Cost-Efficient Mining Techniques for Data Streams

Mohamed Medhat Gaber, Shonali Krishnaswamy and Arkady Zaslavsky
School of Computer Science and Software Engineering, Monash University,
900 Dandenong Rd, Caulfield East, VIC 3145, Australia

{Mohamed.Medhat.Gaber, Shonali.Krishnaswamy, Arkady.Zaslavsky}@infotech.monash.edu.au

Abstract

A data stream is a continuous and high-speed flow of data items. High speed refers to the phenomenon that the data rate is high relative to the computational power. The increasing focus of applications that generate and receive data streams stimulates the need for online data stream analysis tools. Mining data streams is a real time process of extracting interesting patterns from high-speed data streams. Mining data streams raises new problems for the data mining community in terms of how to mine continuous high-speed data items that you can only have one look at. In this paper, we propose algorithm output granularity as a solution for mining data streams. Algorithm output granularity is the amount of mining results that fits in main memory before any incremental integration. We show the application of the proposed strategy to build efficient clustering, frequent items and classification techniques. The empirical results for our clustering algorithm are presented and discussed which demonstrate acceptable accuracy coupled with efficiency in running time.

Keywords: Mining data streams, clustering, frequent items, classification and algorithm output granularity.

1 Introduction

A data stream is a sequence of unbounded, real time data items with a very high data rate that can only read once by an application. (Muthukrishnan 2003, Henzinger, Raghavan, and Rajagopalan 1998, Golab and Ozsu 2003, Babcock, Babu, Datar, Motwani, and Widom 2002). For example in NASA Earth Observation System (EOS), a pair of Landsat 7 and Terra spacecraft generates 350 GB of data per day as mentioned by Park and Kargupta (2002). Another example by Muthukrishnan (2003), an oil drill can transmit its current drilling conditions at 1 Mb/Second.
Two recent advancements result in the need for data stream processing systems as discussed in (Muthukrishnan 2003, Golab and Ozsu 2003):

- The automatic generation of a highly detailed, high data rate sequence of data items in different scientific and business applications.
- The need for complex analyses of these high-speed data streams.

The main constraints for querying data streams are the unbounded memory requirement and the high data rate. Thus, the computation time per data element should be less than the data rate. Also, it is very hard due to unbounded memory requirements to have an exact result. Research studies have been done to approximate the query result. Sliding window, batch processing, sampling, and synopsis data structures have been discussed by (Babcock, Babu, Datar, Motwani, and Widom 2002, Garofalakis, Gehrke, Rastogi 2002) for query result approximation.

Recently, load shedding and rate based query optimization have been proposed by (Babcock, Datar, and Motwani 2003, Tatbul, Cetintemel, Zdonik, Cherniack and Stonebraker 2003) as an approach for querying data streams. Load shedding refers to the process of dropping data elements from data stream randomly or semantically.

Analogous to load shedding for query processing, we propose data rate adaptation as a solution approach for mining data streams. Data rate adaptation could be used from the input side using sampling, filtering and aggregation. We propose the use of data rate adaptation from the output side using algorithm output granularity. Algorithm output granularity is the amount of mining results that fits in main memory before any incremental integration. The application of the proposed technique for different data mining techniques is presented in this paper accompanied with empirical studies for the clustering technique.

The paper is organized as follows. Section 2 presents related works in mining data streams. Section 3 reviews the problem of the high-data rate feature, discusses data rate adaptation techniques and introduces algorithm output granularity as an approach for data stream mining. The application of the algorithm output granularity for clustering, classification and frequent items is presented in terms of new techniques for these data mining strategies in Section 4. Section 5 shows the experimental results of the clustering algorithm. Finally we conclude the paper and present our future work in Section 6.

2 Related Works

Clustering data streams has been studied in (Guha, Mishra, Motwani, and O'Callaghan 2000, Domingos and Hulten 2001, O'Callaghan, Mishra, Meyerson, Guha , and Motwani 2002, Aggarwal, Han, Wang, Yu 2003, Ordonez 2003, Keogh, Lin, and Truppel 2003, Babcock, Datar, and Motwani 2003, Charikar, O'Callaghan, and Panigrahy 2003, Datar, Gionis, Indyk, and Motwani 2002). Data stream classification has been studied in (Domingos and Hulten 2000, Hulten, Spencer, and Domingos 2001, Ganti, Gehrke, Ramakrishnan 2002, Papadimitriou, Faloutsos, and Brockwell 2003, Wang, Fan, Yu and Han 2003). Extracting frequent items and frequent itemsets have been studied in (Cormode, Muthukrishnan 2003, Giannella, Han, Pei, Yan, and Yu 2003, Manku and Motwani 2002).

