Lacking Labels in the Stream: Classifying Evolving Stream Data with Few Labels

Clay Woolam, Mohammad M. Masud, and Latifur Khan

Department of Computer Science, University of Texas at Dallas
{clayw,mehedy,lkhan}@utdallas.edu

Abstract. This paper outlines a data stream classification technique that addresses the problem of insufficient and biased labeled data. It is practical to assume that only a small fraction of instances in the stream are labeled. A more practical assumption would be that the labeled data may not be independently distributed among all training documents. How can we ensure that a good classification model would be built in these scenarios, considering that the data stream also has evolving nature? In our previous work we applied semi-supervised clustering to build classification models using limited amount of labeled training data. However, it assumed that the data to be labeled should be chosen randomly. In our current work, we relax this assumption, and propose a label propagation framework for data streams that can build good classification models even if the data are not labeled randomly. Comparison with state-of-the-art stream classification techniques on synthetic and benchmark real data proves the effectiveness of our approach.

1 Introduction

Data stream classification has gained increasing attention in recent years because large volumes of data are being generated continuously in different domains of knowledge. Data stream classification poses several challenges because of fundamental properties: infinite length and evolving nature. Stream evolution may occur in two ways. First, a new class of data may evolve in the stream that has not been seen before. This phenomenon will be referred to henceforth as *concept-evolution*. Second, the underlying concepts of the data may change. This will be referred to as *concept-drift*. Many solutions have been proposed to classify evolving data streams [1,2,3,4,5,6]. However, most of those techniques assume that as soon as a data point (or a batch of data points) has arrived in the stream and classified by the classifier, that data point (or the batch of data points) would be labeled by an independent labeling mechanism (such as a human expert), and can be used for training immediately. This is an impractical assumption, because in a real streaming environment, it is far beyond the capability of any human expert to label data points at the speed at which they arrive in the stream. Thus, a more realistic assumption would be that only a fraction of the instances would be labeled for training. This assumption was first made by us in our previous work [7].

J. Rauch et al. (Eds.): ISMIS 2009, LNAI 5722, pp. 552–562, 2009.
© Springer-Verlag Berlin Heidelberg 2009

In the previous work, we [7] proposed a technique to train classification models with $P\%$ *randomly* chosen labeled data from each chunk. So, if a training data chunk contained 100 instances, then the algorithm required only P labeled instances. However, the prediction accuracy of the trained model in this technique may vary depending on the quality of the labeled data. That is, this approach should work better on a sample that is uniformly distributed in the feature space rather than a biased, non-uniform sample. In our current work, we propose a more robust technique by making no prior assumption about the uniformity of the labeled instances. Our only requirement is that there should be some labeled instances from each class.

Our ensemble classification technique works as follows. First, we classify the latest (unlabeled) data chunk using the existing ensemble. Second, when $P\%$ of instances in the data chunk have been labeled, we apply constraint-base clustering to create K clusters and split them into homogeneous clusters (micro-clusters) that contain only unlabeled instances, or only labeled instances from a single class. We keep a summary of each micro-cluster (e.g. the centroid, number of data points etc.) as a "pseudo-point" and discard all the raw data points in order to save memory and achieve faster running time. Finally, we apply a label propagation technique on the pseudo-points to label the unlabeled pseudo-points. These labeled pseudo-points act as a classification model. This new model replaces an old model in the ensemble if necessary and the ensemble is kept up-to-date with the current concept. We also periodically refine the existing models to cope with stream evolution.

This paper details several contributions. First, we propose a robust stream classification technique that works well with limited amount of labeled training data. The accuracy of our classification technique is not dependent on the quality of the labeled data. Second, we suggest an efficient label propagation technique for stream data. This involves clustering training instances into pseudo-points and applying label propagation on the pseudo-points. To the best of our knowledge, no label propagation technique exists for data streams. Third, in order to handle concept-evolution and concept-drift, we introduce pseudo-point injection and deletion techniques and analyze their effectiveness both analytically and empirically. Finally, we apply our technique to synthetic and real data streams and achieve better performance than other data stream classification techniques that use limited labeled training data.

