Semi Supervised Adaptive Framework for Classifying Evolving Data Stream

Ahsanul Haque[1(✉)], Latifur Khan[1], and Michael Baron[2]

[1] Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA
{ahsanul.haque,lkhan}@utdallas.edu
[2] Department of Mathematical Sciences, The University of Texas at Dallas, Richardson, TX, USA
mbaron@utdallas.edu

Abstract. Most of the approaches for classifying evolving data stream divide the stream into fixed size chunks to address infinite length and concept drift problems. These approaches suffer from trade-off between performance and sensitivity. To address this problem, existing adaptive sliding window techniques determine chunk boundaries dynamically by detecting changes in classifier error rate which requires true labels for all of the data instances. However, true labels are scarce and often delayed in reality. In this paper, we propose an approach which determines dynamic chunk boundaries by detecting significant changes in classifier confidence scores using only limited number of labeled data instances. Moreover, we integrate suitable classification technique with it to propose a complete semi supervised framework which uses dynamic chunk boundaries to address concept drift and concept evolution efficiently. Results from the experiments using benchmark data sets show the effectiveness of our proposed framework in terms of handling both concept drift and concept evolution.

Keywords: Dynamic chunk size · Change detection · Concept drift

1 Introduction

Data streams have inherent properties which make it difficult for the traditional data mining techniques to classify stream data. Some of the most challenging properties of data streams include but not limited to infinite length, concept drift, concept evolution, limited labeled data and delayed labeling. Since data stream is an infinite stream of data, it cannot be stored into any storage for analyzing, e.g., labeling. So, data stream classification is essentially a single pass process. Concept drift occurs when the target class or concept evolves within the feature space such that, the class encroaches or crosses previously defined decision boundaries of the classifier [1]. So, any classification method used in the context of data streams need to be updated to cope up with changing concepts. Concept evolution occurs when a new class emerges in the data stream [2,3].

T. Cao et al. (Eds.): PAKDD 2015, Part II, LNAI 9078, pp. 383–394, 2015.
DOI: 10.1007/978-3-319-18032-8_30

To address infinite length and concept drift problems, most of the approaches in the literature divide the data stream into fixed size chunks [2–4]. As a result, these approaches fail to adapt to the change of concepts immediately. If the chunk size is too small, classification method may end up with frequent training during stable period when there is no concept drift, causing performance drawback due to unnecessary update of the classifier. On the contrary, if the chunk size is too large, the classifier may remain outdated for a long period of time. Some other approaches [5,6] use *gradual forgetting* to address infinite length and concept drift problems. These approaches use various decay functions to assign weight to the instances based on their age. This strategy also suffers from similar trade-off while choosing the decay rate to match unknown rate of change.

To solve the problems due to fixed chunk size or decay rate, a dynamic sliding window is maintained in [7,8] by tracking any major change in error rate of the classifier. These approaches mostly assume that, true labels of the data instances will be available to calculate the error rate as soon as it is tested. However, in the real world, labeled data is scarce since labeling data instances manually is costly and time consuming [9]. In data streams, where data arrives very quickly, it may not be possible to label all the data instances as soon as they arrive. So, a good classifier in streaming context should be able to defer the training until true labels become available yet continuing labeling newly arrived instances using the current classifier. Moreover, it should be able to use partially labeled training data [9].

In this paper, we present a complete framework which addresses all of the above challenges. It uses similar semi supervised approach as [2] for classification and novel class detection. However, unlike [2], our proposed framework divides the data stream into dynamically determined chunks using change detection technique. To avoid the use of true labels of data instances for change detection, our proposed framework calculates a confidence value while predicting label of each data instance. It then uses a change point detection technique to detect any significant change in the classifier confidence scores and determines the chunk size dynamically. If a significant change is detected, the classifier is updated using only recent labeled training data instances. To address the concept drift problem, our framework maintains an ensemble of classifier models, each trained on different dynamically determined chunks. Both of classification and confidence value calculation in our framework are semi supervised. So, the proposed framework can work with delayed labeling and partially labeled training data. To the best of our knowledge, our framework is the first semi supervised approach which addresses both of concept drift and concept evolution using dynamically determined chunk boundaries.