The main focus of the above algorithms is how to reduce the number of passes and the number of data elements being tested in order to have an efficient approximate algorithm compared to the traditional techniques. The fact that the main problem in data stream mining is the high data rate and how to develop light-weight data mining techniques that adapt to the incoming data rate according to the available resources is our approach. Our proposed solution for data rate adaptation has the advantage of the generality to any algorithm as well as the simplicity of implementation. The next section presents our algorithm output granularity approach and discusses the issues that result from the adoption of data rate adaptation approach in mining data streams.

3 Data Rate Adaptation

In this section, a formalization of mining data streams problem is given followed by an approach for solving this problem.
High-speed data stream: data rate > processing rate given the processing environment, this requires the development of an algorithm that can result in an approximate solution for the given problem.
The above is the general definition of the data stream processing problem. Thus, our research problem can be formalized as follows:
Given data rate > mining rate (given the processing environment), it is required to design, develop and test resource-aware mining algorithms that can result in an acceptable output accuracy.

Data rate adaptation techniques represented by sampling, aggregation and our proposed technique *algorithm granularity* (AG) try to reduce the data rate in order to give the algorithm the capability to catch up with data in a resource constrained environment. These adaptation techniques can limit the algorithm accuracy. The ultimate goal is to find a solution using one or more of the adaptation techniques in order to maximize the algorithm output accuracy given the available resources (memory, CPU utilization, battery consumption for mobile devices, effective bandwidth). We define algorithm granularity as the amount of generated results kept in main memory before doing any incremental integration in order to catch up with the high data rate. The algorithm granularity is based on the following:
a) The algorithm rate (AR) is function in a data rate (DR), i.e., $AR = f(DR)$. The number of generated cluster centers per unit time for example depends on the data rate.
b) The time needed to fill the available memory by the algorithm results (TM) is function in (AR), i.e., $TM = f(AR)$. The time needed for example to fill the available memory by cluster centers depends on the algorithm rate.
c) The algorithm accuracy (AC) is function in (TM), i.e., $AC = f(TM)$. That is if the time needed to fill the available memory is enough to the algorithm at the highest data rate without sampling, aggregation or algorithm granularity, this would be the best solution
The higher the algorithm granularity, the more accurate the algorithm output will be.
The above discussion raises a number of issues:

1) What is the optimum combination of data rate adaptation techniques that might be used to catch up with the high-speed data stream?
2) How to measure the effect of one strategy on the available resources? For example how to measure the effect of the algorithm granularity on the available memory or CPU utilization.
3) How to measure the algorithm accuracy for any combination of data adaptation techniques?
4) How to dynamically change this combination according to the variability of the available resources in order to achieve the required accuracy?
5) Which data rate techniques are more appropriate to a specific mining algorithm?
6) How to deal with unbounded memory requirements due to the continuous flow of data streams?
7) How to achieve the required accuracy?

Our light-weight algorithms address the issues of unbounded memory requirements and the required accuracy. The next section presents our light-weight mining algorithms using the algorithm output granularity approach.

4 Algorithm Granularity based Mining Techniques

In the following subsections, we show the application of the algorithm output granularity to clustering, classification and frequent items. Since all these algorithms perform only one pass over the data stream, we term them light-weight mining algorithms: LWC, LWClass, and LWF for the above techniques respectively.