The paper is organized as follows: section 2 discusses related works, section 3 presents an overview of the whole process, section 4 describes the training process, section 5 discusses the ensemble technique, section 6 explains the experiments and analyzes the results, and section 7 concludes with directions to future works.

2 Related Work

Our work is closely related to both data stream classification and label propagation techniques. We explore both of these methods below.

Data stream classification techniques can be divided into two major categories: single model and ensemble classification. Single model classification techniques

apply incremental learning so that the models can be updated as soon as new training data arrives [8,9]. The main limitation of these single model techniques is that only the most recent data is used to update the model, and so, the influence of historical data is quickly forgotten. Other single model approaches like [2,6] have been proposed to handle concept-drift efficently.

Ensemble techniques like [3,4,5,10,11] can update their models efficiently and cope with the stream evolution effectively. We also follow an ensemble approach that is different from most other ensemble approaches in two aspects. First, ensemble techniques like [4,5,10] mainly focus on building an efficient ensemble whereby the underlying classification technique, say decision tree, is a blackbox. We concentrate on building an efficient learning paradigm rather than focusing on the ensemble construction. Second, most of the previous classification techniques assume that all instances in the stream will eventually be labeled and can be used for training, but we assume that only a fraction, like 10%, will be labeled and be available for training. In this regard, our approach is related to our previous work [7], which will henceforth be referred as SmSCluster.

Our current approach is different from SmSCluster, the previous approach, in several aspects. First, it was assumed in SmSCluster that the labeled instances would be uniformly distributed, which may not be the case in a real world scenario. We do not make any such assumption. Second, SmSCluster applied only semi-supervised clustering to build the pseudo-points, but did not apply cluster splitting, pseudo-point deletion, or label propagation. Finally, SmSCluster applied K-Nearest Neighbor classification, whereas we apply inductive label propagation for classification.

Algorithm 1. LabelStream

Input: $\mathcal{X}^n$: data points in chunk D_n
K: number of pseudo-points to be created
M: current ensemble of L models $\{M_1, ..., M_L\}$
Output: Updated ensemble M
1. Predict the class labels of each instance in $\mathcal{X}^n$ with M (section 5).
/* Assuming that P% instances in D_n has now been labeled */
2. $M' \leftarrow$ **Train**(D_n) /* Build a new model M' */
3. $M \leftarrow$ **Refine-Ensemble**(M, M') (section 5.1)
4. $M \leftarrow$ **Update-Ensemble**(M, M', D_n) (section 5.2)

Function Train(D_n) Returns Model
2.1. Set of macro-clusters, $\mathcal{MC} \leftarrow$ **Semi-supervised-Clustering**(D_n) (section 4.1)
2.2. Set of micro-clusters, $\mu\mathcal{C} \leftarrow$ **Build-Micro-clusters**$(\mathcal{MC})$ (section 4.1)
2.3. **for each** micro-cluster $\mu\mathcal{C}_i \in \mu\mathcal{C}$ **do** pseudo-point $\psi_i \leftarrow$ Summary$(\mu\mathcal{C}_i)$
2.4. $M' \leftarrow$ Set of all pseudo-points ψ_i
2.5. $M' \leftarrow M' \bigcup_{t=n-r}^{n-1}$ Set of all *labeled* pseudo-points in Chunk D_t
2.6. $M' \leftarrow$ **Propagate-Labels**(M')
2.7. **return** M'

3 Overview of the Approach

Algorithm 1 summarizes the overall process. Line 2 executes the Train operation on an incoming data chunk. Training begins with a semi-supervised clustering technique. The clusters are then split into pure micro-clusters in line 2.2 of the training function. Then, the summary of each micro-cluster is saved as a pseudo-point in line 2.3. In line 2.4 and 2.4, we combine our new set of pseudo-points with the labeled pseudo-points from the last r contiguous chunks. By a labeled pseudo-point we mean the pseudo-points that correspond to only the manually labeled instances. In line 2.6, a modified label propagation technique, [12], is applied on the combined set of pseudo-points. Once we complete training a new model, we return to the main algorithm. In line 3 and 4, the ensemble is refined and updated. The following sections describe this process in detail.

