Mining Top-K Itemsets over a Sliding Window Based on Zipfian Distribution

Raymond Chi-Wing Wong, Ada Wai-Chee Fu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
cwwong,adafu@cse.cuhk.edu.hk

Abstract

Frequent pattern discovery in data streams can be very useful in different applications. In time critical applications, a sliding window model is needed to discount stale data. In this paper, we adopt this model to mine the K most interesting itemsets, or to estimate the K most frequent itemsets of different sizes in a data stream. In our method, the sliding window is partitioned into buckets. *We maintain the statistics of the frequency counts of the itemsets for the transactions in each bucket. We prove that our algorithm guarantees no false negatives for any data distributions. We also show that the number of false positives returned is typically small according to Zipfian Distribution. Our experiments on synthetic data show that the memory used by our method is tens of times smaller than that of a naive approach, and the false positives are negligible.*

1 Introduction

Data mining processing is typically time-consuming. However, there are some recent demands on real-time data mining for unbounded data stream arriving at high speed. Examples include financial data monitoring and network monitoring. The mining process becomes much more difficult because it requires not only the handling of massive unbounded data stream but also the ability to return the results within a short time.

With limited memory storage, it is natural to devise methods to store some kinds of statistics or summary of the data stream. Until now, most research work consider all data read so far. However, in many applications, old data are less important or not relevant, compared with more recent data. There are two common approaches to deal with this issue. The first one is *aging* [11, 5], where each data is assigned a weight, with more weight for more recent data (e.g. exponential-decay model). Another approach is to use a *sliding window* [2, 4, 8, 3, 9], so that only the most recent W data elements in the data stream is considered, where W is the width of a sliding window. In this paper, we adopt the second approach.

For association rule mining, the difficult subproblem of frequent itemset discovery has been the focus of research for some time. Many motivating examples are given in [12] for the mining of frequent itemsets in data streams. A major issue with mining frequent itemsets is that user has to define a support or frequency threshold s on the resulting itemsets, and without any guidance, this is typically a wild guess. In some previous study [6, 7], it is found that, in different data sets, or even with different subsets of the same data set, the proper values of s can differ by an order of magnitude. In most previous work of data stream mining a major concern is to minimize the error of the false positive to a small fraction of s. However, if the threshold s is not appropriate in the first place, such a guarantee is quite pointless.

Therefore, it is of interest to replace the requirement of a frequency threshold to that of the simpler threshold on the amount of results. It is much easier for users to specify that say the 20 most frequent patterns should be returned. Some previous work assumed that such a threshold can be applied to itemsets of all sizes. However, there is a major pitfall with such an assumption. It is that it implies a uniform frequency threshold for itemsets of all sizes. It is obvious that small size itemsets have an intrinsic tendency to appear more often than large size itemsets. The result from this assumption is that smaller size itemsets can dominate and hide some interesting large size itemsets. The mining of closed patterns does not help much. For example, an interesting closed itemsets X of size 4 may have a frequency of 0.01, while many smaller size closed itemsets have frequencies above 0.l, and hence X cannot hope to reach the top K frequency. Therefore, some previous work has proposed to mine the K most frequent itemsets of size l, for each l that is within a range of sizes specified by user. We shall focus on this mining problem for data streams.

Let us call an itemset of size l an l-itemset. Our problem is about mining K l-itemsets with the greatest frequencies (supports) for each l up to a certain L. We shall tackle this problem for a data stream with a sliding window of size m (contains m transctions). In our approach, the sliding window is divided into n_B partitions, called *buckets*. Each bucket corresponds to a set of transactions and we maintain the statistics for the transactions in each bucket separately. The window is slided forward one bucket at a time. When the window is advanced, the oldest bucket is discarded and a newly generated bucket is appended to the sliding window. At the same time, the candidate top K interesting itemsets are adjusted. Our method have some guarantees for the results. It gives no false negatives for any data distribution. Given a Zipfian data distribution with Zipfian parameter θ and an error parameter $\epsilon > 0$, it outputs no more than $K[(1+\epsilon)^{1/\theta} - 1]$ false positives. The memory usage of our algorithm is bounded by $O(\frac{n_B^{1+2/\theta} m^{1/\theta} K}{\epsilon^{1/\theta}})$. From our experiment, we found that error in frequency of false positives is very small, and the proposed method can achieve memory usage that is many times less than a more naive approach.

2 Problem Definitions and Terminologies

In this section we introduce the problem definition and also other terminologies.

