Efficient Data-Reduction Methods for On-Line Association Rule Discovery

Herve Bronnimann[a], Bin Chen[b], Manoranjan Dash[c], Peter Haas[d], Yi Qiao[c], Peter Scheuermann[c]

[a]Computer and Information Science, Polytechnic University, 6 Metro Tech Center, Brooklyn, NY 11201

[b]Exelixis Inc., 170 Harbor Way, S. San Francisco, CA 94083

[c]Department of Electrical and Computer Engineering, Northwestern University, 2145 Sheridan Road, Evanston, IL 60208

[d]IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120

Contact Author (e-mail): peters@ece.northwestern.edu

Abstract

Classical data mining algorithms that require one or more computationally intensive passes over the entire database can be prohibitively slow. One effective method for dealing with this ever-worsening scalability problem is to run the algorithms on a small sample of the data. We present and empirically compare two data-reduction algorithms for producing such a sample; these algorithms, called FAST *and* EA*, are tailored to "count" data applications such as association-rule mining. The algorithms are similar in that both attempt to produce a sample whose "distance" — appropriately defined — from the complete database is minimal. They differ greatly, however, in the way that they greedily search through the exponential number of possible samples.* FAST*, originally presented in [8], uses random sampling together with trimming of "outlier" transactions. On the other hand, the* EA *algorithm, introduced in the current paper, repeatedly and deterministically halves the data to obtain the final sample. Unlike* FAST*, the* EA *algorithm provides a guaranteed level of accuracy. Our experiments show that* EA *is more expensive to run than* FAST*, but yields more accurate results for a given sample size. Thus, depending on the specific problem under consideration, the user can trade off speed and accuracy by selecting the appropriate method. We conclude by showing how the* EA *data-reduction approach can potentially be adapted to provide data-reduction schemes for streaming data systems. The proposed schemes favor recent data while still retaining partial information about all of the data seen so far.*

Keywords: Sampling, Distance function, Association mining, Streaming data, Data reduction

1 Introduction

The volume of electronically accessible data in warehouses and on the Internet is growing faster than the speedup in processing times predicted by Moore's Law [14]. Consequently, classical data mining algorithms that require one or more computationally intensive passes over the entire database are becoming prohibitively slow, and this problem will only become worse in the future. The scalability of mining algorithms has therefore become a major research topic. One approach to the scalability problem is to run mining algorithms on a small subset of the data, sometimes called a *synopsis* or *sketch*. This strategy can yield useful approximate results in a fraction of the time required to compute the exact solution, thereby speeding up the mining process by orders of magnitude.

A number of synopses have been proposed in the literature but many of them require one or more expensive passes over all of the data. Using a sample of the data as the synopsis is a popular technique that can scale well as the data grows. Another nice property of sampling methods is that it is often possible to explicitly trade off processing speed and accuracy of results. Recent work in the area of approximate aggregation processing [1, 13] shows that the benefits of sampling are most fully realized when the sampling technique is tailored to the specific problem at hand. In this spirit we investigate sampling methods that are designed to work with mining algorithms for "count" datasets, that is, datasets in which there is a base set of "items" and each data element is a vector of item counts — here "items" may correspond to physical items, responses on a survey, income levels, and so forth. As a first step, we present and compare two novel data-reduction methods in the

context of the most well-studied mining problem defined on count data: the discovery of association rules in large transaction databases.

The two algorithms that we consider are similar in that both attempt to produce a sample whose "distance" from the complete database is minimal. The algorithms differ greatly, however, in the way they greedily search through the exponential number of possible samples. As discussed below, the choice of which algorithm to use depends on the desired tradeoff between speed of computation and accuracy of results, the amount of available memory, and other factors. The first algorithm, FAST (*F*inding *A*ssociation rules from *S*ampled *T*ransactions) was recently presented in [8]. FAST starts with a large random sample and trims away "outlier" transactions to produce a small final sample that more accurately reflects the entire database. The second algorithm, EA (*E*psilon *A*pproximation), is new. The EA algorithm repeatedly and deterministically halves the data to obtain the final sample. Unlike FAST, the EA algorithm provides a guaranteed upper bound on the distance between the sample and the entire database.