4 Model Generation

The training data is a mixture of labeled and unlabeled data. Training consists of three basic steps. First, semi-supervised clustering is used to build K clusters, denoted as **macro-cluster**s, from the training data. Second, to build homogeneous clusters, denoted as **micro-cluster**s, in order to facilitate label propagation process, and save cluster summaries as **pseudo-point**s. Third, to propagate labels from the labeled pseudo-points to the unlabeled pseudo-points, a transductive label propagation algorithm, from [12], is used. The collection of the labeled pseudo-points are used as a classification model for classifying unlabeled data. The classification and ensemble updating process is described in the next section, section 5.

4.1 Semi-supervised Clustering

With semi-supervised clustering, clusters can be built efficiently in terms of both running time and storage space. The label propagation algorithm takes $O(n^3)$ time in a dataset having n instances. Although this running time is tolerable in a static environment, it may not be practical for a streaming environment where fast training is a critical issue. Training time is reduced by reducing the number of instances to a constant K. This is done by partitioning the instances into K clusters and using the cluster centroids as pseudo-points. This also reduces memory consumption because rather than storing the raw data points, we store the pseudo-points only. Thus, the storage requirement goes from being linear to constant.

The semisupervised clustering objective is to minimize both cluster impurity and intra-cluster dispersion, expressed by

$$\mathcal{O}_{ImpDisp} = \sum_{i=1}^{K}\Big(\sum_{\boldsymbol{x}\in\mathcal{X}_i} ||\boldsymbol{x}-\boldsymbol{u_i}||^2 + \sum_{\boldsymbol{x}\in\mathcal{L}_i} ||\boldsymbol{x}-\boldsymbol{u_i}||^2 * (|\mathcal{L}_i|)^2 * Gini_i * Ent_i\Big) \quad (1)$$

where K is the total number of clusters, μ_i is the centroid of cluster i, $\mathcal{X}_i$ is the set of instances belonging to cluster i, Imp_i is the impurity of cluster i, $\mathcal{L}_i$ is the set of labeled instances in cluster i, and $|\mathcal{L}_i|$ is the corresponding cardinality, $Gini_i$ is the Gini index of cluster $i = \sum_{c=1}^{C}(p_c^i)^2$, C being the total number of classes in the dataset, and Ent_i is the entropy of cluster $i = \sum_{c=1}^{C}(-p_c^i * log(p_c^i))$. This minimization problem, equation 1, is an incomplete-data problem which we solve using the Expectation-Maximization (E-M) technique. Since we follow a similar approach to [7], the details of these steps are omitted here.

Although most of the macro-clusters constructed in the previous step are made as pure as possible, some of them may contain instances from a mixture of classes. A completely pure macro-cluster may also contain some unlabeled instances. So, the macro-clusters are split into micro-clusters so that each micro-cluster contains only unlabeled instances or only labeled instances from a single class. At this point, the reader may ask whether we could create the pure micro-clusters in one step using K-means clustering separately for each class and the unlabeled data in a supervised fashion, rather than creating them in two steps, namely, 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. So, the cluster representatives, or pseudo-points, would have less precision in representing the corresponding data points. As a result, the label-propagation may also perform poorly.

Building micro-clusters is done as follows. Suppose $\mathcal{MC}_i$ is a macro-cluster. In the first case, $\mathcal{MC}_i$ contains only unlabeled instances or only labeled instances from a single class. It is assumed to be a valid micro-cluster and no splitting is necessary. In the second case, $\mathcal{MC}_i$ contains both labeled and unlabeled instances, and/or labeled instances from more than one classes. For each class, we create a micro-cluster with the instances of that class. If $\mathcal{MC}_i$ contains unlabeled instances, then another micro-cluster is created with those unlabeled instances (see figures 1(a) and 1(b)). So, each micro-cluster contains only unlabeled instances, or labeled instances from a single class. If the total number of macro-clusters is K and the total number of classes is C, then the total number of micro-clusters will be at most $C * K = \hat{K}$, which is also 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 unlabeled micro-clusters: The unlabeled micro-clusters may be further split into smaller micro-clusters. This is because, if the instances of an unlabeled micro-cluster actually come from different classes, then this micro-cluster will have a negative effect on the label propagation (see the analysis in section 5.3). 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 unlabeled micro-clusters are split into purer clusters based on the predicted labels of the unlabeled instances (see figure 1).

