Adaptive Mining Techniques for Data Streams using Algorithm Output Granularity

Mohamed Medhat Gaber[1], Shonali Krishnaswamy[1], Arkady Zaslavsky[1]

[1] School of Computer Science and Software Engineering, Monash University,
900 Dandenong Rd, Caulfield East, VIC3145, Australia
{Mohamed.Medhat.Gaber, Shonali.Krishnaswamy,
Arkady.Zaslavsky}@infotech.monash.edu.au

Abstract. Mining data streams is an emerging area of research given the potentially large number of business and scientific applications. A significant challenge in analyzing/mining data streams is the high data rate of the stream. In this paper, we propose a novel approach to cope with the high data rate of incoming data streams. We termed our approach "algorithm output granularity". It is a resource-aware approach that is adaptable to available memory, time constraints, and data stream rate. The approach is generic and applicable to clustering, classification and counting frequent items mining techniques. We have developed a data stream clustering algorithm based on the algorithm output granularity approach. We present this algorithm and discuss its implementation and empirical evaluation. The experiments show acceptable accuracy accompanied with run-time efficiency. They show that the proposed algorithm outperforms the K-means in terms of running time while preserving the accuracy that our algorithm can achieve.

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 [2], [16], [17], [24], [25]. Data stream analysis has recently attracted attention in the research community. Algorithms for mining data streams and ongoing projects in business and scientific applications have been developed and discussed in [2], [13], [19]. Most of these algorithms focus on developing approximate one-pass techniques.

Two recent advancements motivate the need for data stream processing systems [16],[24] :

- The automatic generation of a highly detailed, high data rate sequence of data items in different scientific and business applications. For example: satellite, radar, and astronomical data streams for scientific applications, and stock market and transaction web log data streams for business applications.
- The need for complex analyses of these high-speed data streams such as clustering and outlier detection, classification, frequent itemsets and counting frequent items.

There are recent projects that stimulate the need for developing techniques that analyze high speed data streams in real time. These include:

- JPL/NASA are developing a project called Diamond Eye [5]. They aim to enable remote systems as well as scientists to analyze spatial objects in real time image stream. The project focuses on enabling "a new era of exploration using highly autonomous spacecraft, rovers, and sensors" [5].
- Kargupta et al. [19], [21] have developed MobiMine. It is a client/server PDA-based distributed data mining application for financial data streams.
- Kargupta et al. [20] have developed The Vehicle Data Stream Mining System (VEDAS) which is a ubiquitous data mining system that allows continuous monitoring and pattern extraction from data streams generated on-board a moving vehicle.
- Tanner et al. [30] are developing EnVironment for On-Board Processing (EVE). This system analyzes data streams continuously generated from measurements of different satellite on-board sensors using data mining, feature extraction, event detection and prediction techniques. Only interesting patterns are sent to the ground processing centre saving the limited bandwidth.
- Srivastava and Stroeve [29] are developing a NASA project for onboard detection of geophysical processes such as snow, ice and clouds using kernel clustering methods for data compression conserving the limited bandwidth needed to send streaming images to the ground centers.

These projects and others demonstrate the need for data stream analysis techniques and strategies that can cope with the high data rate and deliver the analysis results in real time in resource constrained environments.

There are two strategies for addressing the problem of the high speed nature of data streams. Input and output rate adaptation of the mining algorithm is the first strategy. The rate adaptation means controlling the input and output rate of the mining algorithm according to the available resources. The algorithm approximation by developing new light-weight techniques that have only one look at each data item is the second strategy. The main focus of mining data stream techniques proposed so far is the design of approximate mining algorithms that have only one-pass or less over the data stream. In this paper, we propose a novel approach that is able to mine data streams in one pass. Moreover, it is adaptable to memory, time constraints and data stream rate. We termed our approach as algorithm output granularity (AOG). This approach has the advantage of simplicity, generality and is an enhancement of the approximate algorithms research by being resource-aware. That means that the algorithm can adapt the output rate according to available resources.