After presenting and comparing the FAST and EA algorithms, we then show how the EA approach can potentially be adapted to provide data-reduction schemes for streaming data systems. The proposed schemes favor recent data while still retaining partial information about all of the data seen so far.

The paper is organized as follows. The FAST and EA algorithms are described and compared in Section 2, and guidelines are given for their usage. In Section 3 we discuss a possible approach for applying the EA data-reduction method to streaming data. Section 4 contains our conclusions and directions for future work.

2 Sampling-Based Association Rule Mining

Agrawal, et al. [3] proposed association rules as a means of identifying relationships among sets of items, which can be used to evaluate business trends, identify purchasing patterns, and classify customer groups. Two measures, called *support* and *confidence*, are introduced in [3] in order to quantify the significance of an association rule. The mining of association rules from a set of transactions is the process of identifying all rules having support and confidence greater than specified minimum levels; such rules are said to have "minimum confidence and support." We focus on the problem of finding the "frequent" itemsets, i.e., the itemsets having minimum support, because this operation is by far the most expensive phase of the mining process. We assume that the reader is familiar with the basic Apriori algorithm, introduced in [3], for identifying frequent itemsets. A variety of modifications have been proposed to reduce the computational burden—see, for example, [2, 12] and references therein—but with few exceptions all current algorithms require at least one expensive pass over the data.

Throughout this section, we assume that the contents of the transactional database do not change during the mining process. We also assume that the database is very large. Denote by D the database of interest, by S a simple random sample drawn without replacement from D, and I the set of all items that appear in D. Also denote by $\mathcal{I}(D)$ the collection of itemsets that appear in D; a set of items A is an element of $\mathcal{I}(D)$ if and only if the items in A appear jointly in at least one transaction $t \in D$. If A contains exactly k (≥ 1) elements, then A is sometimes called a k-itemset. The collection $\mathcal{I}(S)$ denotes the itemsets that appear in S; of course, $\mathcal{I}(S) \subseteq \mathcal{I}(D)$. For $k \geq 1$ we denote by $\mathcal{I}_k(D)$ and $\mathcal{I}_k(S)$ the collection of k-itemsets in D and S, respectively. Similarly, $L(D)$ and $L(S)$ denote the frequent itemsets in D and S, and $L_k(D)$ and $L_k(S)$ the collection of frequent k-itemsets in D and S, respectively. For an itemset $A \subseteq I$ and a set of transactions T, let $n(A;T)$ be the number of transactions in T that contain A and let $|T|$ be the total number of transactions in T. Then the support of A in D and in S is given by $f(A;D) = n(A;D)/|D|$ and $f(A;S) = n(A;S)/|S|$, respectively.

2.1 FAST

Given a specified minimum support p and confidence c, the FAST algorithm for data reduction proceeds as follows:

1. Obtain a simple random sample S from D.
2. Compute $f(A;S)$ for each 1-itemset $A \in \mathcal{I}_1(S)$.
3. Using the supports computed in Step 2, obtain the final small sample S_0 from S.
4. Run a standard association-rule mining algorithm against S_0 – with minimum support p and confidence c — to obtain the final set of association rules.

Steps 1, 2, and 4 are straightforward. The drawing of a sample in Step 1 can be performed with a computational cost of $O(|S|)$ and a memory cost of $O(|S|)$, i.e., sample size $|S|$ is determined by the available memory. Step 2 requires at most $O(T_{\max} \cdot |S|)$, where $T_{\max}$ stands for the maximal transaction length. From a computational point of view, because the cost

of Step 2 is relatively low, the sample S can be relatively large, thereby helping to ensure that the estimated supports are accurate. Step 4 computes the large itemsets using a standard association rule mining algorithm such as Apriori [4].