The primary contributions of our work are as follows: 1) We present a technique to estimate classifier confidences in predicting labels of stream data instances. 2) We propose a change detection approach which takes classifier confidence values as input and detects if there is any significant change in the classifier confidence. It detects the chunk boundary dynamically if there is a significant change, which triggers updating of the existing classifier on recent labeled training data. Unlike other adaptive sliding window techniques which detect changes

in error rates of the classifier, our approach does not need true labels of all the data instances for determining the chunk boundary dynamically. 3) We present a semi supervised framework by integrating classification and novel class detection technique with the change detection approach to address both of concept drift and concept evolution using dynamic chunk sizes. 4) We implement and evaluate our proposed framework on several benchmark and synthetic data sets. Results from the experiments show that, our framework outperforms other state of the art approaches for data stream classification and novel class detection.

The rest of the paper is organized as follows: In Section 2, we briefly discuss some related works. Section 3 describes our approach in detail. We describe the data sets, evaluation metrics and present experiment results in Section 4. Finally, Section 5 concludes the paper.

2 Related Works

Typically, existing data stream classification approaches address infinite length and concept drift in data stream by dividing it in fixed length chunk sizes [2–4] or using gradual forgetting [5,6]. However, setting a fixed size of the chunks or finding the perfect decay function for gradual forgetting are challenging tasks if the information on time-scale of change is not available [8]. Unlike these approaches, we determine the chunk size dynamically based on a significant change in classifier confidence in predicting labels of test data instances.

There are two types of techniques in terms of change detection, i.e., detecting change in the posterior distribution of the classes given the features $P(y|\mathbf{X})$, another one is detecting change in the generating distribution $P(\mathbf{X})$ [10]. In the literature, several methods[11,12] exist to deal with change of $P(\mathbf{X})$ in multidimensional data. However, detecting change in $P(\mathbf{X})$ is a hard problem especially in case of multi-dimensional data and does not work well to detect change of concept in multi-dimensional multi-class data streaming context [13]. In this paper, we focus on detecting changes in one dimensional classifier confidence values.

Various techniques to detect changes in $P(y|\mathbf{X})$ have been proposed in [7, 8,13], which are mostly based on loss estimation of a predictor performance. These approaches track any significant change in classifier error rate over time which requires the true labels of the data instances. Instead of using the error rate, we calculate classifier confidence in the prediction. Monteith et al. propose a method to estimate classifier confidence in [14], but they use confidence scores only for weighted voting. Unlike this approach, we use confidence scores both for weighted voting and for determining chunk boundaries dynamically. We use two sample *t-test* for one sided right tail hypothesis testing to detect changes in classifier confidence scores. Our proposed framework uses this change detection technique to address both of concept drift and concept evolution problems where [7,8,13] address only the concept drift problem. Unlike most of the above approaches, both of the classification and change detection of our approach are semi supervised in nature.

3 Proposed Approach

As discussed in Section 2, finding the fixed size of chunks or rate of decay is a non trivial task without prior knowledge on time-scale of change [8,10]. Moreover, approaches which use dynamic sliding window using change detection techniques [7,8,13] are based on loss estimation of predictor performance. This estimation needs true labels of the data instances to calculate the predictive performance. However, in the real world data streams, labeled data is scarce and not readily available. The above mentioned approaches might suffer in these scenario.

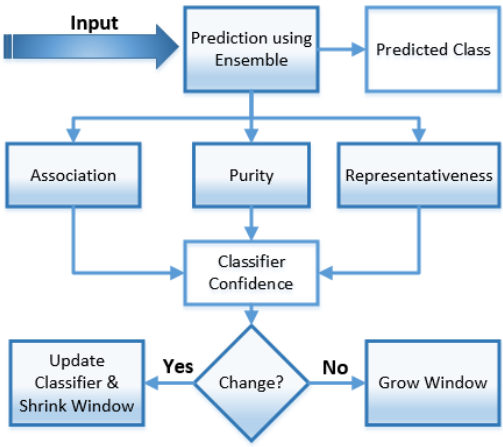

Fig. 1. High level work flow of the framework

In this paper, we present a complete framework *SCDMiner* (Adaptive Semi supervised Concept Drift Miner with novel class detection and delayed labeling) for classifying evolving data streams with novel class detection. It predicts the label of a data instance along with a confidence behind this prediction. Moreover, we also propose a change detection technique which takes these confidence values as input and detects any significant change in the classifier confidence over the time. If a significant change is detected, chunk boundary is determined and the classifier is updated using only the recent labeled data. In this way, *SCDMiner* addresses different challenges of data stream mining discussed in Section 1.