The paper is organized as follows. Section 2 is a discussion on issues related to mining data streams and proposes our algorithm output granularity approach. One-pass mining techniques using our approach are proposed in section 3. The empirical studies for clustering data streams using algorithm output granularity are shown and discussed in section 4. Section 5 presents related work in mining data streams algorithms. Finally, we conclude the paper and present our future work in section 6.

2 Issues in Mining Data Streams

In this section, we present issues and challenges that arise in mining data streams and solutions that address these challenges. Fig. 1 shows the general processing model of mining data streams.

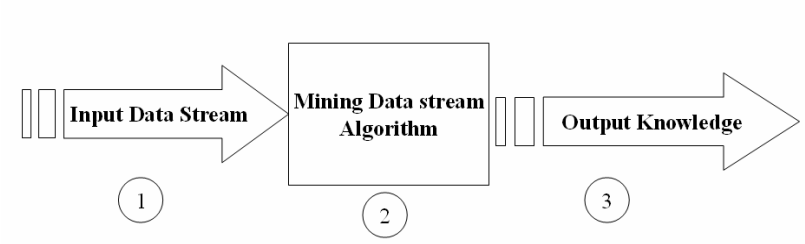

Fig. 1. Mining Data Stream Process

Issues and challenges with mining data streams:

1) Unbounded memory requirements due to the continuous feature of the incoming data elements.
2) Mining algorithms require several passes over data streams and this is not applicable because of the high data rate feature of the data stream.
3) Data streams generated from sensors and other wireless data sources create a real challenge to transfer these huge amounts of data elements to a central server to be analyzed.

There are several strategies that address these challenges. These include:

1) **Input data rate adaptation:** this approach uses sampling, filtering, aggregation, and load shedding on the incoming data elements. Sampling is the process of statistically selecting the elements of the incoming stream that would be analyzed. Filtering is the semantics sampling in which the data element is checked for its importance for example to be analyzed or not. Aggregation is the representation of number of elements in one aggregated elements using some statistical measure such as the average. While load shedding, which has been proposed in the context of querying data streams [3], [31], [32], [33] rather than mining data streams, is the process of eliminating a batch of subsequent elements from being analyzed rather than checking each element that is used in the sampling technique. Fig. 2 illustrates the idea of data rate adaptation from the input side using sampling.

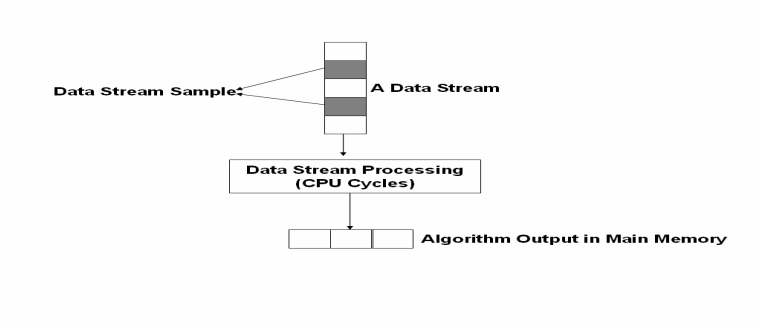

Fig. 2. Data Rate Adaptation using Sampling

2) **Output concept level:** using the higher concept level in applying data mining in order to cope with the data rate, that is to categorize the incoming elements into a limited number of categories and replacing each incoming element with the matching category according to a specified measure or a look-up table. This would produce fewer results conserving the limited memory. Moreover, it would require fewer number of processing CPU cycles.
3) **Approximate algorithms:** design one pass mining algorithms to approximate the mining results according to some acceptable error margin.
4) **On-board analysis**: To avoid transferring huge amounts of data, the data mining would be done at the data source location. For example, (VEDAS) project [20], (EVE) project [30] and Diamond Eye project [5]. This however assumes the availability of significant computational resources at the site of data stream generation.
5) **Algorithm output granularity:** This is our proposed solution approach. It uses a control parameter as a part of the algorithm logic to control the output rate of the algorithm according to the available memory, the remaining time to fill the available memory before incremental knowledge integration takes place and the data rate of the incoming stream. Fig. 3 shows the idea of our proposed approach.