The crux of the algorithm is Step 3. Two approaches (trimming and growing) for computing the final small sample S_0 from S are given in [8]. In this paper we discuss only the trimming method, which removes the "outlier" transactions from the sample S to obtain S_0. In this context an outlier is defined as a transaction whose removal from the sample maximally reduces (or minimally increases) the difference between the supports of the 1-itemsets in the sample and the corresponding supports in the database D. Since the supports of the 1-itemsets in D are unknown, we estimate them by the corresponding supports in S as computed in Step 2. To make the notion of difference between 1-itemset supports precise we define a distance function, based on the symmetric set difference, by setting

$$Dist_1(S_0, S) = \frac{|L_1(S) - L_1(S_0)| + |L_1(S_0) - L_1(S)|}{|L_1(S_0)| + |L_1(S)|} \tag{1}$$

for each subset $S_0 \subseteq S$ — in accordance with our previous notation, where $L_1(S_0)$ and $L_1(S)$ denote the sets of frequent 1-itemsets in S_0 and S. Observe that $Dist_1$ takes values in $[0, 1]$, and that it is sensitive to both false frequent 1-itemsets and missed frequent 1-itemsets. Our goal is to trim away transactions from S so that the distance from the final sample S_0 to the initial sample S is as small as possible. Other definitions of distance are possible, for example based on the L_2 metric between the frequency vectors:

$$Dist_2(S_0, S) = \sum_{A \in \mathcal{I}_1(S)} \big(f(A; S_0) - f(A; S)\big)^2. \tag{2}$$

The basic FAST-TRIM algorithm is as follows:

obtain a simple random sample S from D;
compute $f(A; S)$ for each item A in S;
set $S_0 := S$;
while ($|S_0| > n$) { `//trimming phase`
  divide S_0 into disjoint groups of $\min(k, |S_0|)$ transactions each;

  for each group G {
    compute $f(A; S_0)$ for each item A in S_0;
    set $S_0 = S_0 - \{t^*\}$, where $Dist(S_0 - \{t^*\}, S) = \min_{t \in G} Dist(S_0 - \{t\}, S)$;
  }
}
run a standard association-rule algorithm against S_0 to obtain the final set of association rules;

By choosing a value of k between 1 and $|S|$, the user can strike a balance between ineffective but cheap "oblivious" trimming and very effective but very expensive "pure greedy" trimming. For more details on different aspects of FAST such as distance functions, variants of FAST, stopping criteria, detailed algorithm, and complexity analyses see [8].

2.2 Epsilon Approximation

The epsilon approximation method is similar to FAST in that it tries to find a small subset having 1-itemset supports that are close to those in the entire database. The "discrepancy" of any subset S_0 of a superset S (that is, the distance between S_0 and S with respect to the 1-itemset frequencies) is computed as the L_∞ distance between the frequency vectors:

$$Dist_\infty(S_0, S) = \max_{A \in \mathcal{I}_1(S)} |f(A; S_0) - f(A; S)| \tag{3}$$

where A is an 1-itemset. The sample S_0 is called an ε-approximation of S if its discrepancy is bounded by ε. Obviously there is a trade-off between the size of the sample and ε: the smaller the sample, the larger ε. For literature on ε-approximations, see e.g. the book by Chazelle [7, Ch.4].

The halving method. We now explain how we compute the approximations. At the heart of the epsilon approximation method is a method that computes a subset S_0 of approximately half the size. We use a variant due to Chazelle and Matoušek of the hyperbolic cosine method (see e.g. Alon and Spencer's book [5, Ch.15] or [7, Ch.1]). To start with, S is D, the entire database. The method processes the transactions from $j = 1$ up to $n = |S|$, and for each one sequentially makes the decision to color that transaction blue or red. At the beginning all transactions are grey (i.e., uncolored). At the end, all the transactions will be colored, the red transactions will form a set S_r and the blue a set S_b, and both sets will have approximately the same size. Choosing either one as S_0 will do. Repeated iterations of this halving procedure will result in the smallest subset S_0 for which $Dist_\infty(S_0, S) \leq \varepsilon$. Thus, while FAST will terminate when it reaches a sample of a desired size, the epsilon approximation will terminate when it detects that further subdivision will cause the distance to exceed the upper bound ε.