Figure 1 depicts the high level workflow of *SCDMiner*. It maintains an ensemble of L classification models and a dynamic window W containing classifier confidence scores in predicting labels of data instances in the stream. Let $\{M_1, ..., M_L\}$ be the models in the ensemble. As soon as an instance of the data stream arrives, label for this instance is predicted by the current ensemble along with a confidence score which is inserted into W. Subsequently, a change detection technique is executed on W. If it detects a significant change in

the confidence scores, i.e., values stored in W, *SCDMiner* determines the chunk boundary which contains all the instances corresponding to the values stored in W. A new model is trained on the instances which are already labeled in this chunk, and the ensemble is updated by including the newly trained model. On the other hand, if the change detector finds no significant change in the confidence scores, the current ensemble is retained and W keeps growing. Since, *SCDMiner* tracks changes in the confidence values instead of the predictive performance, so it does not need all the true labels immediately after the prediction.

3.1 Classification and Novel Class Detection

SCDMiner uses similar techniques as *ECSMiner* [2] for classification and novel class detection. A k-NN based classifier is trained with the training data. Rather than storing the raw training data, K clusters are built using a semi-supervised K-means clustering, and the cluster summaries (mentioned as *pseudopoint*s) of each cluster are saved. These pseudopoints constitute the classification model. The summary contains the *centroid*, *radius*, and *frequencies* of data points belonging to each class. The radius of a pseudopoint is equal to the distance between the centroid and the farthest data point in the cluster. The raw data points are discarded after creating the summary. Therefore, each model M_i is a collection of K pseudopoints. A test instance x_j is classified using M_i as follows. Let $h \in M_i$ be the pseudopoint whose centroid is nearest from x_j. The predicted class of x_j is the class that has the highest frequency in h. A confidence score between 0 to 1 is calculated based on certain criteria (will be discussed shortly) which is used as the weight of this prediction. The data point x_j is classified using the ensemble M by taking a weighted majority vote among all the classifiers.

Each pseudopoint corresponds to a "hypersphere" in the feature space with a corresponding centroid and radius. The *decision boundary* of a model M_i is therefore the union of the feature spaces encompassed by all pseudopoints $h \in M_i$. The decision boundary of the ensemble M is the union of the decision boundaries of all models $M_i \in M$. If a test instance is outside of the ensemble decision boundary, it is declared as an *F-outlier*, or filtered outlier. These are potential novel class instances, and are temporarily stored in a buffer buf to observe whether they are close to each other (*cohesion*) and farther apart from the data points of other classes (*separation*) [2]. A *new class* is declared if there are sufficient number of *F-outlier*s fulfilling these conditions.

3.2 Calculation of Confidence Scores

We calculate three different confidence estimator values on each of the test instance to calculate confidence of each individual model. Finally, we combine all these individual model confidences to calculate the overall confidence of the ensemble classifier. Assuming h is the closest pseudopoint from labeled data instance x in model M_i, our proposed confidence estimators are as follows:

- *Association* is calculated by $R_h - D_i(x)$, where R_h is the radius of h and $D_i(x)$ is the distance of x from h.

- *Purity* is calculated by N_m/N_s, where N_s is the sum of all the *frequencies* and N_m is the highest *frequency* in h.
- *Representativeness* is calculated by N_s/N_t, where N_t is the number of labeled training instances used to build M_i and N_s is the sum of all the *frequencies* in h.