4.1 LWC

In this section, our one-look clustering algorithm (LWC) is explained and discussed. The algorithm has two main components. The first one is the resource-aware RA component that uses the data adaptation techniques to catch up with the high-speed data stream and at the same time to achieve the optimum accuracy according to the available resources. It starts by checking the minimum data rate that could be achieved using data adaptation techniques with an acceptable accuracy. If the algorithm can catch up with the minimum data rate, the RA component tries to find a solution that maximizes the accuracy by increasing the data rate. Otherwise the algorithm should send a data mining request to a data mining server that can achieve the minimum acceptable accuracy.
The other component is the LWC algorithm. The algorithm follows the following steps:

1- Data items arrive in sequence with a data rate.
2- The algorithm starts by considering the first point as a center.
3- Compare any new data item with the centers to find the distance.
4- If the distance for all the centers is greater than a threshold, the new item is considered as a new center; else increase the weight for the center that has the shortest distance between the data item and the center by 1 and let the new center equals the weighted average.
5- Repeat 3 and 4.
6- If the number of centers = k (according to the available memory) then create a new centers vector.
7- Repeat 3, 4, 5, and 6.
8- If memory is full then re-cluster (incrementally integrate the clusters)

The algorithm output granularity (k) is represented here by the number of cluster centers' kept in memory before doing any incremental clustering integaration. The higher the algorithm granularity the higher is the algorithm accuracy. The threshold value here represents the minimum distance between any point and the cluster center. The lower the threshold the more the clusters is created.

The algorithm according to the given threshold and the data set domain generates the maximum number of subsequent data items, each of which represents a center; that will be given using the following formula:
Maximum number of subsequent data points that could be centers = [(Maximum item value in the data set - Minimum item value in the data set) / threshold]

Since these points in the worst case might be the first points in the data stream in order for them to be centers, the following formula gives the number of data elements that would do the comparison over the generated centers:
Cluster Members = Data Set Size - [(Maximum item value in the data set - Minimum item value in the data set) / threshold].

Thus the algorithm complexity is *O(nm)*, where "n" is the data set size, and "m" is maximum number of subsequent data points that could be centers.

4.2 LWClass

In this section, we present the application of the algorithm output granularity to light weight K-Nearest-Neighbors classification LWClass. The algorithm starts with determining the number of instances according to the available space in the main memory. When a new classified data element arrives, the algorithm searches for the nearest instance already in the main memory according to a pre-specified distance threshold. The threshold here represents the similarity measure acceptable by the algorithm to consider two or more elements as one element according to the element attributes' values. If the algorithm finds this element, it checks the class label. If the class label is the same, it increases the weight for this instance by one, otherwise it decrements the weight by one. If the weight becomes zero, this element will be released from the memory. The algorithm granularity here could be controlled by the distance threshold value and could be changing over time to cope with the high speed of the incoming data elements. The algorithm procedure could be described as follows:

1) Data streams arrive item by item. Each item contains attribute values for a_1, a_2, …,a_n attributes and the class category.
2) According to the data rate and the available memory, we apply the algorithm output granularity as follows:
    a) Measure the distance between the new item and the stored ones.
    b) If the distance is less than a threshold, store the average of these two items and increase the weight for this average as an item by 1. (The threshold value determines the algorithm accuracy and should be chosen according to the available memory and data rate that determines the algorithm rate).
    This is in case that both items have the same class category. If they have different class categories, decrement the weight by 1 till the weight equals to zero, then delete the item from the memory.
    c) After a time interval threshold for the training, we come up with the classification results.
3) Using the above classification results, we have some items that we need to classify them. According to the available time for the

classification process, we choose nearest K-items and these items will be variable according to the time needed by the process.

4) Find the majority class category taking into account the calculated weights from the K items and this will be the answer for this classification task.

4.3 LWF

In this section, we present our light-weight frequent items LWF algorithm. The algorithm starts by setting the number of frequent items that will be calculated according to the available memory. This number changes over time to cope with the high data rate. The AG is represented here by the number of frequent items that the algorithm can calculate as well as the number of counters that will be re-set after some time threshold to be able to cope with the continuous nature of the data stream. The algorithm receives the data elements one by one and tries to find a counter for any new item and increase the item for the registered items. If all the counters are occupied, any new item will be ignored and the counters will be decreased by one till the algorithm reaches some time threshold a number of the least frequent items will be ignored and their counters will be re-set to zero. If the new item is similar to one of the items in memory according to a similarity threshold, the average of both items will be allocated and the counter will be increased by one. The main parameters that can affect the algorithm accuracy are time threshold, number of calculated frequent items and number of items that will be ignored and their counter will be re-set after some time threshold.

We presented our light weight mining techniques. The empirical studies for the clustering technique will be discussed in the following section.