Specifically, let $m = |\mathcal{I}_1(S)|$ be the number of items. For each item A_i we define a penalty Q_i as follows. Denote by S^i the set of all transactions that contain item A_i and suppose that we have colored the first j transactions. Then the penalty Q_i is given by

$$Q_i = Q_i^{(j)} = (1+\delta_i)^{r_i}(1-\delta_i)^{b_i} + (1-\delta_i)^{r_i}(1+\delta_i)^{b_i} \tag{4}$$

where $r_i = r_i^{(j)}$ and $b_i = b_i^{(j)}$ are the numbers of red and blue transactions in S^i. Initially (for $j = 0$, for each i, $r_i = b_i = 0$ and so $Q_i = 2$. In order to decide how to color transactions, we introduce the global penalty $Q = \sum_{1 \leq i \leq m} Q_i$. Assuming the colors of the first j transactions have been chosen, there are two choices for the $(j+1)^{th}$ transaction. Coloring it red yields $Q_i^{(j\|r)} = (1+\delta_i)^{r_i+1}(1-\delta_i)^{b_i} + (1-\delta_i)^{r_i+1}(1+\delta_i)^{b_i}$ while coloring it blue yields $Q_i^{(j\|b)} = (1+\delta_i)^{r_i}(1-\delta_i)^{b_i+1} + (1-\delta_i)^{r_i}(1+\delta_i)^{b_i+1}$. It is readily verified that $Q_i^{(j)} = \frac{1}{2}(Q_i^{(j\|r)} + Q_i^{(j\|b)})$. By linearity, summing over all items we get $Q^{(j)} = \frac{1}{2}(Q^{(j\|r)} + Q^{(j\|b)})$. Hence, there is one choice of color c for $(j+1)^{th}$ transaction such that $Q^{(j\|c)} \leq Q^j$, and this is the color chosen for the transaction. At the end of the coloring, we have $Q^{final} \leq Q^{init} = 2m$. Since all the Q_i's are positive, this implies that, for each item, we have also $Q_i^{final} \leq 2m$. If $r_i = r_i^{(n)}$ and $b_i = b_i^{(n)}$ denote the final numbers of red and blue transactions in S^i, we have $r_i + b_i = |S^i|$. Hence

$$\begin{aligned} 2m &\geq (1+\delta_i)^{r_i}(1-\delta_i)^{b_i} &(5)\\ &\geq (1+\delta_i)^{r_i-b_i}(1-\delta_i^2)^{|S^i|} &(6)\end{aligned}$$

and the same bound holds when exchanging r_i and b_i, from which it follows that

$$|r_i - b_i| \leq \frac{1}{\ln(1+\delta_i)} \left(\ln(2m) - |S^i| \ln(1-\delta_i^2)\right). \tag{7}$$

Choosing a good value of δ_i can now be done to optimize this quantity. A reasonable choice in theory is to balance both terms of the inner sum, leading to

$$\delta_i := \sqrt{1 - \exp\left(-\frac{\ln(2m)}{|S^i|}\right)} \tag{8}$$

Since $x = \ln(2m)/|S^i|$ is typically very small whenever $|S^i|$ is reasonably large, applying the equivalents $1 - \exp(-x) \approx x$ into (8) and $\ln(1+x) \approx x$ into (7) implies that $|r_i - b_i| = O(\sqrt{|S^i| \log(2m)})$. Note that if $|S^i|$ is not too small, the latter quantity is much less than $|S^i|$. Hence, there are about as many red as blue transactions in S^i for each item. Since $r_i + b_i = |S^i|$, this also means that

$$|r_i - |S^i|/2| = O(\sqrt{|S^i| \log(2m)}). \tag{9}$$

In order to guarantee that there are as many transactions in S_r as in S_b, we can add a (fictitious) item A_0 that is contained in all transactions. In that case, $|S_r| - |S_b|$ is also $O(\sqrt{n \log(2m)})$, and this implies that $|S_r| = n/2 + O(\sqrt{n \log(2m)})$. Dividing (9) by S_r (or by $n/2$), we get that for each i,

$$|f(A_i; S_r) - f(A_i; S)| \leq \varepsilon(n, m) = O(\sqrt{\log(2m)/n}).$$

In practice, the halving method will work with any choice of δ_i, but the bounds on $f(A_i; S_r) - f(A_i; S)$ will not necessarily be guaranteed. In the implementation, we have found that setting

$$\delta_i := \sqrt{1 - \exp\left(-\frac{\ln(2m)}{n}\right)}$$

proved to be very effective. The advantage is that if the defining parameters of the database, i.e., the number n of transactions and number m of items, are already known then the halving method requires a single scan of the database.