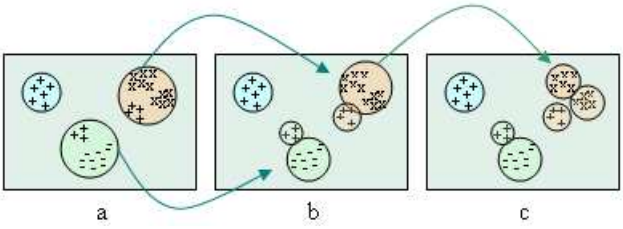

Fig. 1. Illustrating micro-cluster creation. '+' and '-' represent labeled data points and 'x' represents unlabeled data points. (a) Macro-clusters created using the constraint-based clustering. (b) Macro-clusters are split into micro-clusters. (c) Unlabeled micro-clusters are further split based on the predicted labels of the data points.

Creating pseudo-points: The centroid of each micro-cluster is computed and a summary of each micro-cluster is saved as a pseudo-point. This summary contains three fields: i) the *centroid*, ii) the *weight*, i.e., the total number of instances in the micro-cluster, and iii) the assigned class label. After saving the pseudo-point, we discard all the raw instances from the main memory. A pseudo-point will be referred to henceforth with the symbol ψ. The centroid of ψ will be denoted with $\mathcal{C}(\psi)$ and the weight of ψ will be denoted with $\mathcal{W}(\psi)$. We consider a pseudo-point as labeled if all the data points in the micro-cluster corresponding to the pseudo-point are labeled.

5 Ensemble Classification

An unlabeled test data may be classified using the transductive label propagation technique by adding the point to an existing model and rerunning the entire label propagation algorithm. Unfortunately, running the transduction for every test point would be very expensive. The efficient alternative is to use the inductive label propagation technique from [12], $\hat{y} = \frac{\sum_j \boldsymbol{W}_{\Psi}(x,\psi_j)\hat{y_j}}{\sum_j \boldsymbol{W}_{\Psi}(x,\psi_j)+\epsilon}$, x is the test point, ψ_j's are the pseudo-points in the model, $\boldsymbol{W}_{\Psi}$ is the function that generated the matrix $\boldsymbol{W}$ on $\Psi = \{\psi_1, ..., \psi_{\hat{K}}\}$, and ϵ is a small smoothing constant to prevent the denominator from being zero. The complexity of this is linear with respect to the number of pseudo-points in a model, $\hat{K}$.

5.1 Ensemble Refinement

We may occasionally need to refine the exiting models in the ensemble if a new class arrives due to concept-evolution in the stream or old models become outdated due to concept-drift.

Pseudo-point injection: This is required when a new class arrives in the stream. We call a class $\hat{c}$ as a "new" class if no existing model in the ensemble contains any pseudo-point with class label $\hat{c}$, but M', the new model built from

the latest training data, contains some pseudo-points with that class label. Refinement is done by injecting pseudo-points of class $\hat{c}$ into the existing models. When a pseudo-point is injected in a model $M_i \in M$, two existing pseudo-points in M_i are merged to ensure that the total number of pseudo-points remains constant. The closest pair of pseudo-points having the same class labels are chosen for merging.

Pseudo-point deletion: In order to improve the classification accuracy of a classifier M_i, we occasionally remove pseudo-points that may have negative effect on the classification accuracy. For each pseudo-point $\psi \in M_i$, 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. So, if for any labeled instance x, its nearest pseudo-point ψ has a different class label than x, then $A(\psi)$ drops. This statistic help us to determine whether any pseudo-point has been wrongly labeled by the label propagation or if the pseudo-point has become outdated because of concept-drift. In general, we delete ψ if $A(\psi)$ drops below 70%.

5.2 Ensemble Updating

Our ensemble classifier M consists of L classification models $= \{M_1, ..., M_L\}$. The training algorithm described in the previous section (section 4) builds one such model M' from the training data. Each of the L+1 models, M' and L models in the ensemble, are evaluated using the labeled instances of the training data and the best L of them based on accuracy are chosen for the new ensemble. The worst model is discarded. The ensemble is always kept up to date with the most recently trained model. This is an efficient way to handle concept-drift.

