of class c, we introduce some data from this class into the models, which reduces their misclassification rate. It is obvious that when more training instances are provided to a model, its classification error is more likely to decrease. However, if the same set of micro-clusters are injected in all the models, the correlation among the models may increase, resulting in reduced prediction accuracy of the ensemble. If the errors of the models in an L-model ensemble are independent, then the added error (i.e., the error in addition to Bayes error) of the ensemble is $1/L$ times the added error of a single model [35]. However, the ensemble error may be higher if there is correlation among the errors of the models. Even if correlation is introduced by injecting the micro-clusters, the following lemma (Lemma 1) shows that under certain conditions, the overall added error of the ensemble is reduced after injection. The lemma is based on the assumption that after injection, single model error monotonically decreases with increasing prior probability of class c. In other words, we assume that there is a continuous monotonic decreasing function $f : [0, 1] \rightarrow [0, 1]$ such that

$$\mathcal{E} = f(\gamma_c)\mathcal{E}^0 \tag{15}$$

where $\mathcal{E}^0$ and $\mathcal{E}$ are the single model errors before and after injection (respectively), and γ_c is the prior probability of class c. Observe that $f(0) = 1$, since $\gamma_c = 0$ means class c has not appeared at all and therefore no injection has been made. Lemma 1 quantifies an upper bound of the function that is necessary for ensemble error reduction.

Lemma 1 *Let c be an evolved class, $\mathcal{E}_M^0$ and $\mathcal{E}_M$ be the added errors of the ensemble before and after injection (respectively), $\mathcal{E}^0$ and $\mathcal{E}$ be the added errors of a single model before and after injection (respectively), and γ_c be the prior probability of class c. The injection process will reduce the added error of the ensemble provided that*

$$f(\gamma_c) \leq \frac{1}{1 + \gamma_c^2(L-1)}.$$

where L is the ensemble size.

Proof According to [35],

$$\mathcal{E}_M = \mathcal{E}\frac{1 + \delta(L-1)}{L} \tag{16}$$

where L is the total number of models in the ensemble, and δ is the mean correlation among the models, given by:

$$\delta = \sum_{i=1}^{C} \gamma_i \delta_i \tag{17}$$

where γ_i is the prior probability of class i and δ_i is the mean correlation associated with class i, given by:

$$\delta_i = \frac{1}{L(L-1)} \sum_{m=1}^{L} \sum_{l \neq m} Corr(\eta_i^m, \eta_i^l) \tag{18}$$

where $Corr(\eta_i^m, \eta_i^l)$ is the correlation between the error η_i^m of model m and the error η_i^l of model l. For simplicity, we assume that the correlation between two models is proportional to the number of instances that are common to both these models. That is, the correlation is 1 if they have all instances in common, and 0 if they have no instances in common. Thus, before injection, the correlation between any pair of models is zero (since the models are trained using disjoint training data). As a result,

$$\mathcal{E}_M^0 = \frac{\mathcal{E}^0}{L} \tag{19}$$

After injection, some instances of class c may be common among a pair of models, leading to $\delta_c \geq 0$, where c is the evolved class.

Consider a pair of models m and l whose prior probabilities of class c are γ_c^m and γ_c^l, respectively, after injection. So, the correlation between m and l reduces to:

$$Corr(\eta_c^m, \eta_c^l) = \tfrac{1}{2}(\gamma_c^m + \gamma_c^l)$$

Substituting this value into Eq. 18, we obtain

$$\begin{aligned}\delta_c &= \frac{1}{L(L-1)}\frac{1}{2}\sum_{m=1}^{L}\sum_{l\neq m}(\gamma_c^m + \gamma_c^l) \\ &= \frac{1}{L(L-1)}\frac{1}{2}2(L-1)\sum_{m=1}^{L}\gamma_c^m = \frac{1}{L}\sum_{m=1}^{L}\gamma_c^m = \bar{\gamma}_c\end{aligned} \tag{20}$$

where $\bar{\gamma}_c$ is the mean prior probability of class c in each model. Note that the mean prior probability $\bar{\gamma}_c$ represents the actual prior probability γ_c, so they can be used interchangeably. Substituting this value of δ_i in Eq. 17 yields

$$\delta = \sum_{i=1}^{C}\gamma_i\delta_i = \gamma_c\delta_c + \sum_{i=1, i\neq c}^{C}\gamma_i\delta_i = (\gamma_c)^2 + 0 = (\gamma_c)^2$$

since $\delta_i = 0$ for all non-evolved classes (since no instance of those classes is common between any pair of models). Substituting this value of δ into Eq. 16, we obtain

$$\begin{aligned}\mathcal{E}_M &= \mathcal{E}\frac{1+{\gamma_c}^2(L-1)}{L} \\ &= f(\gamma_c)\mathcal{E}^0\frac{1+{\gamma_c}^2(L-1)}{L} \quad \text{(using Eq. 15)} \\ &= \frac{\mathcal{E}^0}{L}f(\gamma_c)(1+{\gamma_c}^2(L-1)) \\ &= \mathcal{E}_M^0 f(\gamma_c)(1+{\gamma_c}^2(L-1)) \quad \text{(using Eq. 19)}\end{aligned}$$

We will therefore have an error reduction provided that $\mathcal{E}_M \leq \mathcal{E}_M^0$, which leads to:

$$\begin{aligned}f(\gamma_c)(1+\gamma_c^2(L-1)) &\leq 1 \\ f(\gamma_c) &\leq \frac{1}{1+\gamma_c^2(L-1)}\end{aligned} \tag{21}$$

□

From Lemma 1, we infer that function f becomes more restricted as γ_c or L increase. For example, for $\gamma_c = 0.5$, if $L = 10$ then $f(\gamma_c) \leq 0.31$, meaning that $\mathcal{E} \leq 0.31\mathcal{E}_0$ is required for error reduction. For the same value of γ_c, if $L = 2$ then $\mathcal{E} \leq 0.8\mathcal{E}_0$ is required for error reduction. However, in our experiments, we always observe error reduction after injection, i.e., Eq. 21 has always been satisfied. Still, we recommend that the value of L be kept within 10 for minimizing the risk of violating Eq. 21.

Micro-cluster deletion. In order to improve the classification accuracy of a classifier M^i, we occasionally remove micro-clusters that may have negative effect on the classification accuracy. For each micro-cluster $\psi \in M^i$, we maintain the accuracy of ψ as $A(\psi)$. $A(\psi)$ is the percentage of manually labeled instances for which ψ is a nearest neighbor and whose class label is the same as that of ψ. If for any labeled instance x, its nearest micro-cluster ψ has a different class label than x, then $A(\psi)$ drops. This statistic helps determine whether any micro-cluster has been wrongly labeled by the label propagation, or if the micro-cluster has become outdated because of concept drift. In general, we delete ψ if $A(\psi)$ drops below 70%.