5 Empirical studies for LWC

The experiments were conducted using Matlab 6.0 in which the LWC is developed and the k-means algorithm included in the Matlab package is used as a guide to measure the algorithm accuracy. The experiments were conducted using a machine with Pentium 4 CPU 2.41 GHz, 480 MB of RAM, and running Windows XP Professional operation system. Data sets are synthesized data generated using random number generators in Matalab. We have conducted a number of experiments to evaluate the algorithm.

Experiment 1: (Figure 1)
Aim: Measure the algorithm running time with different threshold values.
Experiment Setup: Running LWC several times using different threshold values with a synthesized data set.
Results: The higher the threshold the lower the running time.

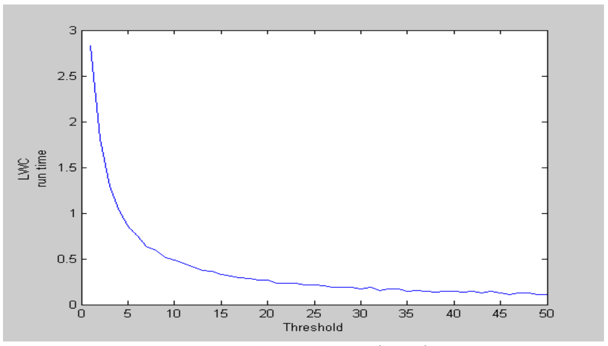

Figure 1: LWC Running Time.

Analysis: We have to minimize the threshold according to the available resources of memory and CPU utilization. We can use the threshold as an output adaptation technique which could be used together with the data rate adaptation techniques discussed earlier. That is because the threshold value controls the algorithm rate (The higher the threshold the lower the algorithm rate). On the other hand, we can use the threshold as an application-oriented parameter that does not affect the accuracy; however it might increase it according to some domain knowledge about the clustering problem that might be known in advance.

Experiment 2: (Figure 2)
Aim: Measuring the algorithm accuracy with different threshold values.
Experiment Setup: Running LWC and K-means several times with different threshold values for LWC. The experiment is repeated three times with different data set sizes.
Results: The lower the threshold the higher the accuracy of the algorithm which is measured as follows: *Accuracy (LWC) = average (|sorted LWC centers – sorted K-means centers|). The lower this measure will be, the higher the accuracy.*

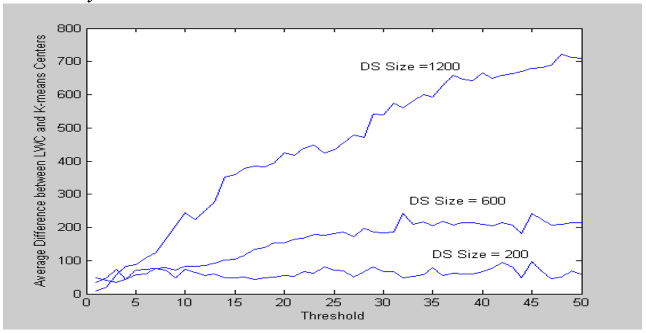

Figure 2: LWC Accuracy (DS Size measured in number of data items).

Analysis: Choosing the threshold value is an important issue to achieve the required accuracy. It should be noted that from this experiment and the previous one the higher the accuracy the higher the running time. And that both factors are affected by the threshold value.

Experiment 3: (Figure 3)
Aim: Measure the LWC algorithm running time against the data set sizes.
Experiment setup: Running the LWC algorithm with different large data sets.
Results: The algorithm has a linear relation with the data set size.

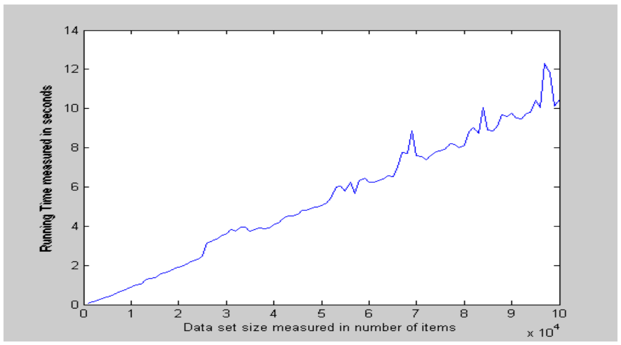

Figure 3: LWC running time with different data set sizes

Analysis: the LWC algorithm is efficient for large data sets due to the linearity of the running time against data set size

Experiment 4: (Figure 4)

Aim: Measuring the effect of the threshold on the above experiment.