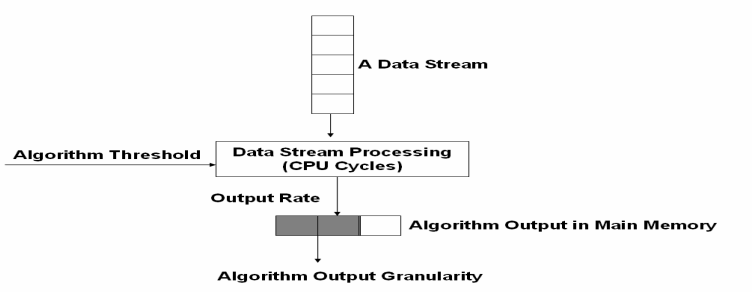

Fig. 3. Algorithm Output Granularity Approach

Algorithm output granularity:

To demonstrate our approach in mining data streams, we first define the following terms:

Algorithm threshold: is a controlling parameter built in the algorithm logic that encourages or discourages the creation of new outputs according to three factors that vary over temporal scale:

a) Available memory.
b) Remaining time to fill the available memory.
c) Data stream rate.

Output granularity: is the amount of generated results that are acceptable according to specified accuracy measure. This amount should be resident in memory before doing any incremental integration.

Time threshold: is the required time to generate the results before any incremental integration according to some accuracy measure. This time might be specified by the user or calculated adaptively based on the history of running the algorithm.

The main steps for mining data streams using our proposed approach:

1) Determine the time threshold and the algorithm output granularity.
2) According to the data rate, calculate the algorithm output rate and the algorithm threshold.
3) Mine the incoming stream using the calculated algorithm threshold.
4) Adjust the threshold after a time frame to adapt with the change in the data rate using linear regression.
5) Repeat the last two steps till the algorithm lasts the time interval threshold.
6) Perform knowledge integration of the results.

The following section will show the use of algorithm output granularity in clustering, classification and frequent items mining algorithms.

3 Algorithm Granularity based Mining Techniques

In the following subsections, we show the application of the algorithm output granularity to clustering, classification and frequent items.

3.1 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. The process 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 (integrate clusters) and send to the server if needed.

The algorithm output granularity (k) is represented here by the number of cluster centers' kept in memory before doing any incremental re-clustering. 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.

Fig. 4 shows the pseudo code for this algorithm. The following is the notation used in the algorithm pseudo code.

Let *D* be the data rate in items/second.
Let *Max(D)* be unfiltered data rate in items/second.
Let *Min(D)* be filtered and aggregated data rate in items/second.
Let *AR* be algorithm rate: number of centers generated by the algorithm in centers/second.
Let *Dist* be the minimum distance between any point and the cluster center.
Let *M* be number of memory blocks, each block can store one center.
Let *T* be the time needed for generating a number of Centers that can fit all the memory blocks in seconds.
Let *TT* be the time threshold that is required for the algorithm accuracy in seconds.

```
1. x = 1, c=1, M = number of memory blocks avail-
   able
2. Receive data item DI[x].
3. Center[c] = DI[x].
4. M = M -1
5. Repeat
       a. x = x+1
       b. Receive DI[x]
       c. For i = 1 to c
            Measure the distance between Center[i]
          and DI[x]
        d. If distance > dist (The threshold)
           Then
                c=c+1
                if (M <> 0)
                Then
                      Center[c] = DI[x]
                Else
                      Recluster DI[]
          Else
          For j=1 to c
          Compare between Center[j] and DI[x] to
          find the shortest distance.
          Increase the weight for the Center[j] with
          the shortest distance.
          Center[j] = (Center[j] * weight + DI[x]) /
          (weight + 1)
Until Done
```

Fig. 4. Light-Weight Clustering Algorithm

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.