5.3 Ensemble update

After the refinement, the ensemble is updated to adapt to the concept drift in the stream. This is done as follows. We have now $L+1$ models—L models from the ensemble plus the newly trained model M'. One of these $L+1$ models is discarded, and the rest of them comprise the new ensemble. The discarded victim is chosen by evaluating the accuracy of each of these $L+1$ models on the labeled instances in the training data D_n. The model having the worst accuracy is discarded.

5.4 Time complexity

The ensemble training process consists of four main steps: (1) creating clusters using E-M, (2) splitting clusters and label propagation, (3) refining the ensemble, and (4) updating the ensemble. Step 2 requires $O(\hat{K}^3 L)$ time, and Step 3 requires $O(\hat{K} L (P/100) S)$ time, where P is the percentage of labeled data ($P \leq 100$) in the chunk and S is the chunk size. Step 1 (E-M) requires $O(KSI_{icm}I_{em})$ time, where I_{icm} is the average number of ICM iterations per E-step and I_{em} is the total number of E-M iterations.

Although it is not possible to find the exact values of I_{icm} and I_{em} analytically, we obtain an approximation by observation. We observe from our experiments that I_{em} depends only on the chunk size S, and I_{icm} is constant ($I_{icm} \approx 2$) for any dataset. On average, a data chunk having 1,000 instances requires 10 E-M iterations to converge. This increases sub-linearly with chunk size. For example, a 2,000 instance chunk requires 14 E-M iterations and so on. There are several reasons for this fast convergence of E-M, including: (1) proportionate initial seed selection from the labeled data using farthest-fast traversal, and (2) using the compound impurity measure in the objective function.

Therefore, the overall time complexity of the ensemble training process of ReaSC is $O(\hat{K}^3 L + \hat{K} L (P/100) S + K S g(S))$, where g is a sub-linear function. This complexity is almost linear in S for a moderate chunk size. The time complexity of ensemble classification is $O(\hat{K} L S)$, which is also linear in S for a fixed value of $\hat{K}$ and L. Note that K is the number of initial clusters built from each chunk using semi-supervised clustering, and $\hat{K}$ is the number of total micro-clusters obtained by splitting the clusters into pure clusters.

6 Experiments

In this section, we discuss the data sets used in the experiments, the system setup, and the results.

6.1 Dataset

We apply our technique on two synthetic and two real datasets. The synthetic datasets we generate both simulate concept drift, but one simulates concept evolution, as well. One of the two real datasets is the 10% version of the KDD cup 1999 intrusion detection dataset [23]. The other one is the aviation safety reporting systems (ASRS) dataset obtained from NASA [33]. All of these datasets are discussed in the following paragraphs.

Concept drifting synthetic dataset (SynD). We use this dataset in order to show that our approach can handle concept drift. SynD data are generated using a moving hyperplane technique. The equation of a hyperplane is as follows:

$$\sum_{i=1}^{d} a_i x_i = a_0.$$

where d is the total number of dimensions, a_i is the weight associated with dimension i, and x_i is the value of ith dimension of a datapoint x. If $\sum_{i=1}^{d} a_i x_i \leq a_0$, then an example is considered negative, otherwise it is considered positive. Each instance is a randomly generated d-dimensional vector $\{x_1, ..., x_d\}$, where $x_i \in [0, 1]$. Weights $\{a_1, ..., a_d\}$ are also randomly initialized with a real number in the range [0,1]. The value of a_0 is adjusted so that roughly the same number of positive and negative examples are generated. This can be done by choosing $a_0 = \frac{1}{2}\sum_{i=1}^{d} a_i$. We also introduce noise randomly by switching the labels of $p\%$ of the examples, where $p = 5$ in our experiments.

There are several parameters that simulate concept drift. Parameter m specifies the percent of total dimensions whose weights change, and it is set to 20%. Parameter t specifies the magnitude of the change in every N examples. In our experiments, t varies from 0.1 to 1.0, and $N = 1,000$. For all $i \in \{1, \ldots, d\}$, s_i specifies the direction of change for each weight. Weights change continuously; a_i is adjusted by $s_i t/N$ after each example is generated. There is a possibility of $r\%$ that the change would reverse direction after every N examples are generated. In our experiments, r is 10%. We generate a total of 250,000 instances and divide them into equal-sized chunks.

Concept drifting with concept-evolving synthetic dataset (SynDE). The SynDE dataset simulates both concept drift and concept evolution. That is, new classes appear in the stream, old classes disappear, and the concept for each class gradually changes over time. The dataset size varies from 100 to 1,000 K points, the number of class labels varies from 5 to 40, and data dimensions vary from 20 to 80. Data points belonging to each class are generated by following a normal distribution having different mean (−5.0 to +5.0) and variance (0.5–6) for different classes. That is, each class is assigned a different mean and variance for the feature vector. In order to simulate the evolving nature of data streams, the prior probabilities of different classes are varied with time. This causes some classes to appear and other classes to disappear at different times in the stream history. In order to simulate the drifting nature of the concepts, the mean values for the feature vector for each class are gradually changed in a way similar to the SynD dataset. We name the different synthetic datasets using the

syntax ⟨*size*⟩*C*⟨*# of classes*⟩*D*⟨*# of dimensions*⟩. For example, dataset 300KC5D20 has 300 K points, 5 classes, and 20 dimensions.

KDDCup 99 network intrusion detection real dataset (KDD). This dataset contains TCP connection records extracted from LAN network traffic at MIT Lincoln Labs over a period of two weeks. We have used the 10% version of the dataset, which is more concentrated than the full version. Here, different classes appear and disappear frequently. Each instance in the dataset refers to either to a normal connection or an attack. There are 22 types of attacks, such as buffer-overflow, portsweep, guess-passwd, neptune, rootkit, smurf, spy, etc. So, there are 23 different classes of data, most of which are normal. Each record consists of 42 attributes, such as connection duration, the number bytes transmitted, number of root accesses, etc. We use only the 34 continuous attributes and remove the categorical attributes.

Aviation safety reporting systems real dataset (ASRS). This dataset contains around 150,000 text documents. Each document is a report corresponding to a flight anomaly. There are a total of 55 anomalies, such as "aircraft equipment problem : critical", "aircraft equipment problem : less severe", "inflight encounter : birds", "inflight encounter : skydivers", "maintenance problem : improper documentation" etc. Each of these anomalies is considered a class. These documents represent a data stream since the dataset contains the reports in order of their creation time and new reports are added on a regular basis.

We perform several preprocessing steps on this dataset. First, we discard the classes that contain very few (less than 100) documents. We choose 21 classes among the 55, which reduced the total number of selected documents to 125,799. Second, we extract word features from this corpus and select the best 1,000 features based on information gain. Then, each document is transformed into a binary feature vector, where the value corresponding to a feature is 1 if the feature (i.e., word) is present, or 0 if it is not present in the document. The instances in the dataset are multi-label, meaning that an instance may have more than one class label. We transform the multi-label classification problem into 21 separate binary classification problems by generating 21 different datasets from the original dataset, one for each class. The dataset for the ith class is generated by marking the instances belonging to class i as positive, and all other instances as negative. When reporting the accuracy, we report the average accuracy of the 21 datasets.

