International Journal of Information Technology & Decision Making
Vol. 5, No. 4 (2006) 659–670

World Scientific
www.worldscientific.com

DETECTION AND CLASSIFICATION OF CHANGES IN EVOLVING DATA STREAMS

MOHAMED MEDHAT GABER
School of Information Technologies, University of Sydney
NSW 2006, Australia
mgaber@cs.usyd.edu.au

PHILIP S. YU
IBM Thomas J. Watson Research Center
19, Skyline Drive, Hawthorne, NY 10532
psyu@us.ibm.com

Data stream mining has attracted considerable attention over the past few years owing to the significance of its applications. Streaming data is often evolving over time. Capturing changes could be used for detecting an event or a phenomenon in various applications. Weather conditions, economical changes, astronomical, and scientific phenomena are among a wide range of applications. Because of the high volume and speed of data streams, it is computationally hard to capture these changes from raw data in real-time. In this paper, we propose a novel algorithm that we term as STREAM-DETECT to capture these changes in data stream distribution and/or domain using clustering result deviation. STREAM-DETECT is followed by a process of offline classification CHANGE-CLASS. This classification is concerned with the association of the history of change characteristics with the observed event or phenomenon. Experimental results show the efficiency of the proposed framework in both detecting the changes and classification accuracy.

Keywords: Data streams; change detection; classification and clustering.

1. Introduction

A data stream from the processing point of view is an infinite flow of highly rapid generated records that challenge our computing systems to store, process and transmit.[1] Examples of data streams include: Web clickstreams, sensor data, ATM transactions, stock market data, phone calls, computer network traffic, and astronomical/scientific reading devices.[2]

Querying and mining data streams have gained considerable attention over the past few years.[3] Different strategies in conventional querying and data mining techniques have been adapted to be able to cope with the continuous high-speed nature of data streams. Although traditional data mining algorithms have mainly focused on clustering, classification and frequent pattern analysis techniques,

data stream mining has added change detection as one of the data analysis strategies. Detecting changes in data streams is considered as an essential data mining process due to the evolving nature of streaming information in a wide range of applications.

A wide range of data streams follows a stable data distribution within a domain in the normal situation. A change in the distribution and/or domain represents an event or a phenomenon that has already occurred or will occur. For example: IP addresses passing though a router follow a data distribution within the domain of IP addresses that often access this router. The change in the data distribution or the domain can represent a possible attack. Other examples in environmental monitoring, quality control and theoretical significance of change detection in data stream mining have been discussed in Ref. 4. It is worth mentioning that data streams are generated in high volumes that challenge the detection of such changes.

In this paper, we propose a novel approach termed as STREAM-DETECT to identify changes in data streams. The proposed algorithm is concerned with detecting changes in data streams by measuring online clustering result deviation over time.

Previous work in change detection in data streams has used different statistical techniques, however, these methods have not addressed the change in both domain and data distribution. Furthermore, the use of online clustering to detect changes can provide us with a range of information about the encountered change. This information includes the degree of change in domain or distribution (the change could be partial that represents a small change or total that represents a big change), and ability to store the change for further offline analysis and classification. This gives our approach its strength over the previous attempts in change detection. STREAM-DETECT is followed by an offline classification CHANGE-CLASS process that associates the change pattern with an observed event or phenomenon. Thus our approach is a combination of online clustering and offline classification of streaming data. Experimental results have shown the efficiency of the proposed approach.

One of the major potential applications of our proposed framework is in monitoring sensor networks. Sensor networks generate huge amounts of data streams continuously and in a very high data rates. Analysis of data generated from sensor networks has its potential in scientific discovery acceleration and various security applications. Sensing devices are featured by being resource-constrained ones.[5] A local data analysis process onboard a sensor is required due to battery consumption problem while transmitting large amounts of data to a central server.[6] The lightweight feature of our proposed approach for change detection makes our system a strong candidate for sensor network applications. Once a change has been detected onboard a sensor using our CHANGE-DETECT technique, a classification of this change could be done offline at a control station to take an action regarding the event or phenomenon detected.

The rest of paper is organized as follows. Section 2 reviews related work in the area of stream change and concept drift. Our STREAM-DETECT algorithm is presented and discussed in Sec. 3. Section 4 shows the classification process using our CHANGE-CLASS algorithm. Tuning the algorithm parameters for best performance is discussed in Sec. 5. Experimental results are presented in Sec. 6 to show the efficiency and robustness of the proposed framework. Finally, we conclude the paper and present possible future research directions in Sec. 7.