Problem Definition: The data stream is considered a sequence of $equisized$ data buckets with s_B transactions each. The most recent n_B full buckets in the data stream is considered as the sliding window. Given two positive integers K and L. For each l, where $l \leq L$, and let the K-th highest frequency among all l-itemsets in the sliding window be $f(l)$, find all l-itemsets with frequencies greater than or equal to $f(l)$ in the sliding window. These are called the top K l-itemsets. $\square$

Note that in the above definition, at any time, there will be a most recent bucket B, which may or may not be full. A bucket is full when it contains s_B transactions. When a transaction T arrives at the data stream, it will be inserted into B if it is not full; otherwise, a new bucket containing only T will be created and becomes the most recent bucket B. Let $m = n_B \times s_B$. Hence the size of the sliding window is m (number of transactions). The sliding window contains buckets B_1, B_2, ..., B_{n_B}, in chronological order, where bucket B_{n_B} represents the most recently created bucket.

In our algorithm we need to process itemsets of sizes l, $1 \leq l \leq L$. Without loss of generality let us consider size l itemsets, for a certain l. We are going to find the top K l-itemsets.

Let l-itemset denote itemset of size l. Each bucket B_i stores a list of entries (e, f), where e is one of the top $K'_{i,l} \geq K$ l-itemsets and f is the frequency of e in the bucket. [1] We use $f_{i,e}$ to denote the frequency of e in bucket B_i. We say that $f_{i,e}$ is recorded if e is among the top $K'_{i,l}$ itemsets.

[1]Note that $K'_{i,l}$ can be different for different l and will be determined by our algorithm automatically.

Therefore, each bucket stores information about the top $K'_{i,l}$ frequent itemsets.

Let $f_{i,min}$ be the frequency of the $K'_{i,l}$-th frequent l-itemset in bucket B_i. For entry e in bucket B_i, $f_{i,e} \geq f_{i,min}$. We define $min(e)$ and $max(e)$. $min(e) = \sum_{\substack{B_i \text{ and } f_{i,e} \\ \text{is recorded}}} f_{i,e}$ and $max(e) = \sum_{\substack{B_i \text{ and } f_{i,e} \\ \text{is recorded}}} f_{i,e} + \sum_{\substack{B_i \text{ and } f_{i,e} \\ \text{is not recorded}}} (f_{i,min} - 1)$.

When we sum up all recorded frequencies $f_{i,e}$ of itemset e in different buckets B_i, this value should be the least possible frequency of itemset e. However, in some buckets B_i, there may be no recorded frequencies. The itemset e may appear in those buckets. To estimate the maximum possible frequency, we assume the maximum possible frequency for itemsets e with no recorded frequency, and this frequency is $f_{i,min} - 1$ for buckets B_i. Therefore $min(e)$ is the minimum possible frequency of itemset e in the sliding window while $max(e)$ is the maximum possible frequency of itemset e in the sliding window. Let f_e be the frequency of e in the sliding window. Thus, $min(e) \leq f_e \leq max(e)$.

We define $f_{(1)} = max_e\{min(e)\}$, which is the greatest value of $min(e)$ among all e. We define $M_l = min_{B_i}\{f_{i,min}\}$, which is the minimum value of $f_{i,min}$ among all buckets. We also define $\triangle_{max,l} = \sum_{B_i} f_{i,min} - M_l$.

3 Algorithm

Let the size of the sliding window be m (there are m transactions). There are n_B buckets in the sliding window. So, the bucket size s_B is $\lceil m/n_B \rceil$. For each full bucket we store a list of entries (e, f). The 2 major steps of our algorithm will be introduced in this section. At the beginning of the algorithm, we process the first full bucket containing the transactions at the beginning of the data stream in Step 1. For each new bucket, we need to accumulate s_B transactions in the memory temporarily. After receiving the s_B transactions, we process the transactions with Step 2. Every time a bucket leaves the sliding window, the bucket and its entries will be removed.

There are two major parameters in our algorithm - (1) θ and (2) ϵ. (1) θ is a Zipfian parameter in a Zipfian distribution. The greater the value of θ, the greater the skewness of the distribution. The Zipfian parameter $\theta \geq 1$ is commonly used in the Zipfian distribution in previous research on data streams[12, 10]. In [12], $\theta = 1.25$; and [10] sets θ to be 1.0, 1.25 and 1.5. In our real data set, we found that the distribution is quite skew, which also corresponds to $\theta \geq 1$. (2) ϵ is an error parameter. The smaller the value of ϵ is, the more accurate the algorithm is. However, with a small value of ϵ, the memory consumption will be great. So, ϵ is a user input parameter of our algorithm. It can determine the storage and the accuracy of our algorithm. The the accuracy bound and storage bound can be found in Corollaries 1 and

2, respectively in Section 4.