Association, *Purity* and *Representativeness* of the model M_i for instance x are denoted by $\mathcal{A}_i^x$, $\mathcal{P}_i^x$ and $\mathcal{R}_i^x$ respectively. Each of the confidence estimators contribute to the final confidence in prediction of a model according to their estimation capability. We measure this capability by calculating the correlation coefficient between confidence estimator values and classification accuracy for each model M_i using the labeled training instances as follows. M_i calculates confidence estimator values for each of the labeled training instances. Let h_{ij}^k be the value of j^{th} confidence estimator in M_i's classification of instance k. Since we use three confidence estimators, $j \in \{1, 2, 3\}$. Let $\hat{y}_i^k$ be the prediction of M_i on instance k and y^k be the true label of that instance. Let v_i be the vector containing v_i^k values indicating whether the classification of instance k by model M_i is correct or not. In other words, $v_i^k = 1$ if $\hat{y}_i^k = y^k$ and $v_i^k = 0$ if $\hat{y}_i^k \neq y^k$. Finally, correlation vector r_i is calculated for model M_i. It contains r_{ij} values which are *pearson*'s correlation coefficients between h_{ij} and v_i for different j.

Correlation coefficients calculated in the training phase are used for classification and confidence estimation during testing phase as follows. First, *SCDMiner* calculates confidence estimator values h_i^x for a test instance x. Let c_i^x be the confidence value of model M_i in predicting test instance x. c_i^x is calculated by taking the dot product of h_i^x and v_i, i.e., $c_i^x = h_i^x.v_i$. Similarly, *SCDMiner* calculates confidence value of each of the models in the ensemble along with the prediction for each test instance. Each confidence value is normalized between 0 and 1. Normalized confidence value is treated as the weight of the prediction $\hat{y}_i$ by model M_i. Finally, to estimate confidence of the entire ensemble denoted by c^x, *SCDMiner* takes the average confidence of the models in the ensemble towards the predicted class.

3.3 Change Detection and Updating the Ensemble

As discussed earlier, *SCDMiner* maintains a variable size window W to monitor confidence scores of the ensemble classifier on recent data instances. The expectation is, the size of W will increase during stable period, and will decrease when there is a concept drift. The basic intuition behind this is, concept drift or concept evolution causes change of class boundaries which worsens performance of the classifier if not updated timely [15]. Confidence estimators are chosen in such a way that estimator values are expected to be decreased if class boundaries are changed. For example, if class boundaries are changed due to concept drift, more recent instances will be nearby the decision boundary or outside of the decision boundary of the ensemble classifier. So, in case of these instances, classifier models will have low *association* values. A change detection algorithm is therefore applied on the confidence values stored in W to detect any significant

change in classifier confidence scores. If a change is detected, the base learning algorithm is invoked to build a new model on data instances corresponding to the current window W and subsequently W is shrinked. On the contrary, if no change is detected, W keeps growing indicating a stable period.

Algorithm 1.. Change detection algorithm

1: $W \leftarrow \emptyset$
2: **while** true **do**
3: $x \leftarrow$ the latest data point in the stream
4: $[\hat{y}, c^x] \leftarrow$ Classify(x) // c^x is calculated as discussed in Section 3.2
5: $W \leftarrow W \cup c^x$
6: **for** $n \leftarrow \triangle$ **to** $N - \triangle$ **do**
7: $W_b \leftarrow W[1:n]$
8: $W_a \leftarrow W[n+1:N]$
9: $t_{obs} \leftarrow$ calcObsStat()
10: $pVal \leftarrow \Pr(t \geq t_{obs} \mid H_0)$
11: **if** $pVal \leq \alpha$ **then**
12: RetrainClassifier()
13: ShrinkW()
14: **end if**
15: **end for**
16: **end while**

Algorithm 1 sketches our proposed change detection method. The variable size window W is maintained as follows. After inserting each confidence value of the ensemble classifier, our change detection technique divides W into two sub windows. Let W_b and W_a are two sub windows within W, where W_a contains performance values on more recent data instances than W_b. Change detection algorithm detects any significant change of statistical properties between contents of the sub windows for all possible combinations of *sufficiently large* W_a and W_b (Lines 6 to 15). By mentioning *sufficiently large*, we mean that each of the sub windows must have atleast $\triangle$ number of values. We use one tenth of the size of W as $\triangle$ in our experiments.

We use statistical hypothesis testing to detect change of a statistical property θ between elements of W_b and W_a. In this paper, we use the mean of the population as θ. Let μ_a and μ_b be the mean of population of distribution in W_a and W_b respectively. Let D be the difference between the mean of two sub windows, i.e., $D = \mu_b - \mu_a$. Since we want to detect the case where W_b contains greater average confidence value than W_a, i.e., decreasing classifier confidence, we perform a one sided right tail hypothesis testing. In this hypothesis testing, *Null Hypothesis* is $D \leq \delta$; in other words μ_b is at most δ more than μ_a. On the contrary, *Alternative Hypothesis* is $D > \delta$; in other words μ_b is greater than μ_a and difference between them is more than δ. Here, δ is a small real number and an user defined parameter.