2. Related Work

Data stream clustering, classification, frequent pattern and change detection mining algorithms have been studied thoroughly in the literature in the last few years.[1,3] The change detection represents a relatively new category of mining strategies that has emerged due to the evolving nature of streaming information. Different algorithms have been proposed to detect such changes as follows.

Aggarwal[7,8] has proposed the use of differential kernel density estimation over time windows to detect the rate of change in data densities. Changing the window size provides the user by the ability to detect both long- and short-term changes. Graphical representations of these changes have been also studied in this work. Ben-David *et al.*[4] have studied distribution change detection in data streams using statistical tests that are sensitive to distribution changes. The techniques used are non-parametric and can distinguish between statistically significant change and noise.

Other related work includes detecting changes in data mining models. Nasraoui *et al.*[9] have proposed an algorithm to detect clustering results in noisy data streams. Wang *et al.* have proposed a technique to detect changes between old and new classification models using correspondence tracing of the generated rules from the old model with the new one.

Concept drifting based mining techniques have also been studied to give the capability to make the model representative to the current condition of data rather than computing a model that represents both old and new data which can be a misleading model due to the concept drift. Algorithms under this category have been studied in Refs. 10–15.

Having briefly presented the related work, the following section proposes our STREAM-DETECT algorithm for revealing the change in data stream distribution and/or domain using online clustering result deviation, followed by the classification approach.

3. STREAM-DETECT Algorithm

The algorithm starts with an online clustering algorithm that has only one pass over the stream. It uses a distance threshold technique for assigning new points to existing clusters. This process is terminated after a time frame. Measurements

about the characteristics of the clustering results are saved. This process is followed by another run of the clustering algorithm and these measurements are also saved. The deviation between the old and new measurements is calculated. This deviation is stored in one of two cases: The deviation has exceeded a pre-specified threshold, or an event or phenomenon has been encountered. If the deviation has not met any of these criteria, only data characteristics are stored. In case of exceeding a pre-specified threshold, the sequence of change could be analyzed for the first encountered event. In case, the event has occurred, the recent calculated change is stored. This indicates when the event occurs. This iterative process of clustering results and deviation calculations continues over time.

There are two categories of structures in this algorithm: Clustering characteristics and change deviation. Clustering characteristics include:

- mean of cluster centers,
- standard deviation of cluster centers,
- mean size of clusters, and
- maximum and minimum cluster centers.

Clustering deviation calculates the deviation among two cluster characteristics using the absolute value of the difference between each two consecutive runs normalized by the older one. However for the domain detection, we calculate it by discovering the change of the maximum and minimum centers normalized by the distance between the old maximum and minimum centers. Figure 1 depicts STREAM-DETECT algorithm.

The calculation of result deviation is done in a way to detect the change in both data distribution and/or domain. The change in data distribution is detected through the change in cluster means, standard deviation and/or average cluster size. The domain change is detected through the change in the maximum and minimum values of attributes for cluster centers. The higher the threshold value, the higher the change is detected and vice versa given that choosing a low threshold value would result in small changes that may not represent any real change.

```
STREAM-DETECT Algorithm
Call online clustering
Measure clustering characteristics
Do
       Call online clustering
       Measure clustering characteristics
       Measure clustering deviation
       If deviation > threshold deviation
          Or an event has occurred
              Store clustering deviation
Until (END-OF-STREAM)
```

Fig. 1. STREAM-DETECT algorithm.

Table 1. Change detect notation.

Symbol	Meaning
Center_{it}	Vector of cluster center i at time t for $(A_{it}^1, A_{it}^2, \ldots, A_{it}^k)$ where k is the number of attributes
Domchange	The relative change in the data stream domain with regard to the last domain
Meanchange	The relative change in the mean value of the stream with regard to the last mean value
Cstdchange	The relative standard deviation change in the mean of cluster centers with regard to the last standard deviation
Meansizechange	The relative change in the average number of points in each cluster with regard to the last average
domax_{xt}	The maximum value of attribute x for all the cluster centers at time t
domin_{xt}	The minimum value of attribute x for all the cluster centers at time t
meancc_{xt}	The mean of cluster centers at time t for the attribute x
cstd_{xt}	The standard deviation of cluster centers at time t for the attribute x
meansize_t	The mean number of records in each cluster at time t