The major steps of our algorithm are described as follows.

1. After receiving the first bucket of transactions at the beginning of the data stream, we do the following. Let $r_0 = [n_B(n_B - 1)(\frac{1}{2K^\theta(1+1/\epsilon)} + \frac{n_B - 1}{m})^{-1}]^{1/\theta}$, [2] where θ is the Zipfian parameter and ϵ is an error parameter. If r_0 is greater than the number of possible itemsets, r_0 is assigned to be the number of possible itemsets.

   (a) find top r_0 itemsets of size l
   For this task, we can use an existing algorithm for mining top K itemsets (e.g. [7]).

   (b) store the entries (e, s) of the itemsets found

2. After the first bucket, we can process other buckets B_i in the following way. We define $max'(e)$ with the same definition of $max(e)$ but $max'(e)$ is evaluated with the scope of all buckets in the current sliding window except for the bucket B_i.

   (a) find the K-th largest value of $min(e)$ of itemset e of size l, $K_{min,l}$, within the current bucket B_i and all previous buckets in the sliding window

   (b) Determine the rank $r_{i,e}$ of each e in bucket B_i. Find the greatest rank $r_{i,e}$, say $\widetilde{r_i}$, in order that $max'(e) + f_{i,e} \geq K_{min,l}$ and $r_{i,e} \leq r_0$. Store all entries of itemsets e of size l with $r_{i,e} \leq \widetilde{r_i}$. Again we can make use of the existing algorithm in [7].

   (c) calculate $\widetilde{\triangle}_{max,l} = \frac{f_{(1)}}{2K^\theta(1+1/\epsilon)}$. If $\triangle_{max,l} > \widetilde{\triangle}_{max,l}$, then store the additional next top frequent itemsets in the bucket (if any) until $\triangle_{max,l} \leq \widetilde{\triangle}_{max,l}$. [3]

3. We continue our process in Step 2. Whenever a bucket leaves the sliding window, we can remove the entries in that bucket and the bucket itself.

4. We output the result on demand. We find the K-th largest value of $min(e)$ of itemset e of size l, say $K_{min,l}$, for all buckets in the sliding window. Then, we output all itemsets e of size l with $max(e)$ greater than or equal to $K_{min,l}$.

Theorem 1 *For any data distribution, the proposed algorithm gives no false negatives.*

[2]We shall see in Section 4 that r_0 is a bound on the ranks of itemsets that we keep in all buckets.

[3]Storing more top frequent itemsets can lead to a smaller value of $f_{i,min}$ and thus $\triangle_{max,l}$.

Proof: In the algorithm, the K-th largest value of $min(e)$ (i.e. $K_{min,l}$) is found. In this step, we make sure that we have found K l-itemsets e where $min(e) \geq K_{min,l}$. Also these are at least K itemsets found in the algorithm, which have the chance to become the top K itemsets.

The possible values of frequency of an itemset e are in the range between $min(e)$ and $max(e)$. Hence the only other itemsets e which have the chance to become the top K itemsets are those with $max(e) \geq K_{min,l}$. Thus, the entries with $max(e) \geq K_{min,l}$ are in the output. This ensures that no top K l-itemset will be missed, for all l. $\square$

The above theorem shows the correctness. It is quite easy to understand all steps in our algorithm except for Step 2b and Step 2c. The purpose of Step 2b is to store as few entries as possible. Meanwhile, the accuracy can be maintained. We prune all entries e with $r_{i,e} > \widetilde{r_i}$ even though the entries satisfy $max(e) \geq K_{min,l}$. After pruning those entries, we can save a lot of space and can still maintain the accuracy. Step 2c is to maintain the inequality $\triangle_{max,l} \leq \widetilde{\triangle}_{max,l}$ by making $\triangle_{max,l}$ smaller and smaller. When $\triangle_{max,l}$ is smaller, $f_{i,min}$ is also made to be smaller at the same time. This implicitly means that more itemsets are stored and a smaller value of $max(e)$ which depends on $f_{i,min}$ is calculated. When $max(e)$ is smaller, the number of possible frequencies of each itemset in the range between $min(e)$ and $max(e)$ is smaller, leading to a higher accuracy of our algorithm. Thus, the number of false positves in the output can be reduced.