The implementation works as follows: first it initializes all the r_i, b_i, δ_i and Q_i as indicated. Then it performs the scan, for each transaction deciding whether to color it red or blue as given by the penalties. In order to update the penalties, it is better to store both terms of Q_i separately into two terms $Q_{i,1}$ and $Q_{i,2}$. The penalties are then updated according to the formula and the color chosen for the transaction. The red transactions are added to the sample, and the blue are forgotten. The memory required by the halving method is proportional only to the number $m = |\mathcal{I}_1(S)|$ of 1-itemsets, in order to store the penalties.

A further improvement in performance is obtained if we realize that only the penalties for the items contained in the current transaction need to be recomputed, not all m of them. Hence the halving method processes a transaction in time proportional to the number of items that it contains, and the entire halving takes time proportional to the size of the database (number of transactions), which is $O(T_{max} \cdot |S|)$, not the much larger size $|\mathcal{I}_1(S)||S|$.

As sketched above, the implementation of the halving method works as follows, assuming we have knowledge of n and m:

EA-HALVING(S, n, m)
  set $\delta_i = \sqrt{1 - \exp(-\frac{\ln(2m)}{n})}$, and $Q_{i,1} = Q_{i,2} = 1$ for each $i = 1$ to m;
  for each transaction j do {
    for each item i contained in j {
      compute $Q_{i,1}^{(r)} = (1 + \delta_i)Q_{i,1}, Q_{i,2}^{(r)} = (1 - \delta_i)Q_{i,2}$;
      compute $Q_{i,1}^{(b)} = (1 - \delta_i)Q_{i,1}, Q_{i,2}^{(b)} = (1 + \delta_i)Q_{i,2}$;
    }
    let $Q^{(r)} = \sum_i Q_{i,1}^{(r)} + Q_{i,2}^{(r)}$ and $Q^{(b)} = \sum_i Q_{i,1}^{(b)} + Q_{i,2}^{(b)}$
      with the sum taken over those items i contained in j;
    if $Q^{(r)} < Q^{(b)}$ then
      color j red and update $Q_{i,1} = Q_{i,1}^{(r)}, Q_{i,2} = Q_{i,2}^{(r)}$;
    else
      color j blue and update $Q_{i,1} = Q_{i,1}^{(b)}, Q_{i,2} = Q_{i,2}^{(b)}$;
  }
  return $S_0 = S_r$, the set of red transactions;

Computing the approximation. Having fixed a value of ε, we compute an ε-approximation as follows. Note that the halving method computes an $\varepsilon(n, m)$-approximation of size $n/2$, where $\varepsilon(n, m) = O(\sqrt{\ln(2m)/n})$. (Note: $O(\sqrt{\ln(2m)/n})$ is a very small value when m is polynomially bounded in n and n is large). There are two structural properties that we can use to reduce the size of approximations [7, Lem.4.2]: one states that if S_1 is an ε_1 approximation of S and S_2 an ε_2 approximation of S_1, then S_2 is an $(\varepsilon_1 + \varepsilon_2)$-approximation of S. Thus approximations can be *composed* by simply adding the discrepancies.

The repeated halving method starts with S, and applies one round of halving (as described in EA-HALVING) to get S_1, then another round of halving to S_1 to get S_2, etc. The sizes $n_1 \geq n_2 \geq \ldots$ of these samples decrease roughly geometrically by a factor of two (specifically $n_1 \leq n(\frac{1}{2} + \varepsilon(n, m))$, and $n_{i+1} \leq n_i(\frac{1}{2} + \varepsilon(n_i, m))$). Note that by the above observation, S_t is an ε_t-approximation, where $\varepsilon_t = \sum_{k \leq t} \varepsilon(n_k, m)$. We stop the repeated halving for the maximum t such that $\varepsilon_t \leq \varepsilon$.