Using the symbols shown in Table 1, the following equations show the calculation of the change in clustering features in order to determine if a change has occurred.

$$\text{Domchange} = \frac{\sum_{x=1}^{k} |\,\text{domax}_{xt} - \text{domax}_{xt-1}| + \sum_{x=1}^{k} |\,\text{domin}_{xt} - \text{domin}_{xt-1}|}{\sum_{x=1}^{k} |\text{domax}_{xt-1} - \text{domin}_{xt-1}|}, \tag{1}$$

$$\text{Meanchange} = \sum_{x=1}^{k} \frac{|\text{meancc}_{xt} - \text{meancc}_{xt-1}|}{\text{meancc}_{xt-1}}, \tag{2}$$

$$\text{Cstdchange} = \sum_{x=1}^{k} \frac{|\text{cstd}_{xt} - \text{cstd}_{xt-1}|}{\text{cstd}_{xt-1}}, \tag{3}$$

$$\text{Meansizechange} = \frac{|\text{meansize}_t - \text{meansize}_{t-1}|}{\text{meansize}_{t-1}}. \tag{4}$$

The change detection algorithm is followed by a classification process that can associate the change or a sequence of these changes to an event or phenomenon. The following section is devoted for description of this classification process using CHANGE-CLASS algorithm.

4. CHANGE-CLASS Algorithm

The algorithm uses the data produced from STREAM-DETECT to run a voting-based classification algorithm over the change attributes (as shown in Fig. 2). The data stored contains change measurements and the associated event that is used as the class label. CHANGE-CLASS technique is used to classify any detected changes discovered by STREAM-DETECT. This is done through the voting of

```
CHANGE-CLASS Algorithm
Detect-Change
Vote over the change attributes
Find the highest vote
Assign the result as the detected event
```

Fig. 2. CHANGE-CLASS algorithm.

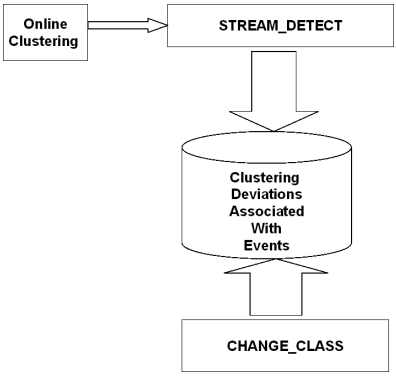

Fig. 3. Clustering and classification of changes.

the nearest neighbor of each change attribute. The classification result will be the event that got the highest vote. That is the event that has attracted the majority of change attributes.

The model of using STREAM-DETECT and CHANGE-CLASS in the problem of identifying and classifying changes in data streams is depicted in Fig. 3. The online training is done using STREAM-DETECT that associates each change with its correspondent event. All the data are stored for a process of offline classification using CHANGE-CLASS technique.

5. Tuning System Parameters

Setting the parameters is an essential pre-processing step for the success of running the proposed framework. Time frame duration (TF) and threshold value (Th) should be set according to the application requirements. Since these requirements are known in advance, the system should be able to tune its parameters in order to get the required output. Setting the time frame basically refers to the time of the change to be occurred. In some applications, the change is too rapid and in this case, the time frame duration should be set as short as possible. In some other applications, the change occurs smoothly and in this case the time frame should be long enough to catch the change. On the other hand, setting the threshold represents how big the change is.

In order to set the parameters, we propose a pre-processing heuristic technique to fine tune the time frame duration and the threshold value. The technique starts

```
Parameter-Tune Algorithm
Repeat
       Set Initial Values for TF, Th
       Call Change-Detect
       Measure the performance
       Set TF = TF + Δf
       Or Set Th = Th + Δt
Until Performance degrading occurred
```

Fig. 4. Parameter-tune algorithm.

with a time frame duration value of a multiple of the data rate and increases it by a user define stepping parameter (Δ_f) till the detection performance gets to the highest possible value. Similarly the tuning of the threshold value is done by choosing a threshold value that can detect small changes and increases the threshold by (Δ_t) to the level of change that needs to be detected.

Figure 4 shows the procedure for adjusting the parameter according to the application needs. The number of steps for setting the parameters varies from the time frame duration to the threshold value. Thus, the tuning is done separately for each of these two parameters.