6.2 Experimental setup

Hardware and software. We implement the algorithms in Java. The experiments were run on a Windows-based Intel P-IV machine with 2GB of memory and a 3 GHz dual processor CPU.

Parameter settings The default parameter settings are as follows, unless stated otherwise: (i) the number of micro-clusters $K = 50$ for all datasets; (ii) the chunk size $S = 1{,}600$ records for real datasets, and $S = 1{,}000$ records for synthetic datasets; and (iii) the ensemble size $L = 6$ for all datasets.

Baseline method To evaluate our algorithm, we compare it with *On Demand Stream* (OnDS) [2]. OnDS also uses a cluster-based classification model. The classification technique is based on the k-NN technique, which is similar to ours. Because of these similarities, we use OnDS as the baseline technique for comparison. However, there are two main differences between OnDS and ReaSC. First, OnDS uses a supervised learning technique, whereas

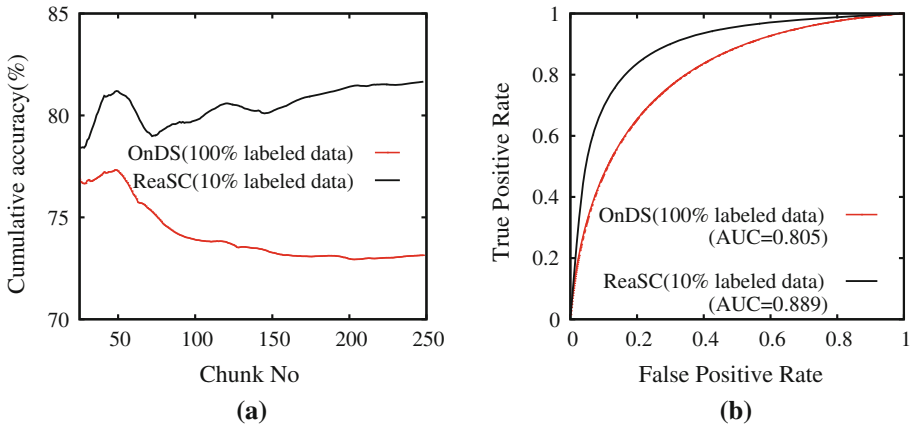


Fig. 2 Cumulative accuracy (**a**) and ROC curve (**b**) for SynD dataset

ReaSC uses semi-supervised technique. Second, OnDS saves historical snapshots and uses them for classification, whereas ReaSC uses a fixed-sized ensemble for classification. We use the default values of all parameters of OnDS, with a buffer size and stream speed of 1,600 and 80 (respectively) for real datasets, and 1,000 and 200 (respectively) for synthetic datasets, as proposed by the authors. Since OnDS assumes two separate streams, one labeled and the other unlabeled, we also apply the same assumption for ReaSC for a fair comparison. This is done by considering each even instance of a stream as a training instance and each odd instance as a test instance.

When training ReaSC, we assume that only 10% of randomly chosen instances in a training chunk have labels (i.e., $P = 10$), whereas for training OnDS, 100% instances in the training chunk are assumed to have labels. Thus, *if there are 100 data points in a training chunk, then OnDS has 100 labeled training data points, but ReaSC has only 10 labeled and 90 unlabeled training instances*. Also, for a fair comparison, the chunk size of ReaSC is always kept equal to the buffer size of OnDS. Note that P is not a parameter of ReaSC, rather, it is a threshold assigned by the user based on the system resources available for labeling data points.

Evaluation. For comparison with baseline: For each competing approach, we use the first *warm-up* chunks to build the initial classification model, which can be thought of as an warm-up period. From the *warm-up* + 1st chunk onward, we use each even instance in the chunk to update the model and test the model against each odd instance (i.e., test instance) in the chunk. Each method is run 20 times on each dataset, and the average result is reported. We use *warm-up*=3 for all datasets.

Evaluation other than comparison: For scalability test (Sect. 6.5), parameter sensitivity (Sect. 6.6), impact of unlabeled data and different components (Sect. 6.4), we apply the usual setting of ReaSC as discussed in Sect. 3 (i.e., single stream).

6.3 Comparison with baseline methods

Figures 2a, 3, 4, 5b compare the accuracies and receiver operating characteristic (ROC) curves for each dataset. Each of these graphs is generated by averaging 20 runs for each method for the same parameter settings.

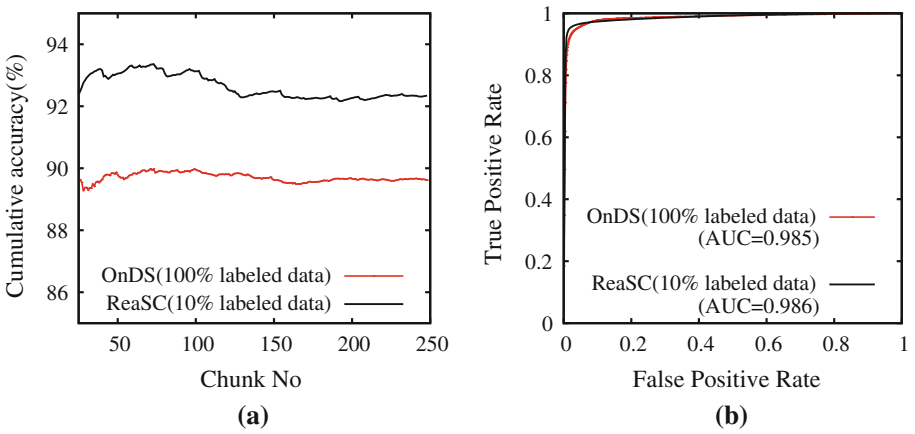


Fig. 3 Cumulative accuracy (**a**) and ROC curve (**b**) for SynDE dataset

Figure 2a shows the cumulative accuracies of each competing method for each chunk on SynD dataset. In this figure, the X-axis represents the chunk number and the Y-axis represents the accuracy of a particular method from the beginning of the stream. For example, in Fig. 2a at chunk 250 ($X = 250$), the Y values for ReaSC and OnDS represent the cumulative accuracies of ReaSC and OnDS from the beginning of the stream to chunk 250, which are 81.7 and 73.1%, respectively. This curve shows that as the stream progresses, accuracy of OnDS declines. This is because OnDS is not capable of handling concept drift properly. Figure 2b shows the ROC curves for SynD dataset. The area under the ROC curve (AUC) is higher for a better classifier. The AUCs for each ROC curve is reported in each graph. For the SynD dataset, AUC of ReaSC is almost 9% higher than that of OnDS.