Implemented naively, the repeated halving method would require t passes over the database. However, observe that the halving process is inherently sequential in deciding the color of a transaction, and that either color may be chosen as the sample. Say we always choose the red transactions as our sample. In a single pass, we may store all the penalties of each halving method and proceed for each transaction as follows: based on the penalties of the first halving method, we decide whether to color that transaction red or not in the first sample. Should this transaction be red, we again compute the penalties of the second halving method, etc. until either the transaction is colored blue in a sample, or it belongs to the sample S_t. (Since the samples are expected to decrease by half at each level, setting $t = \log n$ will do.) Thus all the repeated halving methods can be run simultaneously, in a single pass. The memory required by this algorithm is thus $O(m \log n) = O(|\mathcal{I}_1(s)| \log |S|)$.

2.3 Comparison of FAST and EA

In this section we present an experimental comparison between FAST and EA. Earlier results reported in [8] showed that FAST achieves the same or higher accuracy (between 90-95%) using a final sample that is only a small fraction (15 -35%) of a simple random sample. In addition, its efficiency gain resulted in a speed-up factor of 10 over Toivonen's sampling algorithm that requires a complete expensive database scan. We note here that although the EA and Toivonen algorithms both require a complete database scan, the EA algorithm requires much less processing per transaction. Indeed, the EA algorithm examines each transaction only so far as to decide how to color it, whereas Toivonen's algorithm uses each scanned transaction to update count statistics for a large collection of candidate itemsets.

We used both synthetic and real-world databases in our experiments, but due to lack of space we restrict ourselves here to reporting the results for the synthetic database and the trimming version of FAST. The synthetic database was generated using code from the IBM QUEST project [4]. The parameter settings for synthetic data generation are similar to those in [4]: the total numbers of items was set to 100, the number of transactions was set to 100,000, the number of maximal potentially frequent itemsets was set to 2000, the average length of transactions is 10, and the average length of maximal potentially large itemsets was 4. We used a minimum support value of 0.77%, at which level there are a reasonable number of large itemsets, and the length of the maximal large itemset is 6.

In addition to EA and FAST, we also performed experiments with simple random sampling (denoted SRS in the figures) in order to relate to results previously reported in [8]. In order to make a fair comparison between the three algorithms we used Apriori in all cases to compute the large itemsets and used a common set of functions for performing I/O. We used a publicly available implementation of Apriori written by Christian Borgelt.[1] This implementation, which uses prefix trees, is a reasonably fast and has been incorporated into a commercial data mining package. FAST was implemented using distance functions $Dist_1$ and $Dist_2$. For phase 1 of FAST, a sample size of 30% is chosen by random sampling. For the parameter k, the group size in the FAST-TRIM algorithm, we chose a value of 10 since this was shown to be a reasonable choice in [8]. As EA cannot achieve all the sample sizes (because the halving process has a certain granularity), in each iteration we first ran EA with a given ϵ value, and then used the obtained sample size to run FAST and SRS. EA is not independent of the input sequence, so to account for any difference due to the particular input sequence the results of EA are computed as an average over 50 runs, each one corresponding to a different shuffle of the input. In order to smooth out any difference, the results of FAST and SRS are also averaged over 50 runs, each time choosing a different simple random sample from the database.

Our primary metrics used for the evaluation are accuracy and execution time. Accuracy is defined as follows:

$$accuracy = 1 - \frac{|L(D) - L(S)| + |L(S) - L(D)|}{|L(S) + L(D)|} \tag{10}$$

where, as before, $L(D)$ and $L(S)$ denote the frequent itemsets from the database D and the sample S respectively. Notice that this metric is similar to $Dist_1$ with the difference that the accuracy is based on the set difference between all frequent itemsets generated from D and S, while $Dist_1$ is based only on frequent 1-itemsets. The execution time is the total time that includes the time required for I/O, finding the final sample and running Apriori.