We have performed experimental evaluation and compared our algorithm with k-means. The results presented in Section 4 shows that our algorithm outperforms k-means in running time with an acceptable accuracy.

3.2 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 a1, a2, …,an 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, we delete both).
    c) After a time threshold for the training, we come up with a sample result like the one in table 1.

Table 1. Sample LWClass Training Results

A1	**A2**	**…**	**An**	**Class**	**Weight**
Value(a1)	Value(a2)	…	Value(an)	Class category	X (represents that X items contribute in the values of this tuple)
Value(a1)	Value(a2)	…	Value(an)	Class category	Y
Value(a1)	Value(a2)	…	Value(an)	Class category	Z

3) Using the above table, 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.

3.3 LWF

In this section, we present 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 main idea behind the algorithm is the algorithm output granularity. 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. Fig. 5 shows the algorithm outline for the LWF algorithm.

```
1- Set the number of the top frequent items to k.
2- Set a counter for each k.
3- Repeat
       a. Receive the item.
       b. If the item is new and one of the k count-
          ers are 0
          Then
          Put this item and increase the counter by
          1.
          Else
          If the item is already in one of the k
          counters.
          Then
          Increase the counter by 1.
          Else
          If the item is new and all the counters are
          full
          Then
          Check the time
          If time > Threshold Time
          Then
          Re-set number of least n of k counters to 0
          Put the new item and increase the counter
          by 1
          Else
          Ignore the item.
          Decrease all the counters by 1.
     Until Done
```

Fig. 5. LWF Algorithm

4 Empirical studies for LWC

In this section, we discuss our empirical results for the LWC algorithm. 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.

There are three main parameters that we measure in our experiments; algorithm threshold, running time and accuracy. We have conducted a number of experiments to evaluate the algorithm.

Experiment 1: (Fig. 6)

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.

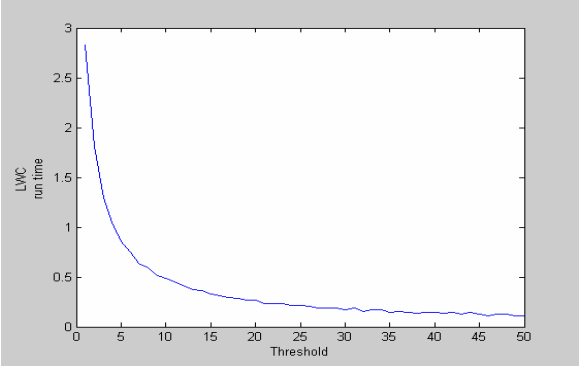

Fig. 6. LWC Running Time.

Analysis: We have to minimize the threshold according to the available resources of memory and CPU utilization. The threshold is an rate output adaptation technique. 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: (Fig. 7)

Aim: Measuring the algorithm accuracy with different threshold values.

Experiment Setup: Running LWC and K-means several times with different threshold values. 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.*

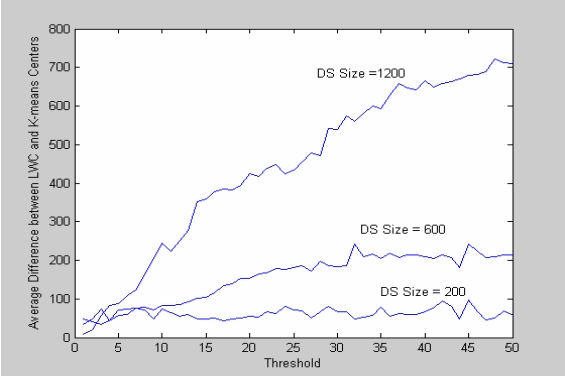

Fig. 7. LWC Accuracy (DS Size measured in number of data items).

Analysis: Choosing the threshold value is an important issue to achieve the required accuracy. It should be pointed out 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: (Fig. 8)

Aim: Comparison of K-means and LWC centers.

Experiment setup: Running LWC and K-means several times with the same threshold but different data set sizes.