4 Analysis

In this section, we are going to analyze our algorithm, and show some useful properties.

We first consider the number of false positives. From our analysis, we have the following theorem.

Theorem 2 *The frequency difference between any l-itemset which is a false positive returned by the algorithm and the K-th frequent l-itemset is at most $2\widetilde{\triangle}_{max,l}$.*

Recall that $\widetilde{\triangle}_{max,l} = \frac{f_{(1)}}{2K^\theta(1+1/\epsilon)}$. The following table shows $\widetilde{\triangle}_{max,l}$ for some particular values of $f_{(1)}$, K and ϵ. In the following table, we observe that $\widetilde{\triangle}_{max,l}$ is small relative to $f_{(1)}$. By Theorem 2, The frequency difference between any l-itemset which is a false positive returned by the algorithm and the K-th frequent l-itemset is small, which can be shown in Table 1 (a).

In the remaining discussion of this section we assume that the l-itemsets in the sliding window follow the Zipfian distribution. We have derived the following theorem and corollary.

Theorem 3 *Our algorithm outputs the itemsets of ranks r*

$f_{(1)}$	K	ϵ	$\widetilde{\triangle}_{max,l}$
1,000	20	0.5	8.33
1,000	20	1	12.50
1,000	10	1	25.00
10,000	20	1	125.00

(a)

θ	ϵ	Max. No. of False Positives
1	1	K
1	0.5	$0.5 \times K$
2	1	$0.41 \times K$
0.5	1	$3 \times K$

(b)

K	n_B	ϵ	θ	m	Max. No. of Entries
20	10	0.5	1	500,000	107,767
20	10	1	1	500,000	71,896
20	10	1	2	500,000	3,741
20	20	1	1	500,000	606,157

(c)

Table 1. Some values of the theoretical bound

Table 2.1

L	Stream Algorithm		BOMO		Ratio
	Structure	Recent Bucket	Structure	Sliding Window	
1	810K	400K	8M	40M	39.66
3	2665K	400K	8M	40M	15.66
5	4667K	400K	8M	40M	9.47
7	6867K	400K	8M	40M	6.60

Table 2.2

s_B	Stream Algorithm		BOMO		Ratio
	Structure	Recent Bucket	Structure	Sliding Window	
2K	5979K	80K	1.6M	8M	1.58
4K	5799K	160K	3.2M	16M	3.22
6K	6069K	240K	4.8M	24M	4.56
8K	5744K	320K	6.4M	32M	6.33
10K	5735K	400K	8M	40M	7.82

Table 2.3

K	Stream Algorithm		BOMO		Ratio
	Structure	Recent Bucket	Structure	Sliding Window	
1	4595K	400K	8M	40M	10.01
10	5680K	400K	8M	40M	7.89
20	5735K	400K	8M	40M	7.82
50	5769K	400K	8M	40M	7.78
100	5780K	400K	8M	40M	7.77

Table 2. Synthetic Data Set: Memory Usage (Default $\epsilon = 1$, $L = 6$, $K = 20$, s_B = 10K and n_B = 100)

within the sliding window with the following bound.

$$r \le K(1+\epsilon)^{1/\theta}$$

Corollary 1 *The number of false positives returned by our algorithm is no more than $K[(1+\epsilon)^{1/\theta} - 1]$.*

Table 1 (b) gives the bound of false positives for some values of θ and ϵ.

Next, we are going to analyze the storage capacity in each bucket and in the whole sliding window. Additionally, we have proved that there is a bound of the entries stored in buckets in the following theorem and corollary.

Theorem 4 *Each bucket stores entries of ranks smaller than or equal to r, where*

$$r \le [n_B(n_B-1)(\frac{1}{2K^\theta(1+1/\epsilon)} + \frac{n_B-1}{m})^{-1}]^{1/\theta}$$

Note that r_0 = $[n_B(n_B-1)(\frac{1}{2K^\theta(1+1/\epsilon)} + \frac{n_B-1}{m})^{-1}]^{1/\theta}$.