Sample average $\bar{X}$ is the estimator of the mean of the population μ. Let $\bar{X}_a$ and $\bar{X}_b$ be the sample averages of the values in W_a and W_b respectively.

According to Central Limit Theorem, if sample size n is large, $\bar{X}$ follows a normal distribution with expectation μ and variance σ^2/n, where σ^2 is the variance of the population. Since we expect each of the sub windows W_a and W_b to contain sufficiently large number of values and true variance of the distribution of these values are unknown, we perform a two-sample t-test for the hypothesis testing. Let n and m be the number of values in W_b and W_a respectively. Test statistic for our case is the following-

$$t = \frac{\bar{X}_b - \bar{X}_a - \delta}{\sqrt{\frac{{s_b}^2}{n} + \frac{{s_a}^2}{m}}} \tag{1}$$

Where ${s_a}^2$ and ${s_b}^2$ are the sample variances of elements in W_a and W_b respectively. Since the samples in W_a and W_b are not paired, i.e., independent samples, the test statistic in Equation 1 follows a Student's t-distribution with degree of freedom γ, which is the Null Distribution in our case. The value γ is approximated using the following Satterthwaite's Formula. To limit the possibility of false positives, we use a small value α as the level of significance so that $Pr[Reject\ H_0 \mid H_0\ is\ true] \leq \alpha$. H_0 is rejected and a change point is detected if $t_{obs} \geq t_{\gamma,\alpha}$ holds. We build a new model on the data instances corresponding to confidence values stored in W and drop the sub window W_b subsequently. Once a new model is trained, it replaces the oldest model in the ensemble classifier. This ensures that we have exactly L models in the ensemble at any given point of time.

4 Experiment Results

We evaluate our proposed approach *SCDMiner* both on several benchmark real world and synthetic data sets. In this section, we present and analyze the experiment results.

4.1 Data Sets

We use three real and four different types of synthetic data sets to test performance of *SCDMiner* along with some other baseline approaches. Table 1 depicts the characteristics of the data sets.

ForestCover [16] contains geospatial descriptions of different types of forests. We normalize the data set, and arrange the data in order to prepare it for novel class detection so that in any chunk at most three and at least two classes co-occur, and new classes appear randomly. In PAMAP [17], nine persons were equipped with sensors that gathered a total of 52 streaming metrics features whilst they performed activities. Electricity [16] data set contains data collected from the Australian New South Wales Electricity Market.

SynCN (Synthetic Data with Concept-Drift and Novel Class) is a synthetic data set generated using the following equation: $\sum_{i=1}^{d} a_i x_i = a_0$ as explained

Table 1. Characteristics of Data Sets

Name of Data set	Num of Instances	Num of Classes	Num of Features
ForestCover	150,000	7	54
PAMAP	150,000	19	52
Electricity	45,312	2	8
SynCN	100,000	20	40
SynRBF@0.001	100,000	7	70
SynRBF@0.002	100,000	7	70
SynRBF@0.003	100,000	7	70

in [2]. SynRBF@X are synthetic data sets generated using *RandomRBFGeneratorDrift* of MOA [18] framework where X is the Speed of change of centroids in the model. We generate three such data sets using different X to check how efficiently different approaches can adapt to a concept drift.

We use ForestCover, PAMAP and SynCN data sets for simulating both concept drift and novel classes. On the contrary, rest of the data sets are used to test only concept drift capturing ability of different approaches.

4.2 Experiment Setup

We implement *SCDMiner* in Java version 1.7.0.51. To evaluate performance, we use a virtual machine which is configured with 8 cores and 16 *GB* of RAM. The clock speed of each virtual core is 2.4 *GHZ*.

We compare classification and novel class detection performance of our approach *SCDMiner* with *ECSMiner* [2]. We choose *ECSMiner* since it is one of the most robust and efficient frameworks available in the literature for classifying data streams having both concept drift and concept evolution. However, *ECSMiner* uses fixed chunk size where our proposed approach uses variable chunk size based on the change in classifier confidence.