6. Experimental Results

The proposed framework has been implemented using Matlab 7.0.4.365. We run the experiments on a workstation with Pentium 4 with 3.00 GHz CPU and 504 MB of RAM. The data used in the experiments have been generated from both uniform and normal distributions with changing domain and/or distribution parameters to represent a change in the streaming data. We have generated three different categories of datasets. We change from one category to another to simulate the change in the streaming data. Since we have three categories, six different changes (events or class labels) could be generated as shown in Fig. 5.

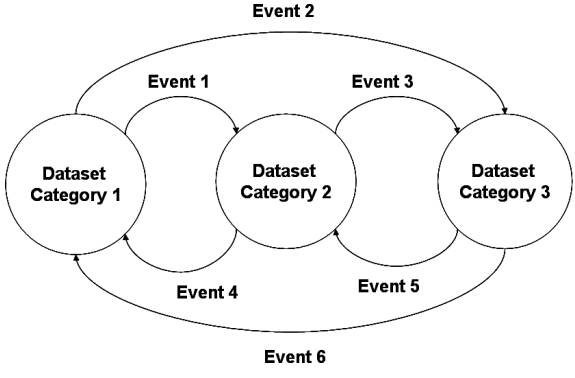

Fig. 5. Events generation using changes in datasets.

The time frame used in the experiment was one second. That means the generated clustering model is re-created every one second. It is important to point out that choosing the time frame depends on two factors that are application dependent:

- The time needed for the change to occur.
- The time for clustering stability.

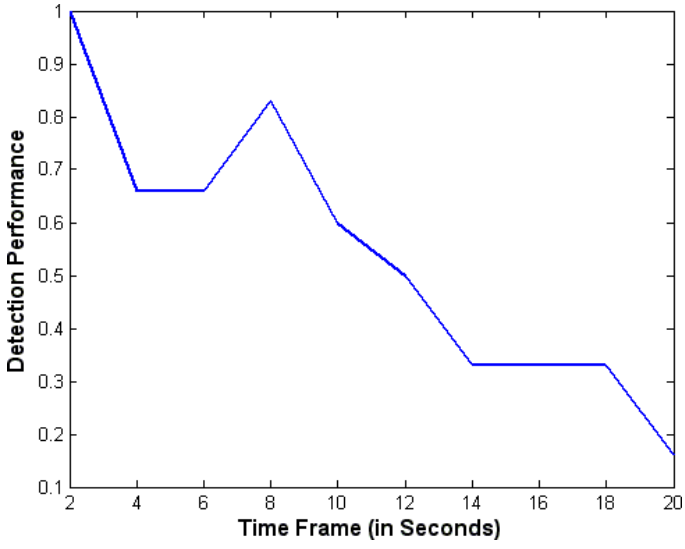

Fig. 6. Time frame duration effect on the detection performance.

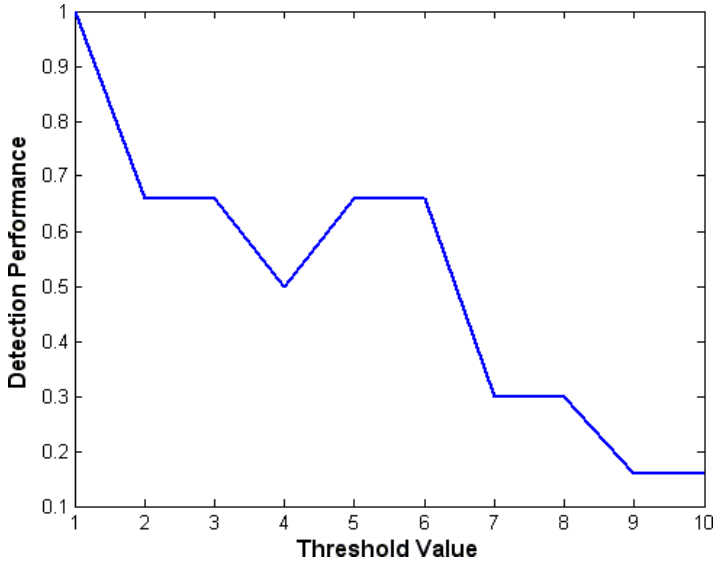

Fig. 7. Threshold value effect on the detection performance.

Once we generated the data using the different distributions and/or domains, the datasets are merged to represent the whole stream of changing data. LWC[16] runs every time frame and the clustering features are reported. The deviation of clustering results is measured. If this change exceeds the threshold determined by the user as one of the application settings, a change is reported and the associated features of the change are stored along with the corresponding event as shown in Fig. 4. This represents the model construction phase of our framework.