Results: Assuming that the accuracy of K-means algorithm is high because it mines static data sets with any number of passes. The experiment shows that LWC generates similar centers that K-means algorithm generates.

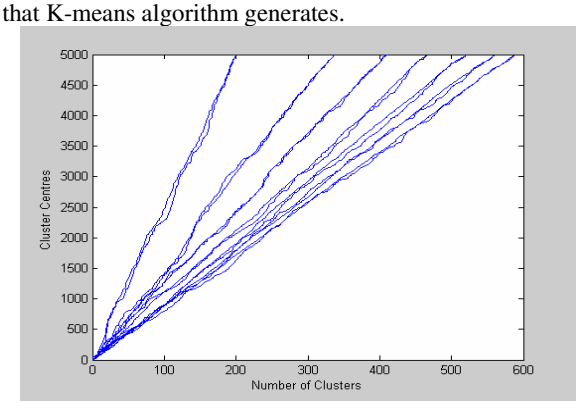

Fig. 8. LWC compared to K-means

Analysis: The accuracy of LWC is acceptable because it is very similar to k-means results that process the data set as static stored data set and not streaming data. That means that k-means algorithm performs several passes over the data set to result in the final cluster centers. As shown in the figure, the seven experiments show very similar cluster centers for our one-pass algorithm compared to k-means.

Experiment 4: (Fig. 9)

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.

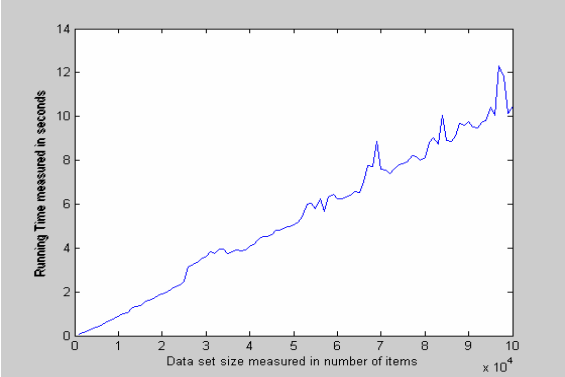

Fig. 9. LWC running time with different data set sizes

Analysis: the LWC algorithm is efficient for large data sets due to the linearity of the running time with data set size. This linearity results from performing only one-pass over the data stream. It is worth to point out here that the data stream rate is the major factor that control the behavior of LWC since the higher the rate the larger the size of the data set.

Experiment 5: (Fig. 10)

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.

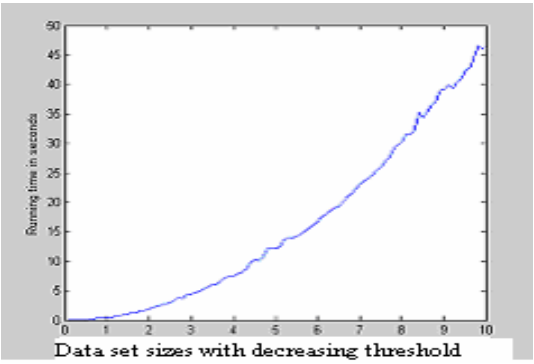

Fig. 10. LWC running time with different data set sizes and threshold values

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. The algorithm threshold would be controlled according to the available memory and a time threshold constraint that represents the algorithm accuracy.

Experiment 6: (Fig. 11)

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.

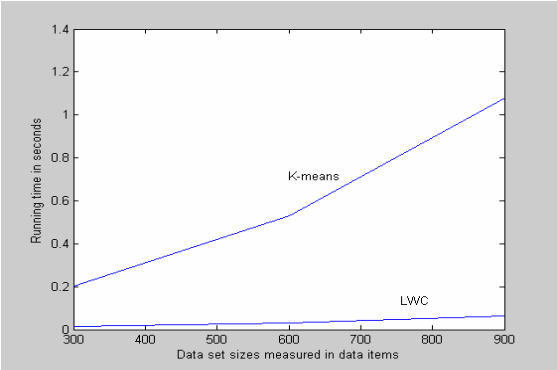

Fig. 11. K-means and LWC comparison in terms of running time

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 outperforms the K-means. The LWC runs with highest possible accuracy (the least threshold value) and outperforms k-means with different data set sizes.