Other than that, we compare performance of *SCDMiner* with *OzaBagAdwin (OBA)* and *Adaptive Hoeffding Tree (AHT)* implemented in MOA [18] framework, since these approaches seem to have superior performance than others on the data sets used in the experiments. Both of *OBA* and *AHT* use *ADWIN* [8] as the change detector. These approaches do not have novel class detection feature. So, we compare these approaches with *SCDMiner* only in terms of classification performance.

We evaluate the above classifiers on a stream by first testing and then training. To evaluate *SCDMiner* and *ECSMiner*, we use 50 pseudopoints, ensemble size 6, and 95% of labeled training data as suggested in [2]. On the contrary, we use 100% labeled training data in case of *OzaBagAdwin (OBA)* and *Adaptive Hoeffding Tree (AHT)*, since training and updating of these approaches are fully supervised.

4.3 Performance Metrics

Let FN = total number of novel class instances misclassified as existing class, FP = total number of existing class instances misclassified as novel class, TP = total number of novel class instances correctly classified as novel class, Fe = total number of existing class instances misclassified (other than FP), N_c = total number of novel class instances in the stream, N = total number of instances the stream. We use the following performance metrics to evaluate our technique:

1. *ERR*: Total misclassification error (percent), i.e., $\frac{(FP+FN+Fe)*100}{N}$.
2. M_{new}: % of novel class instances Misclassified as existing class, i.e., $\frac{FN*100}{N_c}$.
3. F_{new}: % of existing class instances Falsely identified as novel class, i.e., $\frac{FP*100}{N-N_c}$.
4. F_2: F_β score provides the overall performance of a classifier. In this paper, we use $\beta = 2$, which gives us $F_2 = \frac{5*TP}{5*TP+4*FN+FP}$.

Table 2. Summary of classification results

Name of Data set	SCDMiner Error%	ECSMiner Error%	AHT Error%	OBA Error%
ForestCover	**2.62**	4.23	22.89	18.06
PAMAP	**4.96**	35.26	8.76	7.27
Electricity	**0.0**	0.02	27.72	22.26
SynCN	1.20	**0.01**	4.81	4.5
SynRBF@0.001	**9.26**	34.64	18.82	11.31
SynRBF@0.002	**19.72**	63.43	38.75	37.04
SynRBF@0.003	**39.77**	65.39	48.65	46.86

4.4 Classification Performance

As discussed earlier, *SCDMiner* avoids unnecessary training during stable period and frequently updates the classifier when needed using dynamically determined chunks. As an instance, with increasing speed of change of centroids X in SynRBF@X data sets, our change detection technique helps *SCDMiner* to update the ensemble classifier more frequently to cope up with increasing concept drift. *SCDMiner* creates 111 and 160 number of chunks while classifying SynRBF@0.001 and SynRBF@0.003 data sets respectively where *ECSMiner* creates same 69 number of chunks in both of the cases. We do not report the number of chunks for all the data sets due to limited space in this paper.

Table 2 summarizes the classification error of the techniques on each data set described in Section 4.1. In almost all the cases, our proposed approach *SCDMiner* clearly outperforms all the other approaches by large margin in terms of classification accuracy. For example, in case of ForestCover data set, *SCDMiner*

Table 3. Summary of novel class detection results

Data set	Method	M_{new}	F_{new}	F_2
ForestCover	SCDMiner	0.07	2.39	**0.95**
	ECSMiner	8.42	2.13	0.88
PAMAP	SCDMiner	0.09	13.65	**0.98**
	ECSMiner	0.05	37.53	0.45
SynCN	SCDMiner	0.0	0.01	0.99
	ECSMiner	0.0	0.0	**1.0**

shows around 38%, 89% and 85% better performance than *ECSMiner*, *AHT* and *OBA* respectively. Only in case of *SynCN* data set, *ECSMiner* shows slightly better performance than *SCDMiner*. It can be observed that, in case of *SynCN* data set, all the approaches show comparatively better result than the other data sets which indicates that *SynCN* data set contains less frequent concept drift. Since, *ECSMiner* uses fixed chunk size, it updates the model more frequently during stable time period comparing with *SCDMiner*. From the experiment, we know that *SCDMiner* updates the ensemble classifier only 7 times comparing with 44 number of updates by *ECSMiner*. So, *ECSMiner* gains slightly better accuracy in expense of more frequent training and updating the ensemble.