5.3 Error Reduction Analysis for Cluster Splitting and Removal

First, we introduce the notion of *swing voter* for a test instance x. Let the pseudo-point ψ_i be the swing voter for x if the label of ψ_i determines the predicted label of x according to the inductive equation. Note that such a voter must exist for any test instance x. Also, $W_{\Psi}(x, \psi_i)$ is the weight from inductive label propagation. Usually, the ψ_i that have the highest $W_{\Psi}(x, \psi_i)$ should be the swing voter for x. In other words, the nearest pseudo-point to x is most likely be its swing voter. Let the probability that ψ_i is a swing voter for a test point x be α_i. Also, let the probability that the class label of the swing voter ψ_i is different from the actual class of x be p_i. For example, if there are 100 test points and ψ_i is the swing voter for 20 test points, then α_i=20/100 = 0.2. We denote an operation called $CL(x)$ to return the class label of a point or pseudopoint. Also, among the 20 test points, if 10 have different class label than ψ_i, then $p_i = 10/20 = 0.5$. It would be clear shortly that α_i is related to deletion and p_i is related to splitting. Therefore, the probability that the next test instance x will not be misclassified because of this pseudo-point = $P(\psi_i$ will not be a swing voter for $x)+$ $P(\psi_i$ will be a swing voter for x and $CL(x) = CL(\psi_i)) = (1-\alpha_i)+\alpha_i(1-p_i) =$ $1 - \alpha_i + \alpha_i - \alpha_i p_i = 1 - \alpha_i p_i$.

$\Rightarrow$ Probability that none of the next N test instances will be misclassified because of this pseudo-point, assuming independence among the test instances = $(1 - \alpha_i p_i)^N$.

$\Rightarrow$ Probability that one or more of the next N test instances will be misclassified because of this pseudo-point (i.e., probability of error), $P(\mathcal{E}_i) = 1 - (1 - \alpha_i p_i)^N$. The expected error of a classifier m is the weighted average of the error probabilities of its pseudo-points, i.e.,

$$E(\mathcal{E}_m) = \frac{\sum_i \alpha_i P(\mathcal{E}_i)}{\sum_i \alpha_i} = \frac{\sum_i \alpha_i (1 - (1 - \alpha_i p_i)^N)}{\sum_i \alpha_i} = \frac{\sum_i \alpha_i}{\sum_i \alpha_i} - \frac{\sum_i \alpha_i (1 - \alpha_i p_i)^N}{\sum_i \alpha_i}$$
$$= 1 - \sum_i \alpha_i (1 - \alpha_i p_i)^N \text{ (since } \sum_i \alpha_i = 1) \quad (2)$$

According to equation 2, the expected error can be minimized if the second term $\sum_i \alpha_i (1 - \alpha_i p_i)^N$ can be maximized. There are two ways to maximize this quantity: making p_i=0 by splitting unlabeled micro-clusters or making $\alpha_i = 0$ by removing pseudo-points from the model.

Splitting unlabeled micro-clusters: If it is assumed that the test instances are identically distributed as the training instances, then we can apply a heuristic to reduce p_i. Recall that p_i is the probability that a test point (for which ψ_i is a swing voter) will have a different class label than the label of the pseudo-point ψ_i. Intuitively, p_i would be zero if all the training instance in the corresponding micro-cluster has the same class label as ψ_i. Although this is ensured for the micro-clusters that have labeled instances, it cannot be ensured for the micro-clusters that have unlabeled instances. Therefore, we use the classifier-predicted labels of the unlabeled instances to determine whether an unlabeled micro-cluster is pure or not. If it is not pure based on the predicted labels, then we split the micro-cluster into purer micro-clusters. Thus, splitting the unlabeled micro-clusters help to keep p_i to a minimum, and reduce the expected error of the corresponding classifier.