Figure 3a shows the chunk number versus cumulative accuracy for the SynDE dataset (250KC10D20). The ROC curves on this dataset are shown in Fig. 3b. Figure 4a, b show the chunk number versus the cumulative accuracy and the ROC curves for the KDD dataset. The KDD dataset has a lot of concept evolution, almost all of which occurs within the first 120 chunks. The accuracy of OnDS is 2–8% lower than ReaSC in this region. This shows that ReaSC handles concept evolution better than OnDS in real data in addition to synthetic data. The ROC curves shown in Fig. 4b also reflect the performances of these two methods. The AUC of ReaSC is found to be 5% higher than OnDS. Finally, Figures 5a, b show the accuracy and ROC curves for the ASRS dataset. Recall that these graphs are generated by averaging the accuracies and ROC curves from 21 individual binary classification results. Again, ReaSC achieves 5% or higher accuracy than OnDS in all stream positions. In addition, the AUC of ReaSC in this dataset is about 8% higher than that of OnDS.

Again, recall that in all these experiments OnDS uses 10 times more labeled data for training than ReaSC. Nevertheless, ReaSC outperforms OnDS in all datasets, both in accuracy and AUC. The reasoning for the better performance of ReaSC is explained in more detail in Sect. 7.

6.4 Impact of unlabeled training data and different components of ReaSC

ReaSC utilizes a small amount of labeled data and a large amount of unlabeled data for training. The power of semi-supervised learning comes from the ability of ReaSC to utilize this unlabeled training data. Figure 6 reports this fact. In this figure, the accuracy on SynD

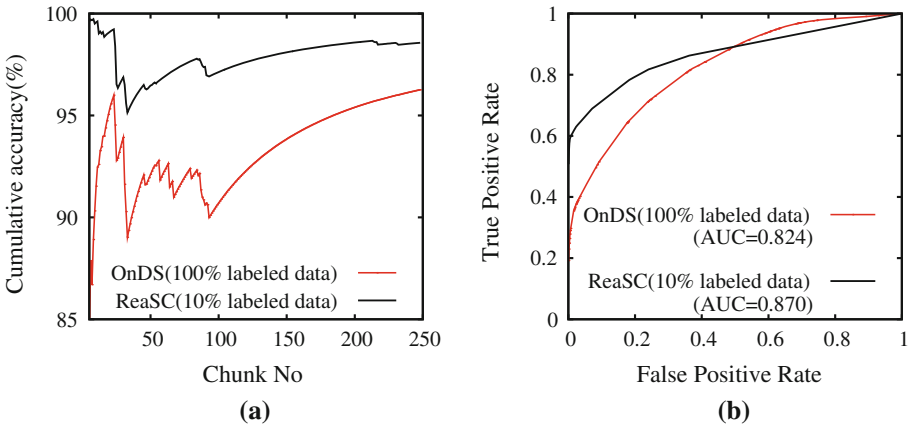


Fig. 4 Cumulative accuracy (**a**) and ROC curve (**b**) for KDD dataset

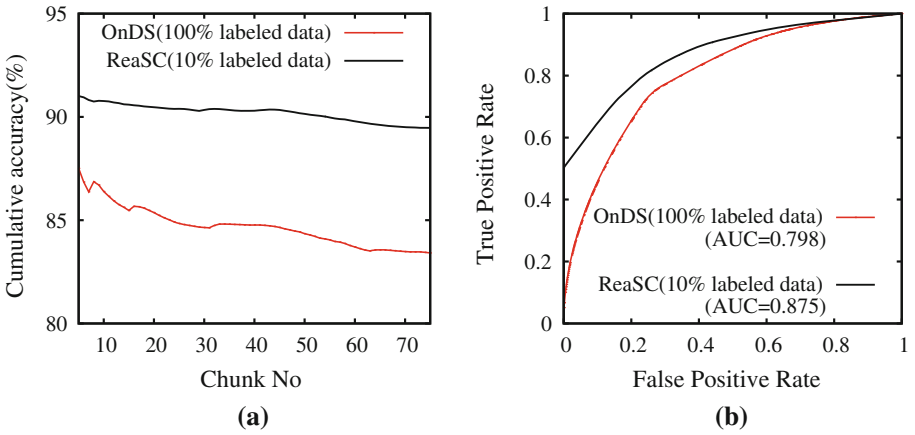


Fig. 5 Cumulative accuracy (**a**) and ROC curve (**b**) for ASRS dataset

dataset is reported for different amounts of unlabeled training data for different percentages of labeled training data. Here, the chunk size is 1,000. For example, when 10% labeled data (i.e., 100 labeled data instances per chunk) are used for training, we observe that increasing the number of unlabeled training data instances from 10 to 200 increases the overall accuracy from 73 to 80%, which is a 7% improvement. Further increasing the amount of unlabeled data also improves the accuracy, but at a slower rate. If all 900 unlabeled data are used, the accuracy becomes 82%. This trend is also observed for other percentages of labeled training data (e.g., 15, 20% etc.), and also in other datasets. The reason for this improvement is that the proposed semi-supervised approach is able to learn from both labeled and unlabeled data. If we add more unlabeled data for training, the learner is equipped with more information and able to build a better classification model.

Note that there are different components of ReaSC, such as label propagation, ensemble framework, and ensemble refinement. The following results highlight the contribution of these three components to the overall accuracy of ReaSC. Figure 7a, b show the contribution of these components in the SynDE and KDD datasets, respectively. For example, in the