Corollary 2 *Our algorithm stores at most $n_B[n_B(n_B - 1)(\frac{1}{2K^\theta(1+1/\epsilon)} + \frac{n_B-1}{m})^{-1}]^{1/\theta}$ entries in all buckets. The memory required is $O(\frac{n_B^{1+2/\theta} m^{1/\theta} K}{\epsilon^{1/\theta}})$.*

Proof: By Theorem 4, each bucket should store at most $[n_B(n_B - 1)(\frac{1}{2K^\theta(1+1/\epsilon)} + \frac{n_B-1}{m})^{-1}]^{1/\theta}$ entries. As there are n_B buckets, the total storage is at most $n_B[n_B(n_B-1)(\frac{1}{2K^\theta(1+1/\epsilon)} + \frac{n_B-1}{m})^{-1}]^{1/\theta}$ entries. The memory requirement is thus $O(\frac{n_B^{1+2/\theta} m^{1/\theta} K}{\epsilon^{1/\theta}})$. $\square$

The above theorem shows that the memory usage of our algorithm is very small. Table 1 (c) shows the number of entries for some particular values of n_B, ϵ and θ. We observe that more buckets, a smaller value of ϵ and a smaller value of θ require more storage space.

Theorem 5 *The memory usage used in our algorithm is bounded by*

$$O\left(n_B[n_B(n_B-1)(\frac{1}{2K^\theta(1+1/\epsilon)} + \frac{n_B-1}{m})^{-1}]^{1/\theta}\right) +$$

memory for the transactions stored in the most recent bucket

5 Empirical Study

The experiment was conducted with a Pentium IV 1.5GHz PC with 512MB memory on the Linux platform. We compare our algorithm with BOMO. BOMO mines the top K itemsets of at most size L in all transactions of in the sliding window. Thus, BOMO has to store all such transactions. Our algorithm and the BOMO algorithm are implemented in C/C++. The code of the BOMO algorithm is provided by [7]. We make use of the BOMO algorithm in our algorithm to obtain top K' itemsets in the bucket. Synthetic data sets are tested. We have conducted some experiments to study the memory usage, the amount of false positives and the execution time, by varying three factors in our algorithm - (1) L, the largest size of the itemsets to be mined and (2) Bucket Size.

We adopt the IBM synthetic data set[1]. The data set is generated with the following parameters (same as the parameters of [9]): 1,000 items, 3×10^6 transactions, 10 items per transaction on average, and 4 items per frequent itemset on average. We apply the same methodology as [9] to scramble the item-number mapping, in order to simulate the seasonal variations. For every five buckets, we permutate 200 items. In all experiments, we set $\theta = 1$. In most previous work, θ was set greater than 1. However, from the analysis of our algorithm, the worst case for the false positives and memory usage occurs when θ is the smallest. Hence we choose a small value for the experiments. For each measurement, we have repeated the experiments 5 times and taken the average.

The experimental results of memory usage with the study of the factors of L, bucket size s_B and K are shown in Table 2. The ratio measured is the ratio of the memory usage of BOMO over that of our algorithm. The ratio shows our algorithm uses much less memory.

The experimental results of the number of false positives over the number of itemsets returned are shown in Table 3. For the number of false positives found in the experiment, we observe that the numbers in the above tables are smaller than $K[(1+\epsilon)^{\frac{1}{\theta}} - 1]$ as predicted in Theorem 3. That means

Table 3.1

L \ l	1	2	3	4	5	6	7
1	0.00						
3	0.00	0.00	0.38				
5	0.00	0.00	0.38	0.35	0.74		
7	0.00	0.00	0.38	0.35	0.74	0.33	0.71

Table 3.2

$s_B \backslash l$	1	2	3	4	5	6
10K	0.00	0.00	0.31	0.29	0.67	0.23
20K	0.00	0.00	0.26	0.33	0.69	0.33
30K	0.00	0.00	0.29	0.33	0.73	0.29
40K	0.00	0.00	0.38	0.30	0.71	0.29
50K	0.00	0.00	0.38	0.35	0.74	0.33

Table 3.3

K \ l	1	2	3	4	5	6
1	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.09	0.09	0.33	0.23	0.63
20	0.00	0.00	0.38	0.35	0.74	0.33
50	0.00	0.00	0.35	0.38	0.39	0.82
100	0.00	0.00	0.27	0.45	0.84	0.64

Table 3. Synthetic Data Set: Fraction of False Positives (Default $\epsilon = 1$, $L = 6$, $K = 20$, s_B = 10K and n_B = 100)

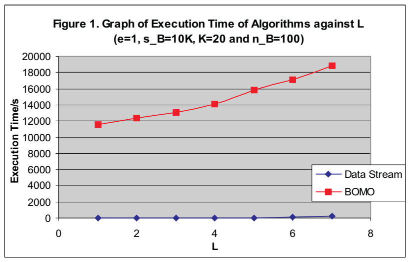

Figure 1. Graph of Execution Time of Algorithms against L (e=1, s_B=10K, K=20 and n_B=100)