Deleting pseudo-points: If for some ψ_i, p_i is too high, then the pseudo-point has a negative effect on the overall classifier accuracy. In this case, we can remove the pseudo-point to improve accuracy, because removal of ψ_i would make α_i=0. Intuitively, p_i = 1-$A(\psi_i)$, where $A(\psi_i)$ is the accuracy of the pseudo-point ψ_i. However, removal helps only if there is no *cascading effect* of the removal on other pseudo-points. A cascading effect occurs if the removal of ψ_i increases p_j of another pseudo-point ψ_j. This is possible if ψ_j becomes the new swing voter for a test instance x, whose original swing voter had been ψ_i, and the class label of x is the same as that of ψ_i, but different from that of ψ_i. To account for this we implemented a simple threshold (i.e. max 50%) deleted.

6 Experiments

Synthetic datasets, **SYN-E** and **SYN-D**, are standard methods for evaluating stream mining methods. These are described in detail in [1]. SYN-E simulates

concept evolution by adding new classes into the stream as time progresses. SYN-D simulates concept drift by changing the slope of a hyperplane over time. The KDDCUP 99 intrusion detection dataset, **KDD**, is also very widely used in stream mining literature, see [1]. It contains 23 different classes, 22 of which are labeled network attacks. The NASA Aviation Safety Reporting System database, **ASRS**, is our second real dataset. The dataset contains around 150,000 text reports, each describing some kind of flight anomaly. See [13] for more details.

6.1 Experimental Setup

Hardware and software: We implement the algorithms in Java. We use a windows-XP based Intel P-IV machine 3GHz processor and 2GB main memory.

Parameter settings: We will refer to our technique as LabelStream. parameter settings of LabelStream are as follows, unless mentioned otherwise: K (number of macro-clusters) = 50; Chunk-size = 1,600 records for real datasets, and 1,000 records for synthetic datasets; L (ensemble size) = 6;

Baseline method: We compare our algorithm with that of Masud et al [7] and Aggarwal et al [1]. We will refer to these approaches as SmSCluster and OnDemandStream, respectively. We run our own implementation of both these baseline techniques. For the SmSCluster, we use the following parameter settings: K (number of micro-clusters) = same as K of LabelStream; Chunk-size = same as the chunk-size of LabelStream; L (ensemble size) = same as the ensemble-size of LabelStream; ρ (injection propability) = 0.75, as suggested by [7]; Q (nearest neighbors in K-NN classification) = 1, as suggested by [7]. For OnDemandStream, we use the following parameter settings: Buffer-size = same as the chunk-size of LabelStream; Stream speed = 80 for real dataset and 200 for synthetic dataset (as suggested by the authors). Other parameters of OnDemandStream are set to the default values.

In the following subsections, we would use the terms "P% labeled" to mean that P% of the instances in a chunk are labeled. So, when we mention that LabelStream is run with 10% labeled data and OnDemandStream is run with 100% labeled data, it means for the same chunk-size (e.g. 1000), LabelStream is trained with a chunk having 100 labeled and 900 unlabeled instances, whereas OnDemandStream is trained with the same chunk having 1000 (all) labeled instances.

6.2 Performance Study

To illustrate the effectiveness of the proposed approach, table 1 shows overall accuracy values for previous approaches SmSCluster and OnDemandStream against LabelStream against all four datasets. There are two methods to decide labeled training instances: *bias* and *random*. Under bias sampling, a point from a class is drawn at random and a labeled set is initialized with that point. Then the nearest neighbor to the labeled set belonging to the same class is added to the labeled set.

Table 1. Performance comparison of SmSCluster and LabelStream at 10% labeled data and OnDemandStream at 100% labeled data

	LabelStream		SmSCluster	OnDemandStream
	Bias	Random		
SYN-E	99.76	98.35	90.28	69.78
SYN-D	84.48	86.40	75.15	73.25
KDD	97.69	98.06	92.57	96.07
ASRS	48.30	41.07	30.33	28.02

Table 2. Testing, training, and manual labeling speeds for the four datasets

	LabelStream		SmSCluster		Manual	OnDemandStream		Manual
	Train	Test	Train	Test	10%	Train	Test	100%
SYN-E	1.1	1.33	1.49	3.0	-	1.15	10.22	-
SYN-D	0.47	0.49	0.66	0.59	-	0.27	7.49	-
KDD	13.2	0.93	7.30	7.39	9600	1.23	20.03	96000
ASRS	60.3	24.5	16.17	41.9	9493.7	613.6	446.8	94936.7