The above experiments show an efficient one-look clustering algorithm that is adaptable to the available resources using our algorithm output granularity approach. The LWC outperforms k-means in terms of running time and has the advantage of linearity of running time with the increase in the data set sizes. The algorithm threshold is the controlling parameter of algorithm accuracy, efficiency, and algorithm output rate.

5 Related Work

There are different algorithms proposed to deal with the high speed nature for mining data streams using different techniques. Clustering data streams has been studied in [1], [4], [6], [7], [9], [10], [15], [22], [26]. Data stream classification has been studied in [11], [12], [18], [28], [34]. Extracting frequent items and frequent itemsets have been studied in [8], [14], [23].

The above algorithms deal with the problem of mining data streams using different methodologies. These algorithms basically focus on the design of approximate algorithms for mining data streams. However these approaches are not resource-aware and do not focus on adaptation strategies to cope with high data rates, our approach for output rate adaptation is resource-aware approach that can adapt to the available resources.

6 Conclusions and Future Work

In this paper, we discussed the problems of mining data streams and proposed possible solutions. Our algorithm output granularity approach in mining data streams has been presented and discussed. The proposed approach is distinguished from previous work in mining data streams by being resource-aware. We have developed a one-pass mining data streams algorithm. The application of the proposed approach to clustering, classification and counting frequent items has been presented. The implementation and empirical studies of our LWC algorithm have been demonstrated. The experiments showed an acceptable accuracy accompanied with efficiency in running time that outperforms k-means algorithm. Having implemented and tested LWC, we are developing LWClass and LWF. The application of these algorithms in a ubiquitous environment is planned for future work. The simplicity, generality, and efficiency of our proposed approach in mining data streams facilitate the application of the algorithms in various scientific and business applications that require data stream analysis.

References

1. Aggarwal C., Han J., Wang J., Yu P. S.: A Framework for Clustering Evolving Data Streams. Proc. 2003 Int. Conf. on Very Large Data Bases (VLDB'03), Berlin, Germany (2003).
2. Babcock B., Babu S., Datar M., Motwani R., and Widom J.: Models and issues in data stream systems. In Proceedings of PODS (2002).
3. Babcock B., Datar M., and Motwani R.: Load Shedding Techniques for Data Stream Systems (short paper). In Proc. of the 2003 Workshop on Management and Processing of Data Streams (MPDS 2003) (2003).
4. Babcock B., Datar M., Motwani R., O'Callaghan L.: Maintaining Variance and k-Medians over Data Stream Windows. To appear in Proceedings of the 22nd Symposium on Principles of Database Systems (PODS 2003) (2003).
5. Burl M., Fowlkes C., Roden J., Stechert A., and Mukhtar S. Diamond Eye: A distributed architecture for image data mining. In SPIE DMKD, Orlando, April (1999).
6. Charikar M., O'Callaghan L., and Panigrahy R.: Better streaming algorithms for clustering problems. In Proc. of 35th ACM Symposium on Theory of Computing (STOC) (2003).