The results of detecting the changes in the streams have reached 100% accuracy in most of the runs. Once the right time frame and threshed value have been set, the accuracy obtained from the experiments is very outstanding. Figure 6 shows the effect of the time frame duration on the accuracy of the CHANGE_DETECTION technique. Similarly, Fig. 7 shows how the threshold value can affect the detection performance.

It is important to point out several points for the efficiency of using the model:

- Detecting small changes requires low threshold value.
- Detecting dramatic changes requires high threshold value.
- Detecting frequently occurring changes requires short time frame.
- Detecting less frequent events requires long time frames.

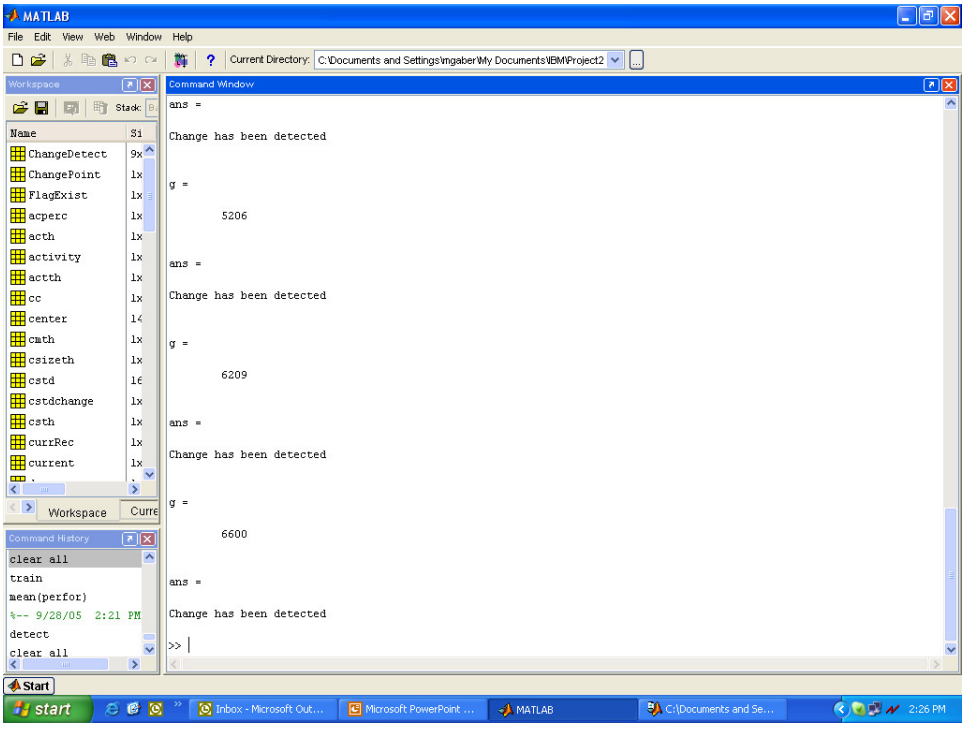

Fig. 8. A screenshot of the running CHANGE-DETECT algorithm.

Table 2. Classification robustness.

Change Factors	Classification Performance (%)
Domain Change, Centroid Change, Standard Deviation Change, and Cluster Size Change	**64.75**
Centroid Change, Standard Deviation Change, and Cluster Size Change	**65.98**
Domain Change, Standard Deviation Change, and Cluster Size Change	60.29
Domain Change, Centroid Change, and Cluster Size Change	56.96
Domain Change, Centroid Change, and Standard Deviation Change	60.17
Standard Deviation Change and Cluster Size Change	57.80
Centroid Change and Cluster Size Change	60.05
Centroid Change and Standard Deviation Change	55.59
Cluster Size Change and Domain Change	47.08
Domain Change and Standard Deviation Change	53.46
Domain Change and Centroid Change	63.63
Domain Change	49.73
Centroid Change	46.00
Standard Deviation Change	**66.53**
Cluster Size Change	41.80

- Detecting changes in unstable datasets requires high threshold value.
- Detecting changes in stable datasets requires low threshold value.