This continues until P% of the points have been drawn for that class. This is done for each class. In random sampling, points are randomly drawn at uniform to be marked as labeled datapoints. The experiment is repeated 20 times and the accuracy value is averaged. SmSCluster and OnDemandStream are run with 10% and 100% labeled data, respectively. LabelStream values are at both 10% randomly drawn data and a special dataset containing biased labeled data. LabelStream performs better than SmSCluster and OnDemandStream in general. For example, table 1 shows under biased and random sampling LabelStream has a 48.9% and 41.07% accuracy, respectively, on the ASRS dataset while SmSCluster has a 30.33% accuracy and OnDemandStream has a 28% accuracy.

LabelStream seems to outperform SmSCluster and OnDemandStream in terms of classification accuracy. Now, we will investigate the difference in processing speed among these algorithms. Table 2 shows a comparison of running times of the three methods across all four datasets. LabelStream and SmSCluster were run with 10% labeled data and OnDemandStream was run with 100% labeled data as in the previous graphs in this section. Results are given in two columns, training and testing times, for each algorithm with the addition of times for the amount of manual annotation needed for each dataset, 60 seconds per instance. These times are just used to illustrate the true gain of LabelStream and SmSCluster over previous approaches as true labeling time is likely much higher. Also, synthetic datasets do not get annotation times because they were machine generated. Times listed are processing times, in seconds, for each data chunk. For example, a chunk containing 1600 data points may take 60.3 seconds to train on LabelStream, 16.17 seconds on SmSCluster, and 613.6 seconds on OnDemandStream. Testing takes 24.5 seconds for LabelStream, 41.9 seconds for SmSCluster, and 446.8 seconds for OnDemandStream. The machine training time is always insignificant compared to the manual annotation time.

References

1. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for on-demand classification of evolving data streams. IEEE Transactions on Knowledge and Data Engineering 18(5), 577–589 (2006)
2. Chen, S., Wang, H., Zhou, S., Yu, P.: Stop chasing trends: Discovering high order models in evolving data. In: Proc. ICDE, pp. 923–932 (2008)
3. Fan, W.: Systematic data selection to mine concept-drifting data streams. In: Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), Seattle, WA, USA, pp. 128–137 (2004)
4. Scholz, M., Klinkenberg., R.: An ensemble classifier for drifting concepts. In: Proc. Second International Workshop on Knowledge Discovery in Data Streams (IWKDDS), Porto, Portugal, October 2005, pp. 53–64 (2005)
5. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: Proc. ninth ACM SIGKDD international conference on Knowledge discovery and data mining, Washington, DC, USA, pp. 226–235. ACM, New York (2003)
6. Yang, Y., Wu, X., Zhu, X.: Combining proactive and reactive predictions for data streams. In: Proc. KDD, pp. 710–715 (2005)
7. Masud, M., Gao, J., Khan, L., Han, J., Thuraisingham, B.: A practical approach to classify evolving data streams: Training with limited amount of labeled data. In: Proc. International Conference on Data Mining (ICDM), Pisa, Italy, December 15-19, pp. 929–934 (2008)
8. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), Boston, MA, USA, pp. 71–80. ACM Press, New York (2000)
9. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: Proc. seventh ACM SIGKDD international conference on Knowledge discovery and data mining (KDD), San Francisco, CA, USA, August 2001, pp. 97–106 (2001)
10. Gao, J., Fan, W., Han, J.: On appropriate assumptions to mine data streams. In: Proc. Seventh IEEE International Conference on Data Mining (ICDM), Omaha, NE, USA, October 2007, pp. 143–152 (2007)
11. Kolter, J., Maloof., M.: Using additive expert ensembles to cope with concept drift. In: Proc. International Conference on Machine Learning (ICML), Bonn, Germany, August 2005, pp. 449–456 (2005)
12. Bengio, Y., Delalleau, O., Le Roux, N.: Label propagation and quadratic criterion. In: Chapelle, O., Schölkopf, B., Zien, A. (eds.) Semi-Supervised Learning, pp. 193–216. MIT Press, Cambridge (2006)
13. Woolam, C., Khan, L.: Multi-label large margin hierarchical perceptron. IJDMMM 1(1), 5–22 (2008)