7. O'Callaghan L., Mishra N., Meyerson A., Guha S., and Motwani R.: Streaming-data algorithms for high-quality clustering. Proceedings of IEEE International Conference on Data Engineering, March (2002).
8. Cormode G., Muthukrishnan S.: What's hot and what's not: tracking most frequent items dynamically. PODS 2003. (2003) 296-306
9. Datar M., Gionis A., Indyk P., Motwani R.: Maintaining Stream Statistics over Sliding Windows (Extended Abstract). In Proceedings of 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002) (2002).
10. Domingos P. and Hulten G., 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. (2001)
11. Domingos P. and Hulten G. Mining High-Speed Data Streams. In Proceedings of the Association for Computing Machinery Sixth International Conference on Knowledge Discovery and Data Mining, (2000) 71—80.
12. Ganti V., Gehrke J., Ramakrishnan R.: Mining Data Streams under Block Evolution. SIGKDD Explorations 3(2): (2002) 1-10.
13. Garofalakis M., Gehrke J., Rastogi R.: Querying and mining data streams: you only get one look a tutorial. SIGMOD Conference 2002: 635. (2002).
14. Giannella C., Han J., Pei J., Yan X., and Yu P.S.: 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 (2003).
15. Guha S., Mishra N., Motwani R., and O'Callaghan L.: Clustering data streams. In Proceedings of the Annual Symposium on Foundations of Computer Science. IEEE, November (2000).
16. Golab L. and Ozsu M. T. : Issues in Data Stream Management. In SIGMOD Record, Volume 32, Number 2, June (2003) 5-14.
17. Henzinger M., Raghavan P, and Rajagopalan S.: Computing on data streams. Technical Note 1998-011, Digital Systems Research Center, Palo Alto, CA, May (1998).
18. Hulten G., Spencer L., and Domingos P.: Mining Time-Changing Data Streams. ACM SIGKDD (2001).
19. Kargupta H.: CAREER: Ubiquitous Distributed Knowledge Discovery from Heterogeneous Data. NSF Information and Data Management (IDM) Workshop (2001).
20. Kargupta. H.: VEhicle DAta Stream Mining (VEDAS) Project. http://www.cs.umbc.edu/%7Ehillol/vedas.html. (2003).
21. Kargupta, H., Park, B., Pittie, S., Liu, L., Kushraj, D. and Sarkar, K. (2002). MobiMine: Monitoring the Stock Market from a PDA. ACM SIGKDD Explorations. January 2002. Volume 3, Issue 2. Pages 37--46. ACM Press.
22. Keogh E., Lin J., and Truppel W.: 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. November (2003) 19-22.

23. Manku G. S. and Motwani R.: Approximate frequency counts over data streams. In Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China, August (2002).
24. Muthukrishnan S.: Data streams: algorithms and applications. Proceedings of the fourteenth annual ACM-SIAM symposium on discrete algorithms (2003).
25. Muthukrishnan S.: Seminar on Processing Massive Data Sets. Available Online: http://athos.rutgers.edu/%7Emuthu/stream-seminar.html (2003).
26. Ordonez C.: Clustering Binary Data Streams with K-means .ACM DMKD (2003).
27. Park B. and Kargupta H.. Distributed Data Mining: Algorithms, Systems, and Applications. Data Mining Handbook. Editor: Nong Ye (2002).
28. Papadimitriou S., Faloutsos C., and Brockwell A.: Adaptive, Hands-Off Stream Mining. 29th International Conference on Very Large Data Bases VLDB (2003).
29. Srivastava A. and Stroeve J.: Onboard Detection of Snow, Ice, Clouds and Other Geophysical Processes Using Kernel Methods. Proceedings of the ICML'03 workshop on Machine Learning Technologies for Autonomous Space Applications (2003).
30. Tanner S., Alshayeb M., Criswell E., Iyer M., McDowell A., McEniry M., Regner K., EVE: On-Board Process Planning and Execution, Earth Science Technology Conference, Pasadena, CA, Jun. 11 - 14, (2002).
31. Tatbul N., Cetintemel U., Zdonik S., Cherniack M. and Stonebraker M.: Load Shedding in a Data Stream Manager. Proceedings of the 29th International Conference on Very Large Data Bases (VLDB), September (2003).
32. Tatbul N., Cetintemel U., Zdonik S., Cherniack M. and Stonebraker M.: Load Shedding on Data Streams. In Proceedings of the Workshop on Management and Processing of Data Streams (MPDS 03), San Diego, CA, USA, June (2003).
33. Viglas S. D. and Naughton J.: Rate based query optimization for streaming information sources. In Proc. of SIGMOD (2002).
34. Wang H., Fan W., Yu P. and Han J.: 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 (2003).