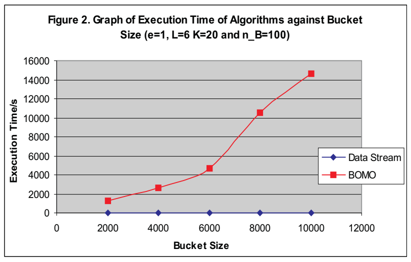

Figure 2. Graph of Execution Time of Algorithms against Bucket Size (e=1, L=6 K=20 and n_B=100)

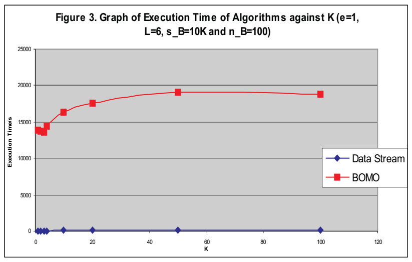

Figure 3. Graph of Execution Time of Algorithms against K (e=1, L=6, s_B=10K and n_B=100)

the experimental results give a verification of our analysis.

The experimental results of the execution time are shown in Figures 1, 2 and 3. We observe that our Data Stream algorithm runs much faster than BOMO algorithm. This is because the process of finding top K itemsets in our algorithm is more efficient due to a smaller data set in each bucket. Besides, the overhead of the combination of different results in different buckets is small. One more reason is that for every bucket to be processed, our data stream algorithm needs to manipulate one bucket only but BOMO requires to handle all buckets in the sliding window. Thus, our algorithm runs much faster.

Let us take a closer look at the false positives in our experiments. When we examine the frequencies of the false positives, they have actually a very small differences from the K-th frequent itemset in all cases. For example, in the experiment by varying L (the largest size of the itemsets to be mined), if $K = 20$, the actual count of the K-th frequent 4-itemset is 1733. Although there are 11 false positives in the output in Table 3, all their frequencies are greater than 1730, which means that the frequency difference is at most 3. The small frequency difference holds for all cases. The bound in Theorem 2 is only a worst-case upper bound. In practice, the count difference did not reach this bound.

6 Conclusion

In this paper, we address the problem of mining the K most frequent itemsets in a sliding window in a data stream. We propose an algorithm to estimate these K itemsets in the data stream. We prove that our algorithm gives no false negatives for any data distribution. It outputs at most $K(1+\epsilon)^{1/\theta}$ top frequent itemsets and stores a small number of entries for the Zipfian data distribution. We have conducted experiments to show that our algorithm can manipulate the data stream efficiently and both the memory usage and the execution time are many times smaller compared with a naive approach.

Acknowledgements We thank Y.L Cheung for providing us the coding of BOMO. This research was supported by the RGC Earmarked Research Grant of HKSAR CUHK 4179/01E, and the Innovation and Technology Fund (ITF) in the HKSAR [ITS/069/03].

References

[1] R. Agrawal. Ibm synthetic data generator, http://www.almaden.ibm.com/cs/quest/syndata.html.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, 2002.

[3] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In *SIGMOD*, 2003.

[4] C.-R. Lin C.-H. Lee and M.-S. Chen. Sliding-window filtering: An efficient algorithm for incremental mining. In *Intl. Conf. on Information and Knowledge Management*, 2001.

[5] J. H. Chang and W. S. Lee. Finding recent frequent itemsets adaptively over online data streams. In *SIGKDD*, 2003.

[6] Y.-L. Cheung and A. W.-C. Fu. An fp-tree approach for mining n-most interesting itemsets. In *SPIE Conference on Data Mining*, 2002.

[7] Y.-L. Cheung and A. W.-C. Fu. Mining frequent itemsets without support threshold: With and without item constraints. In *IEEE Trans. on Knowledge and Data Engineering*, to appear 2004.

[8] M. Datar, A. Gionis, P. Indyk, and R. Motwani. "maintaining stream statistics over sliding windows". In *SIAM Journal on Computing*, 2002.

[9] C. Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu. Mining frequent patterns in data streams at multiple time granularities. In *Next Generation Data Mining*, 2003.

[10] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD*, 1998.

[11] A. Gilbert, Y. Kotidis, and S. Muthukrishnan. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB*, 2001.

[12] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, 2002.