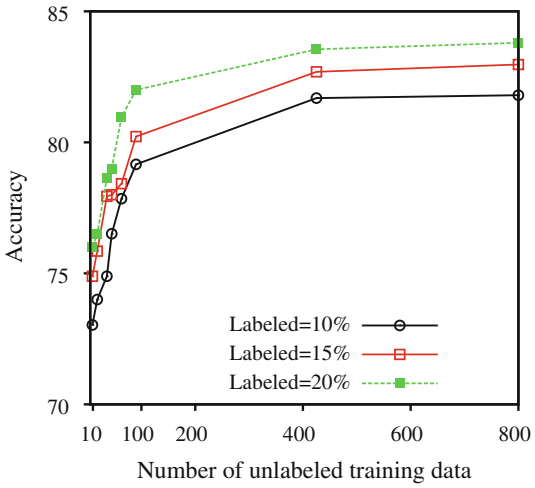


Fig. 6 Impact of unlabeled training data

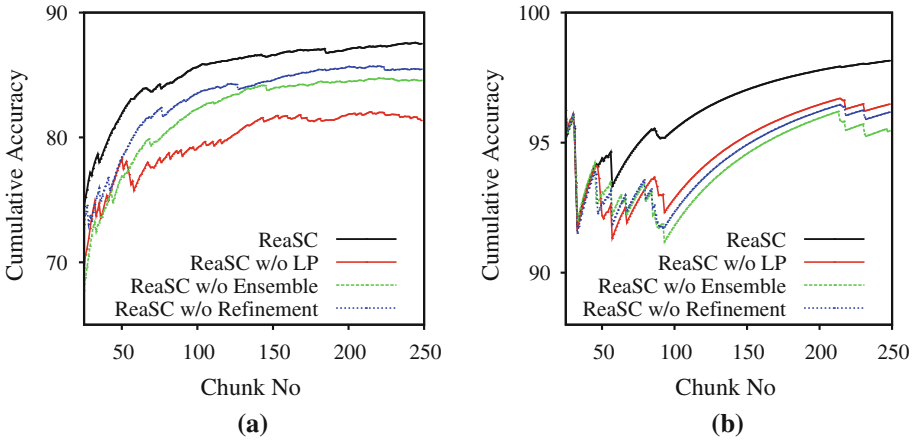


Fig. 7 Accuracy of ReaSC with and without different components **a** SynDE dataset and **b** KDD dataset

KDD dataset (Fig. 7b), the curve labeled "ReaSC" represents the combined approach with all components, the one labeled "ReaSC w/o LP" represents ReaSC without the label propagation component, that labeled "ReaSC w/o Ensemble" represents ReaSC with a single-model approach ($L = 1$), and that labeled "ReaSC w/o Refinement" represents ReaSC without the ensemble refinement function. It is obvious from the graphs that each component has some significant contribution to the combined accuracy. For example, "ReaSC w/o LP" has about 2 and 6% less accuracy than ReaSC in the KDD and SynDE datasets, respectively. Similarly, the ensemble refinement function also contributes 2% or more to the overall accuracy.

6.5 Running times, scalability, and memory requirement

Table 3 compares the running times and classification speeds between ReaSC and OnDS. The columns headed by "Time (s/1,000 pts)" report the total running times (training plus testing) in seconds per thousand points of each of these methods.

Table 3 Comparison of running time (excluding labeling time) and classification speed between OnDS (with 100% labeled data) and ReaSC (with 10% labeled data)

Dataset	Time (s/1,000 pts)		Classification speed (pts/s)	
	OnDS (100% labeled)	ReaSC (10% labeled)	OnDS (100% labeled)	ReaSC (10% labeled)
SynD	0.67	0.41	798	2,122
SynDE	1.16	0.47	471	2,045
KDD	1.78	0.83	296	2,762
ASRS	23.00	12.08	23	91

Table 4 Comparison of running time including labeling time for real datasets

Dataset	Labeling time (s/1,000 pts)		Total time (s/1,000 pts)	
	OnDS (100% labeled)	ReaSC (10% labeled)	OnDS (100% labeled)	ReaSC (10% labeled)
KDD	1,000	100	1,001.78	100.83
ASRS	60,000	6,000	60,023.00	6,012.08

Note that these running times do not consider the data labeling time, which is an essential part of classifier training and a major bottleneck for OnDS, to be explained shortly. The columns headed by "classification speed (pts/s)" report classification (testing) speed of each of these methods in points per second. The total running times of ReaSC are lower than OnDS in all datasets. It is worth mentioning that the dimensions of the datasets are in increasing order (SynD = 10, SynDE = 20, KDD = 34, and ASRS = 1,000), as are the running times.

Running times of both OnDS and ReaSC appear to grow linearly with increasing dimensionality and class labels; however, Table 3 shows that the running time of OnDS grows at a greater rate than that of ReaSC. This is because there is a classification overhead associated with OnDS that increases with stream length, data dimension, and number of class labels. OnDS keeps snapshots of the micro-clusters for different time-stamps in stream history. When classification is needed, OnDS needs to find the best time horizon by searching through the saved snapshots. This searching time is directly related with the data dimension, number of class labels, and stream length. This overhead is not present in ReaSC. As a result, OnDS takes relatively higher time on higher dimensions and larger datasets than ReaSC. Classification speed of OnDS is also much lower than ReaSC for the same reason, as shown in the table.

If we include data labeling time, we get a more realistic picture of the total running time. Suppose the labeling time for each data point for KDD dataset is 1 s, and the same for ASRS dataset is 60 s. In fact, real annotation times would be much higher for any text dataset [36]. Table 4 shows the comparison. The labeling time for OnDS is 10 times higher than that of ReaSC, since OnDS requires 1,000 labels for every 1,000 instances, whereas ReaSC requires only 200 labels for the same chunk. The net effect is that ReaSC is 10 times faster than OnDS in both datasets.

In Figure 8, we report the scalability of ReaSC on high-dimensional and multi-class SynDE data. This graph reports the running times of ReaSC for different dimensions (20–60) of synthetic data with different numbers of classes (10–40). Each of these synthetic datasets has 250 K points. For example, for $C = 10$, and $D = 20$, the running time is 431 seconds, and it increases linearly with the number of classes in the data. On the other hand, for a

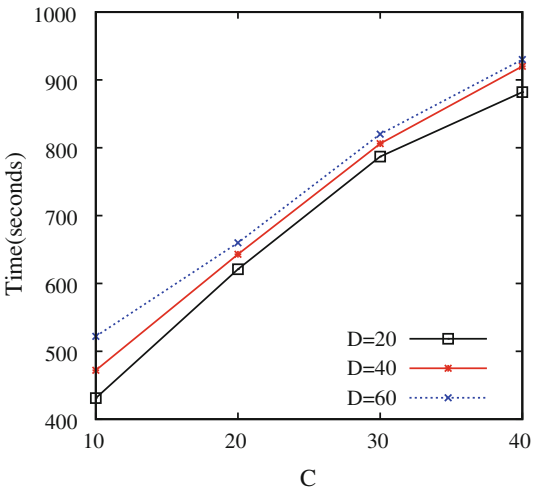


Fig. 8 Running times on different datasets having higher dimensions D and number of classes C

particular value of C, the running time increases very slowly (linearly) with increasing the number of dimensions in the data. For example, for $C = 10$, running times for 20, 40, and 60 dimensions are 431, 472, and 522 s, respectively. Thus, we may conclude that ReaSC scales linearly to higher dimensionality and class labels.

The memory requirement for ReaSC is $O(DKL)$, whereas that of OnDS is $O(DQ\log(N))$, where N is the total length of the stream, Q is the total number of micro-clusters saved per frame, and $\log(N)$ is the total number of frames [2]. A frame in OnDS refers to a set of saved snapshots, and a snapshot is a set of micro-clusters. In general, $Q > KL$, and therefore, ReaSC requires less memory than OnDS. Besides, the memory requirement of ReaSC is independent of the stream length as D, K, L all are independent of the stream length, whereas that of OnDS grows with stream length (because of the $\log(N)$ term). For example, for the ASRS dataset ($D = 1,000$), ReaSC requires less than 10MB memory, whereas OnDS requires approximately 700 MB memory.