Results All the experiments were performed on a SUNW Sparc Ultra workstation with a processor speed of 333 MHz and 256MB memory. The sampling ratios output by EA were 0.76%, 1.51%, 3.02%, 6.04%, 12.4%, and 24.9%. Figure 1 (a) displays the accuracy of FAST-TRIM, EA and SRS on the synthetic database as a function of the sampling ratio, while Figure 1 (b) depicts the execution time of the above mentioned algorithms *vs*. the sampling ratio.

From Figure 1 (a) we observe that EA achieves very good accuracy even for small sample sizes. Thus, even for a sample size of 1.51% it could achieve close to 89% accuracy, while FAST-TRIM with distance function $Dist_2$ achieves only 82% accuracy. For larger sample sizes, the differences in accuracy between EA and FAST-TRIM are smaller. For example, for a sample size of 12.4%, EA achieves close to 99% accuracy, versus 94.5% for FAST-TRIM with $Dist_2$. On the other hand, as shown in Figure 1 (b), EA is more time-consuming than FAST. When using a final sample size of 12.4%, for example, FAST-TRIM has an execution time of about 50% larger than SRS, while EA's time is approximately 4 times that of SRS. The trimming operation performed in FAST is substantially faster than the repeated halving method. Of course, the performance gains for either data-reduction method are more pronounced if the cost of producing the sample is amortized over multiple mining tasks.

[1] http://fuzzy.cs.uni-magdeburg.de/~borgelt/software.html

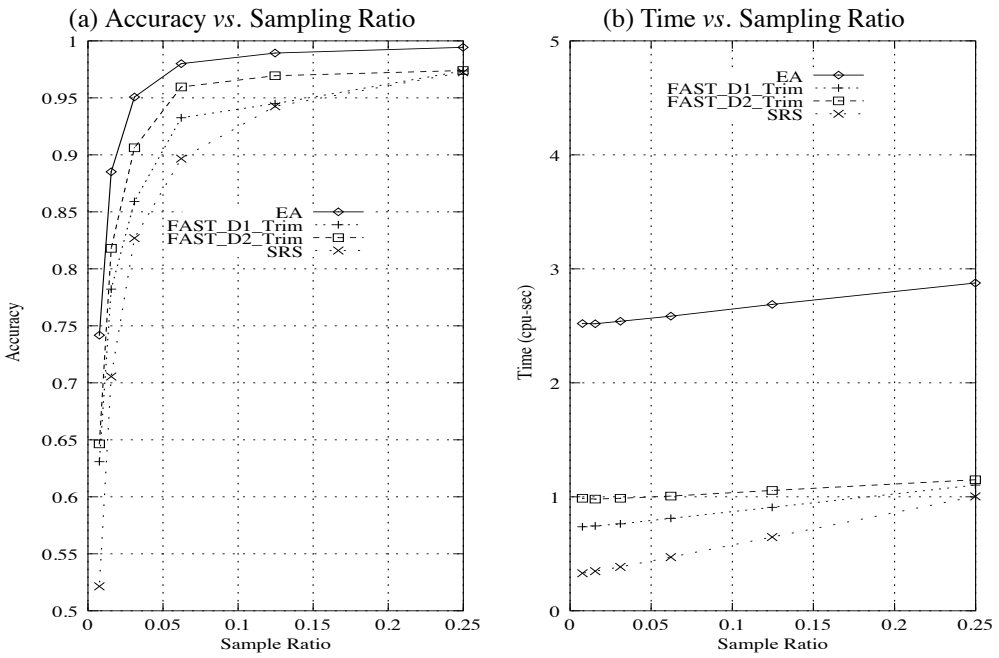

Figure 1. Results for synthetic dataset

3 Data Stream Reduction

3.1 Streaming Data Analysis

Contrary to finite stored databases, streaming databases grow continuously, usually rapidly, and potentially without bound. Examples include stock tickers, network traffic monitors, point-of-sale systems, and phone conversation wiretaps (for intelligence analysis). Unlike the data processing methods for stored datasets, methods for analyzing streaming data require timely response and the use of limited memory to capture the statistics of interest. In addition, network management and stock analysis demand the real-time processing of the most recent information.