Experiment setup: Running LWC algorithm with the same data set sizes as the above experiment, but with decreasing threshold value with each run.

Results: The threshold value affects the running time of the algorithm since the maximum running time in the above experiment is approximately 12 seconds. The maximum running time in this experiment is about 47 seconds.

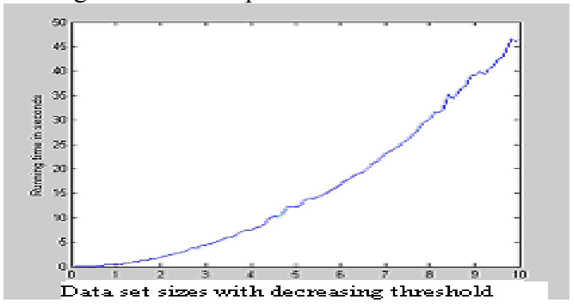

Figure 4: LWC running time with different data set sizes and threshold values

Analysis: According to the application and/or the required accuracy, we have to maximize the threshold value to have more efficient algorithm in terms of running time.

Experiment 5: (Figure 5)

Aim: Comparison between K-means and LWC efficiency.

Experiment setup: Running LWC (with a small threshold value which results in a high accuracy) and K-means several times on the same data sets with different sizes and measuring the running time.

Results: The running time of LWC is low compared to K-means with small data set sizes.

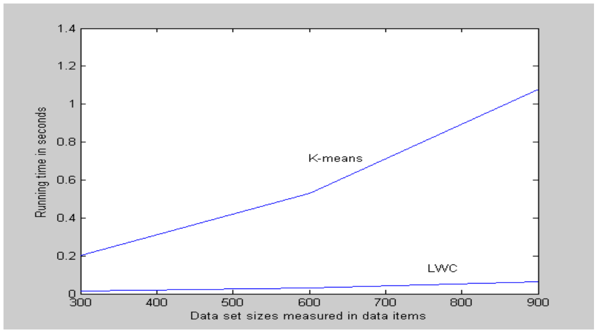

Figure 5: K-means and LWC comparison in terms of running time

Analysis: LWC is efficient compared to K-means for small data sets, when we try to run both on large data sets, we found that LWC over performs the K-means.

The above experiments show an efficient one-look clustering algorithm that is adaptable to the available resources using data adaptation techniques and the control of the threshold value as an algorithm rate adaptation technique.

6 Conclusions and Future Work

In this paper, we show algorithm output granularity as a data rate adaptation technique for mining data streams. The application of this technique to clustering, classification and frequent items is proposed and discussed. The empirical studies of the clustering algorithm are discussed. They show an acceptable accuracy accompanied with efficiency in running time.

The development of frequent items and classification techniques as well as the empirical studies are our future work in testing the efficiency of algorithm output granularity approach in mining data streams. The combination among different data rate adaptation techniques to achieve the required accuracy will be investigated experimentally. Automating the threshold adaptation using linear regression to cope with the change in the data rate, and the deployment of the developed algorithms for potential applications in sensor networks and ubiquitous data analysis business applications such as stock markets are planned for our future work.

7 References

Aggarwal C., Han J., Wang J., Yu P. S. (2003): *A Framework for Clustering Evolving Data Streams*. Proc. 2003 Int. Conf. on Very Large Data Bases (VLDB'03), Berlin, Germany.

Babcock B., Babu S., Datar M., Motwani R., and Widom J.(2002): *Models and issues in data stream systems.* In Proceedings of PODS.

Babcock B., Datar M., and Motwani R. (2003): *Load Shedding Techniques for Data Stream Systems (short paper*). In Proc. of the 2003 Workshop on Management and Processing of Data Streams (MPDS 2003).

Babcock B., Datar M., Motwani R., O'Callaghan L. (2003): *Maintaining Variance and k-Medians over Data Stream Windows*. *To appear* in Proceedings of the 22nd Symposium on Principles of Database Systems (PODS 2003)

Charikar M., O'Callaghan L., and Panigrahy R.(2003): *Better streaming algorithms for clustering problems.* In

Proc. of 35th ACM Symposium on Theory of Computing (STOC).