6.6 Sensitivity to parameters

All the following results are obtained using a SynDE dataset (250KC10D20). Figure 9 shows how accuracy varies with chunk size S and the percentage P of labeled instances in each chunk. It is obvious that higher values of P leads to better classification accuracy since each model is better trained. For any particular chunk size, the improvement gradually diminishes as P approaches to 100. For example, a stream with $P = 10$ has 5 times more labeled data than the one with $P = 2$. As a result, there is a rapid accuracy improvement from $P = 2$ to $P = 10$. But a stream with $P = 75$ has only 1.5 times more labeled data than a stream with $P = 50$, so the accuracy improvement in this case is much less than in the former case. We also observe higher accuracy for larger chunk sizes. This is because as chunk size increases, each model gets trained with more data, which leads to a better classification accuracy. This improvement also diminishes gradually because of concept drift. According to [38], if there is concept drift in the data, then a larger chunk contains more outdated points, canceling out any improvement expected to be gained by increasing the training set size.

Figure 10a shows how classification accuracy varies for ReaSC with the number of micro-clusters K. We observe that higher values of K lead to better classification accuracies.

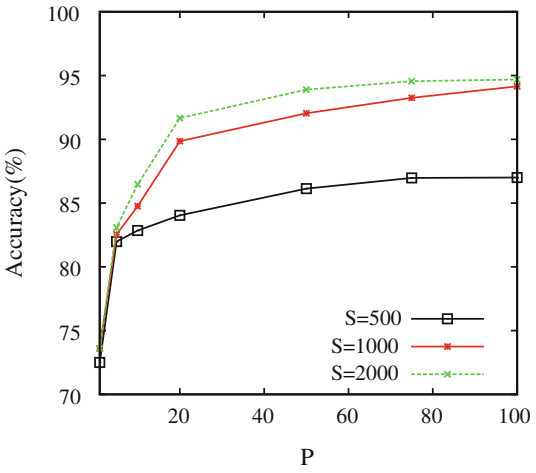


Fig. 9 Sensitivity to chunk size S for different percentages P of labeled data

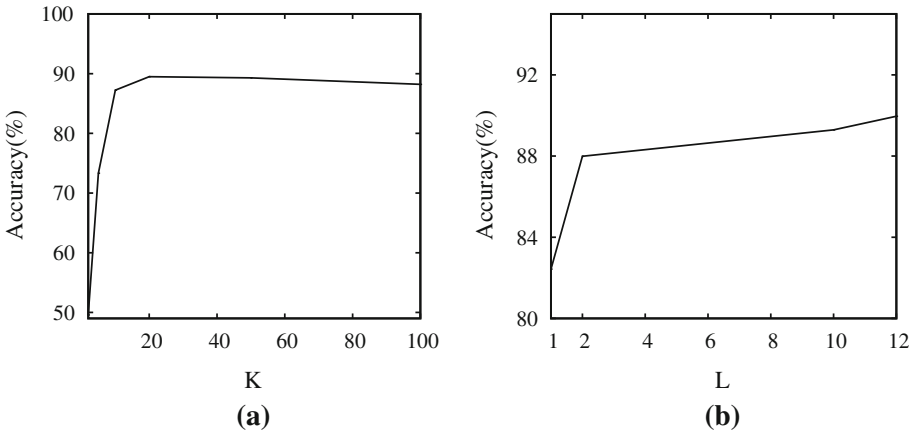


Fig. 10 Sensitivity to number of clusters K (**a**) and ensemble size L (**b**)

This happens because when K is larger, smaller and more compact clusters are formed, which contributes to a better ground for label propagation and classification. However, there is no significant improvement after $K = 50$ for this dataset, where $C = 10$. It should be noted that K should always be much larger than C. Experimental results suggests that K should be between $2C$ and $5C$ for best performance.

Figure 10b shows the effect of accuracy on ensemble size L. Ideally, increasing the ensemble size helps to reduce error [35]. Significant improvement is achieved by increasing the ensemble size from 1 (i.e., single classifier) to 2. After that, the improvement diminishes gradually. Increasing the ensemble size also increases the classification time and causes correlation among the classifiers to increase in the event of concept evolution, diminishing the improvement intended by the ensemble. A reasonable value should therefore be chosen depending on the specific requirements of the system.

7 Discussion

The above results show that ReaSC outperforms OnDS in all tested datasets. There are two main reasons behind this. First, ReaSC considers both the dispersion and impurity measures in building clusters, but OnDS considers only purity, since it applies a K-means algorithm to each class separately. In addition, ReaSC uses proportionate initialization so that more clusters are formed for the larger classes (i.e., classes having more instances), whereas OnDS builds an equal number of clusters for each class, so clusters belonging to larger classes tend to be bigger (and more sparse). Thus, the clusters of ReaSC are likely to be more compact than those of the OnDS. This results in a better classification model for ReaSC. Second, ReaSC applies ensemble classification rather than the *horizon fitting* technique used in OnDS. Horizon fitting selects a horizon of training data from the stream that corresponds to a variable-length window of the most recent (contiguous) data chunks. It is possible that one or more chunks in that window have been outdated, resulting in a less accurate classification model. This is because the set of training data that is the best representative of the current concept are not necessarily contiguous. In contrast, ReaSC always keeps the best training data (i.e., models) that are not necessarily contiguous. The ensemble approach is therefore more flexible in retaining the most up-to-date set of training data, resulting in a more accurate classification model.

It would be interesting to compare ReaSC with some other baseline approaches. First, consider a *single* combined model that contains all the KL clusters in ensemble M. We argue that this combined model is no better than the ensemble of models because our analysis shows that increasing the number of clusters beyond a certain threshold (e.g., 100) does not improve classification accuracy. Since K is chosen to be close to this threshold, it is most likely that we would not get a better model out of the KL clusters. Second, consider a single model having K clusters (not exceeding the threshold) built from L data chunks. Increasing the training set size would most likely improve classification accuracy. However, in the presence of concept drift, it can be shown that a single model built from L consecutive data chunks has a prediction error no less than an ensemble of L models, each built on a single data chunk [38]. This also follows from our experimental results that a single model built on L chunks has 5–10% worse accuracy than ReaSC and is at least L times slower than ReaSC.

8 Conclusion

We address a more realistic problem of stream mining: training with a limited amount of labeled data. Our technique is a more practical approach to the stream classification problem than many prior work since it requires less labeled data, saving much time and cost that would be otherwise required to manually label the data.

We propose and implement a semi-supervised, clustering-based stream classification algorithm to solve this limited labeled data problem. We show that our approach can achieve comparable performance by utilizing a limited amount of labeled data and large amount of unlabeled data. We evaluated our technique on two synthetically generated datasets and two real datasets and achieved better classification accuracies than state-of-the-art stream classification approaches in all datasets.