The CHANGE-DETECT technique runs 50 times. Figure 8 shows a screenshot of the running algorithm. The generated results are stored in a matrix. Each row is a vector that stores the four features of changes along with the associated event. This represents the model usage phase. Similar datasets are generated from the same categories used in the model construction phase and then the classifier attempts to predict the event. This process has been repeated 50 times for each different combination of change features. This is done to seek the best possible combination of features that can classify the events. Table 2 shows the results. Each classification performance represents the average of the 50 different runs. It has to be pointed out that in some cases we have reported 100% accuracy for the highlighted combinations in the table.

From Table 2, we can observe that the standard deviation represents the best performance. However, the standard deviation has been used in both cases for detecting the changes as well as the classification process. In the phase of change detection, the performance was considerably low. The best performance reported for both phases of the model was for the combination of all the different features with outstanding performance in detecting the changes and an average of 64.75% in classification accuracy. It has to be noted that the classification accuracy for six

different classes is in average 64.75% that represents an accurate classifier for a wide range of applications that can benefit from using our proposed framework.

7. Conclusion

We have proposed STREAM-DETECT algorithm for identifying change in data stream distribution and/or domain values using a special clustering algorithm. The technique has its potential due to the importance of the applications that require online change detection that can be a representative of an important event or phenomenon in scientific, astronomical, and commercial applications. STREAM-DETECT is followed by a voting-based classification technique termed as CHANGE-CLASS that associates the current change with a previously encountered event. Experimental results are promising for real-life applications to use this lightweight technique especially in wireless sensor networks.

References

1. S. Muthukrishnan, Data streams: Algorithms and applications, in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (2003).
2. H. Wang, J. Pei and P. S. Yu, Online mining of data Streams: Problems, applications and progress (tutorial), in *21st Int. Conf. Data Engineering* (*ICDE*) (Tokyo, Japan, April 2005).
3. M. M. Gaber, A. Zaslavsky and S. Krishnaswamy, Mining data streams: A review, *ACM SIGMOD Record*, Vol. 34, No. 1 (June 2005), ISSN: 0163–5808.
4. S. Ben-David, J. Gehrke and D. Kifer, Detecting change in data streams, in *Proceedings of VLDB*, Toronto, Canada (2004).
5. M. Tubaishat and S. Madria, Sensor networks: An Overview, *IEEE Potentials* **22**(2), (2003) 20–23.
6. R. Bhargava, H. Kargupta and M. Powers, Energy consumption in data analysis for on-board and distributed applications, in *Proceedings of the ICML'03 Workshop on Machine Learning Technologies for Autonomous Space Applications*, Washington, DC, USA (2003).
7. C. Aggarwal, An intuitive framework for understanding changes in evolving data streams, in *Proceedings of the ICDE Conference*, San Jose, CA, USA (2002).
8. C. Aggarwal, A framework for diagnosing changes in evolving data streams, in *Proceedings of the ACM SIGMOD Conference*, San Diego, CA, USA (2003).
9. O. Nasraoui, C. Cardona, C. Rojas and F. González, TECNO-STREAMS: Tracking evolving clusters in noisy data streams with a scalable immune system learning model, in *Proc. Third IEEE Int. Conf. Data Mining* (ICDM'03) (Melbourne, FL, November 2003), pp. 235–242.
10. W. Fan, *StreamMiner: A Classifier Ensemble-based Engine to Mine Concept Drifting Data Streams*, VLDB'2004, Toronto, Canada.
11. W. Fan, Systematic data selection to mine concept-drifting data streams, *KDD* (2004), pp. 128–137.
12. W. Fan, Y. Huang and P. S. Yu, Decision tree evolution using limited number of labeled data items from drifting data streams, *ICDM* (2004), pp. 379–382.
13. G. Hulten, L. Spencer and P. Domingos, Mining time-changing data streams, in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, USA (2001).

14. R. Klinkenberg, Learning drifting concepts: Example selection vs. example weighting. In Intelligent Data Analysis (IDA), *Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift* **8**(3) (2004) 281–300.
15. H. Wang, W. Fan, P. Yu and J. Han, Mining Concept-Drifting Data Streams using Ensemble Classifiers, in *9th ACM Int. Conf. Knowledge Discovery and Data Mining (SIGKDD)* (Washington DC, USA, 2003).
16. M. M. Gaber, S. Krishnaswamy and A. Zaslavsky, On-board mining of data streams in sensor networks, in *Advanced Methods of Knowledge Discovery from Complex Data*, eds. S. Badhyopadhyay, U. Maulik, L. Holder and D. Cook (Springer Verlag, 2005).