O'Callaghan L., Mishra N., Meyerson A., Guha S., and Motwani R.(2002): *Streaming-data algorithms for high-quality clustering.* Proceedings of IEEE International Conference on Data Engineering, March.

Cormode G., Muthukrishnan S. (2003): *What's hot and what's not: tracking most frequent items dynamically.* PODS 2003: 296-306

Datar M., Gionis A., Indyk P., Motwani R. (2002): *Maintaining Stream Statistics over Sliding Windows (Extended Abstract).* In Proceedings of 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002).

Domingos P. and Hulten G. (2001). *A General Method for Scaling Up Machine Learning Algorithms and its Application to Clustering.* Proceedings of the Eighteenth International Conference on Machine Learning, 106--113, Williamstown, MA, Morgan Kaufmann.

Domingos P. and Hulten G. (2000) *Mining High-Speed Data Streams.* In Proceedings of the Association for Computing Machinery Sixth International Conference on Knowledge Discovery and Data Mining, pages 71—80.

Ganti V., Gehrke J., Ramakrishnan R. (2002): *Mining Data Streams under Block Evolution.* SIGKDD Explorations 3(2): 1-10.

Garofalakis M., Gehrke J., Rastogi R. (2002): *Querying and mining data streams: you only get one look a tutorial.* SIGMOD Conference 2002: 635

Giannella C., Han J., Pei J., Yan X., and Yu P.S. (2003): *Mining Frequent Patterns in Data Streams at Multiple Time Granularities.* In Kargupta H., Joshi A., Sivakumar K., and Yesha Y. (eds.), Next Generation Data Mining, AAAI/MIT.

Guha S., Mishra N., Motwani R., and O'Callaghan L. (2000): *Clustering data streams.* In Proceedings of the Annual Symposium on Foundations of Computer Science. IEEE, November.

Golab L. and Ozsu M. T. (2003): *Issues in Data Stream Management.* In SIGMOD Record, Volume 32, Number 2, June 2003, pp. 5--14.

Henzinger M., Raghavan P, and Rajagopalan S.(1998): *Computing on data streams.* Technical Note 1998-011, Digital Systems Research Center, Palo Alto, CA, May.

Hulten G., Spencer L., and Domingos P.(2001): *Mining Time-Changing Data Streams.* ACM SIGKDD.

Keogh E., Lin J., and Truppel W. (2003): *Clustering of Time Series Subsequences is Meaningless: Implications for Past and Future Research.* In proceedings of the 3rd IEEE International Conference on Data Mining. Melbourne, FL. Nov 19-22.

Manku G. S. and Motwani R. (2002): *Approximate frequency counts over data streams.* In Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China, August.

Muthukrishnan S. (2003): *Data streams: algorithms and applications.* Proceedings of the fourteenth annual ACM-SIAM symposium on discrete algorithms.

Muthukrishnan S. (2003): *Seminar on Processing Massive Data Sets.* Available Online: http://athos.rutgers.edu/%7Emuthu/stream-seminar.html,

Ordonez C. (2003): *Clustering Binary Data Streams with K-means* .ACM DMKD.

Park B. and Kargupta H. (2002). *Distributed Data Mining: Algorithms, Systems, and Applications.* To be published in the Data Mining Handbook. Editor: Nong Ye.

Papadimitriou S., Faloutsos C., and Brockwell A. (2003): *Adaptive, Hands-Off Stream Mining.* 29th International Conference on Very Large Data Bases VLDB.

Tatbul N., Cetintemel U., Zdonik S., Cherniack M. and Stonebraker M. (2003): *Load Shedding in a Data Stream Manager.* Proceedings of the 29th International Conference on Very Large Data Bases (VLDB), September.

Tatbul N., Cetintemel U., Zdonik S., Cherniack M. and Stonebraker M. (2003): *Load Shedding on Data Streams.* In Proceedings of the Workshop on Management and Processing of Data Streams (MPDS 03), San Diego, CA, USA, June.

Viglas S. D. and Naughton J. (2002): *Rate based query optimization for streaming information sources.* In Proc. of SIGMOD.

Wang H., Fan W., Yu P. and Han J. (2003): *Mining Concept-Drifting Data Streams using Ensemble Classifiers.* In the 9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Aug., Washington DC, USA.