In future work, we would like to incorporate feature weighting and distance learning in the semi-supervised clustering, which should lead to a better classification model. We would also like to apply our technique to classify other real stream data.

Acknowledgments This material is based upon work supported by NASA under Award No. NNX08AC35A and the Air Force of Scientific Research (AFOSR) under Award No. FA9550-08-1-0260.

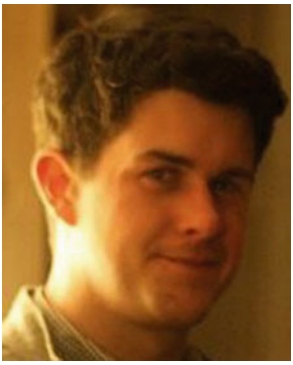
References

1. Aggarwal CC (2009) On classification and segmentation of massive audio data streams. Knowl Inf Syst 20:137–156
2. Aggarwal CC, Han J, Wang J, Yu PS (2006) A framework for on-demand classification of evolving data streams. IEEE Trans Knowl Data Eng 18(5):577–589
3. Aggarwal CC, Yu PS (2010) On clustering massive text and categorical data streams. Knowl Inf Syst 24:171–196
4. Basu S, Banerjee A, Mooney RJ (2002) Semi-supervised clustering by seeding. In: Procedings of nineteenth international conference on machine learning (ICML), Sydney, Australia, pp 19–26
5. Basu S, Banerjee A, Mooney RJ (2004) Active semi-supervision for pairwise constrained clustering. In: Proceedings of SIAM international conference on data mining (SDM), Lake Buena Vista, FL, pp 333–344
6. Basu S, Bilenko M, Banerjee A, Mooney RJ (2006) Probabilistic semi-supervised clustering with constraints'. In: Chapelle O, Schoelkopf B, Zien A (eds) Semi-supervised learning. pp 73–102
7. Bengio Y, Delalleau O, Le Roux N (2006) Label propagation and quadratic criterion. In: Chapelle O, Schölkopf B, Zien ASemi-Supervised Learning. MIT Press, Cambridge pp 193–216
8. Besag J (1986) On the statistical analysis of dirty pictures. J R Stat Soc Ser B (Methodological) 48(3):259–302
9. Bilenko M, Basu S, Mooney RJ (2004) Integrating constraints and metric learning in semi-supervised clustering. In: Proceedings of 21st international conference on machine learning (ICML), Banff, Canada, pp 81–88
10. Chen S, Wang H, Zhou S, Yu P (2008) Stop chasing trends: discovering high order models in evolving data. In: Proceedings of ICDE, pp 923–932
11. Cohn D, Caruana R, McCallum A (2003) Semi-supervised clustering with user feedback. Technical report TR2003-1892, Cornell University
12. Demiriz A, Bennett KP, Embrechts MJ (1999) Semi-supervised clustering using genetic algorithms. In: Artificial neural networks in engineering (ANNIE-99). ASME Press, pp 809–814
13. Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the em algorithm. J R Stat Soc B 39:1–38
14. Domingos P, Hulten G (2000) Mining high-speed data streams. In: Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining KDD. ACM Press, Boston MA, USA, pp 71–80
15. Fan W (2004) Systematic data selection to mine concept-drifting data streams. In: Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining (KDD), Seattle, WA, USA, pp 128–137
16. Fan W, an Huang Y, Wang H, Yu PS (2004) Active mining of data streams. In: Proceedings of SDM '04'. pp 457–461
17. Gao J, Fan W, Han J (2007) On appropriate assumptions to mine data streams. In: Proceedings of seventh IEEE international conference on data mining (ICDM), Omaha, NE, USA, pp 143–152
18. Grossi V, Turini F (2011) Stream mining: a novel architecture for ensemble-based classification in preprints. knowl Inf Syst
19. Halkidi M, Gunopulos D, Kumar N, Vazirgiannis M, Domeniconi C (2005) A framework for semi-supervised learning based on subjective and objective clustering criteria. In: Proceedings of fifth IEEE international conference on data mining (ICDM), Houston, Texas, USA, pp 637–640
20. Hochbaum D, Shmoys D (1985) A best possible heuristic for the k-center problem. Math Oper Res 10(2):180–184
21. Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: Proceedings of seventh ACM SIGKDD international conference on Knowledge discovery and data mining (KDD), San Francisco, CA, USA, pp 97–106
22. Katakis I, Tsoumakas G, Vlahavas I (2010) Tracking recurring contexts using ensemble classifiers: an application to email filtering. Knowl Inf Syst 22:371–391
23. KDD Cup 1999 Intrusion Detection Dataset (n.d.) http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

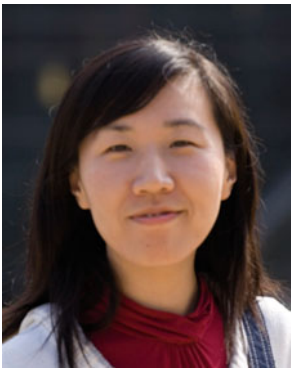
24. Klein D, Kamvar SD, Manning CD (2002) From instance-level constraints to space-level constraints: making the most of prior knowledge in data clustering. In: Proceedigs of 19th international conference on machine learning (ICML). Morgan Kaufmann Publishers Inc., Sydney, pp 307–314
25. Kolter J, Maloof M (2005) Using additive expert ensembles to cope with concept drift. In: Proceedings of international conference on machine learning (ICML), Bonn, Germany, pp 449–456
26. Kolter JZ, Maloof MA (2007) Dynamic weighted majority: an ensemble method for drifting concepts. J Mach Learn Res 8:2755–2790
27. Kranen P, Assent I, Baldauf C, Seidl T (2010) The clustree: indexing micro-clusters for anytime stream mining. Knowl Inf Syst (In preprints)
28. Kuncheva LI, Sánchez JS (2008) Nearest neighbour classifiers for streaming data with delayed labelling. In: 'ICDM'. pp 869–874
29. Li P, Wu X, Hu X (2010) Learning from concept drifting data streams with unlabeled data. In: 'AAAI'. pp 1945–1946
30. Li X, Yu PS, Liu B, Ng SK (2009) Positive unlabeled learning for data stream classification. In: 'SDM'. pp 257–268
31. 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 if international conference on data mining (ICDM), Pisa, Italy, pp 929–934
32. Masud MM, Gao J, Khan L, Han J, Thuraisingham BM (2009) Integrating novel class detection with classification for concept-drifting data streams. In: ECML PKDD '09, Vol. II. pp. 79–94
33. NASA Aviation Safety Reporting System (n.d.) http://akama.arc.nasa.gov/ASRSDBOnline/QueryWizard_Begin.aspx
34. Scholz M, Klinkenberg R (2005) An ensemble classifier for drifting concepts. In: Proceedings of second international workshop on knowledge discovery in data streams (IWKDDS), Porto, Portugal, pp 53–64
35. Tumer K, Ghosh J (1996) Error correlation and error reduction in ensemble classifiers. Connect Sci 8(304):385–403
36. van Huyssteen GB, Puttkammer MJ, Pilon S, Groenewald HJ (2007) Using machine learning to annotate data for nlp tasks semi-automatically. In: Proceedings of computer-aided language processing (CALP'07)
37. Wagsta K, Cardie C, Schroedl S (2001) Constrained k-means clustering with background knowledge. In: Proceedings of 18th international conference on machine learning (ICML), Morgan Kaufmann, Williamstown, MA, USA, pp 577–584
38. Wang H, Fan W, Yu PS, Han J (2003) Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of ninth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, Washington, DC, pp c226–c235
39. Woolam C, Masud MM, Khan L (2009) Lacking labels in the stream: classifying evolving stream data with few labels. In: Proceedings of international symposium on methodologies for intelligent systems (ISMIS), Prague, Czech Republic, pp 552–562
40. Xing EP, Ng AY, Jordan MI, Russell S (2003) Distance metric learning, with application to clustering with side-information. In: Advances in neural information processing systems vol 15. MIT Press, pp 505–512
41. Yang Y, Wu X, Zhu X (2005) Combining proactive and reactive predictions for data streams. In: Proceedigs of KDD. pp 710–715
42. Zhou A, Cao F, Qian W, Jin C (2008) Tracking clusters in evolving data streams over sliding windows. Knowl Inf Syst 15:181–214
43. Zhou D, Bousquet O, Lal TN, Weston J, Olkopf BS (2004) Learning with local and global consistency. In: Advances in neural information processing systems, vol 16. MIT Press, pp 321–328
44. Zhu X, Ding W, Yu P, Zhang C (2010) One-class learning and concept summarization for data streams. Knowl Inf Syst 1–31
45. Zhu X, Wu X, Yang Y (2006) Effective classification of noisy data streams with attribute-oriented dynamic classifier selection. Knowl Inf Syst 9:339–363
46. Zhu X, Zhang P, Lin X, Shi Y (2007) Active learning from data streams. In: Proceedings of ICDM '07', pp 757–762