4.5 Novel Class Detection

Table 3 summarizes novel class detection performance of *SCDMiner* and *ECSMiner* on different data sets. From the experiment data, it is clear that *SCDMiner* outperforms *ECSMiner* by a large margin based on F_2 measure on all the data sets except *SynCN*. For example, in case of *ForestCover* data set, *SCDMiner* shows 8% better performance than *ECSMiner* in terms of F_2 measure. In case of *SynCN* data set, *SCDMiner* shows competitive performance. *ECSMiner* gains slightly better performance due to more frequent updates as discussed above.

5 Conclusion

In this paper, we present a framework *SCDMiner* which addresses most of the challenges of classifying evolving data streams. It exploits a change detection technique to determine the chunk boundaries dynamically. As a result, *SCDMiner* determines number of training based on the frequency and intensity of concept drift. Results from the experiments show that, *SCDMiner* outperforms other approaches in the stream mining domain in terms of both classification and novel class detection accuracy.

Acknowledgments. This material is based upon work supported by NSF award no. CNS-1229652 and DMS-1322353.

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. Masud, M.M., Gao, J., Khan, L., Han, J., Thuraisingham, B.M.: Classification and novel class detection in concept-drifting data streams under time constraints. IEEE Trans. Knowl. Data Eng. **23**(6), 859–874 (2011)
3. Parker, B., Khan, L.: Detecting and tracking concept class drift and emergence in non-stationary fast data streams. In: Twenty-Ninth AAAI Conference on Artificial Intelligence, January 2015
4. Aggarwal, C.C., Yu, P.S.: On classification of high-cardinality data streams. In: SDM, pp. 802–813. SIAM (2010)
5. Koychev, I.: Tracking changing user interests through prior-learning of context. In: De Bra, P., Brusilovsky, P., Conejo, R. (eds.) AH 2002. LNCS, vol. 2347, pp. 223–232. Springer, Heidelberg (2002)
6. Klinkenberg, R.: Learning drifting concepts: Example selection vs. example weighting. Intell. Data Anal. **8**(3), 281–300 (2004)
7. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. In: Bazzan, A.L.C., Labidi, S. (eds.) SBIA 2004. LNCS (LNAI), vol. 3171, pp. 286–295. Springer, Heidelberg (2004)
8. Bifet, A., Gavald, R.: Learning from time-changing data with adaptive windowing. In: SDM. SIAM (2007)
9. Masud, M.M., Gao, J., Khan, L., Han, J., Thuraisingham, B.M.: A practical approach to classify evolving data streams: Training with limited amount of labeled data. In: ICDM, pp. 929–934 (2008)
10. Gama, J.A., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Comput. Surv. **46**(4), 44:1–44:37 (2014)
11. Song, X., Wu, M., Jermaine, C., Ranka, S.: Statistical change detection for multidimensional data. In: 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 667–676, New York. ACM (2007)
12. Kuncheva, L.I., Faithfull, W.J.: PCA feature extraction for change detection in multidimensional unlabelled data. IEEE Transactions on Neural Networks and Learning Systems (2013)
13. Harel, M., Mannor, S., El-yaniv, R., Crammer, K.: Concept drift detection through resampling. In: Proceedings of the 31st International Conference on Machine Learning (ICML 2014), JMLR Workshop and Conference Proceedings, pp. 1009–1017 (2014)
14. Monteith, K., Martinez, T.: Using multiple measures to predict confidence in instance classification. In: The 2010 International Joint Conference on Neural Networks (IJCNN), pp. 1–8, July 2010
15. Vapnik, V.N.: Statistical Learning Theory. Wiley-Interscience (1998)
16. MOA: Moa massive online analysis-real time analytics for data streams repository data sets (2015). http://moa.cms.waikato.ac.nz/datasets/
17. Reiss, A., Stricker, D.: Introducing a new benchmarked dataset for activity monitoring. In: ISWC, pp. 108–109. IEEE (2012)
18. Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T., Seidl, T.: Moa: massive online analysis, a framework for stream classification and clustering. Journal of Machine Learning Research, 44–50 (2010)