Author Biographies

Mohammad M. Masud is a Post Doctoral Fellow at the University of Texas at Dallas (UTD). He received his PhD degree from UTD in December 2009. He graduated from Bangladesh University of Engineering and Technology with MS and BS in Computer Science and Engineering degree in 2004, and 2001, respectively. His research interests are in data stream mining, machine learning, and intrusion detection using data mining. His recent research focuses on developing data mining techniques to classify data streams. He has published more than 25 research papers in journals including IEEE Transactions on Knowledge and Data Engineering (TKDE) and peer reviewed conferences including ICDM, ECML/PKDD, and PAKDD. He is an invited reviewer of a number of journals including IEEE TKDE and Information Systems Frontier (ISF) journal. He also served as a PC member of several conferences and workshops including WWW 2011 conference, SigKDD Workshop 2010 (StreamKDD), and ICDM Workshop 2011 (DDDM). He is a professional member of ACM and IEEE.



Clay Woolam received a Master's degree in Computer Science and a Bachelors degree in Electrical Engineering from the University of Texas at Dallas. He is an engineer at InboxQ in San Francisco.



Jing Gao, www.ews.uiuc.edu/~jinggao3 received the BE and ME degrees, both in Computer Science from Harbin Institute of Technology, China, in 2002 and 2004, respectively. She is currently working toward the PhD degree in the Department of Computer Science, University of Illinois at Urbana Champaign. She is broadly interested in data and information analysis with a focus on data mining and machine learning. In particular, her research interests include ensemble methods, transfer learning, mining data streams, and anomaly detection. She has published more than 20 papers in refereed journals and conferences, including KDD, NIPS, ICDCS, ICDM, and SDM conferences.

Latifur Khan is currently an Associate Professor in the Computer Science department at the University of Texas at Dallas (UTD), where he has taught and conducted research since September 2000. He received his PhD and MS degrees in Computer Science from the University of Southern California, in August of 2000, and December of 1996, respectively. He obtained his BSc degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh in November of 1993. His research work is supported by grants from NASA, the Air Force Office of Scientific Research (AFOSR), National Science Foundation (NSF), IARPA, Raytheon, Alcatel, and the SUN Academic Equipment Grant program. In addition, Dr Khan is the director of the UTD Data Mining/Database Laboratory, which is the primary center of research related to data mining, and image/video annotation at University of Texas-Dallas. Dr Khan's research areas cover data mining, multimedia information management, semantic web, and database systems with the primary focus on first three research disciplines. He has served as a committee member in numerous prestigious conferences, symposiums, and workshops including the ACM SIGKDD Conference. Dr Khan has published over 130 papers in journals and conferences.

Jiawei Han is a Professor of Computer Science at the University of Illinois. He has been researching into data mining, information network analysis, and database systems, with over 500 publications. He is the founding Editor-in-Chief of ACM Transactions on Knowledge Discovery from Data (TKDD) and on the editorial boards of several other journals. Jiawei has received IBM Faculty Awards, HP Innovation Awards, ACM SIGKDD Innovation Award (2004), IEEE Computer Society Technical Achievement Award (2005), and IEEE W. Wallace McDowell Award (2009). He is a Fellow of ACM and IEEE. He is currently the Director of Information Network Academic Research Center (INARC) supported by the Network Science-Collaborative Technology Alliance (NS-CTA) program of U.S. Army Research Lab. His book "Data Mining: Concepts and Techniques" (Morgan Kaufmann) has been used worldwide as a textbook.

Kevin W. Hamlen is an assistant professor in the computer science department at the University of Texas at Dallas. He received his PhD and MS degrees from Cornell University and his BS from Carnegie Mellon. His current work on language-based security investigates the use of stream mining for automated, directed malware mutation. Other research interests include automated mobile code certification, software fault isolation, virtual machines, and cloud computing security. He is the recipient of a 2008 AFOSR Young Investigator Award for research on certifying in-lined reference monitors.

Nikunj C. Oza received his BS in Mathematics with Computer Science from the Massachusetts Institute of Technology (MIT) in 1994. He received his MS (in 1998) and PhD (in 2001) in Computer Science from the University of California at Berkeley. He then joined NASA Ames Research Center and is a member of their Intelligent Data Understanding (IDU) group. He leads of a team applying data mining to the problem of health management for aerospace vehicles. His 40+ research papers represent his research interests that include machine learning (especially ensemble learning and online learning), data mining, fault detection, integration of machine learning with automated planning and other areas of Artificial Intelligence, and their applications to Aeronautics and Earth Science. He received the 2007 Arch T. Colwell Award in 2007 for co-authoring one of the five most innovative technical papers selected out of over 3300 SAE technical papers published in 2005.