Manku and Motwani [13] investigated the applicability of a number of frequency counting applications on streaming data, including Iceberg Queries [10], Iceberg Cubes [6, 11], Apriori [4], and network flow identification [9]. They then proposed Sticky Sampling and Lossy Counting algorithms to identify frequent singleton items over a stream.

Sticky Sampling relies on a fixed-size buffer and various sampling rates to collect and estimate the counts of incoming items. It samples the first t incoming items at rate $r = 1$, selecting one item for every item seen; and at rate $r = 2$, selecting one item for every two items seen, for the next $2t$ items; and $r = 4$ for the next $4t$ items; and so on, where t is a predefined value based on frequency threshold, user specified error, and the probability of failure. The approach is equivalent to randomly selecting the same number of items from an enlarging moving window which keeps doubling itself to include twice as many items as before. While Sticky Sampling could accurately maintain the statistics of items over a stable data stream in which patterns change slowly, it fails to address the needs of important applications, such as network traffic control and pricing, that require the information of the entire stream but with emphasis on the most recent data. In fact, the enlarging window and the increasing sampling rate make the statistics at hand less and less sensitive to the changes in recent data. We will soon introduce an extension of EA that can keep a representative sample of the stream, while reflecting the most recent trends in the data.

Manku and Motwani also proposed a deterministic algorithm, Lossy Counting Algorithm, to find frequent single items from streaming data. Lossy Counting stores the observed frequency and the estimated maximal frequency error for each frequent, or potentially frequent, item in a series of conceptual buckets. It keeps adding new items to and removing existing less frequent items from the buckets. Although this algorithm has worse worst-case space complexities than Sticky Sampling, experiments showed that the former performed much better than the latter one when streams had random arrival distributions.

The authors then extended Lossy Counting Algorithm to identify frequent itemsets. They virtually divided a stream into chunks based on the order of the arrival data and then identified the frequent itemsets from each chunk. Similar to Sticky Sampling, Lossy Counting and its extension could be effective when the goal is to find the frequent itemsets over a stable data stream, but may not be effective with drastically changing data. Moreover, the computation of frequent itemsets from each chunk in the extension of Lossy Counting could be prohibitively expensive for high speed data streams, such as network traffic and words in phone conversations.

3.2 DSR: Data Stream Reduction

In this section, we propose an EA-based algorithm, DSR (*D*ata *S*tream *R*eduction), to sample data streams. Our goal is to generate a representative sample of a given data stream such that the sample carries information about the entire stream while favoring the recent data. Unlike static databases, a data stream, D_S, can be constantly changing. Therefore, its sample, S_S, should also be regularly adjusted to reflect the stream. In this sense, we define *a representative sample of a data stream* as a changing subset of the data stream, which can reflect the statistics, or item (joint) distributions of the data stream at any moment. We decide to maintain a dynamically changed sample of a data stream instead of keeping only count statistics because doing so would offer more flexibilities to users so that they can decide what information should be summarized from the sample, such as frequent itemsets in the sample.

We rely on a relatively simple model to discuss the data reduction problem on streaming data, where each element of the data stream is a transaction consisting of a set of items (0-1 problem). The resulting algorithm, DSR, should be applicable to other more complicated data streams. Consider an ideal scenario: Suppose we want to generate an N_S-element sample, S_S, of a data stream, D_S, where S_S puts more weight on recent data. A straightforward solution to construct such a biased sample is to apply different sampling rates to different parts of D_S, with higher selection rates on recent data. In order to find out a way to construct S_S, let us temporarily "freeze" the data stream after we have observed $m_s \cdot \frac{N_S}{2}$ transactions. We assign the transactions into m_s conceptual buckets based on the order the transactions are received, with each bucket holding $\frac{N_S}{2}$ transactions. The buckets are labeled as $1, 2, ..., m_s$, with bucket 1 containing the oldest transactions and bucket m_s holding the most recent data. Compared with simple random sampling, EA can normally be utilized to get a good, or relatively good, representative subset of the original data regardless whether or not the error bound, ε, is exceeded. We apply EA to half $m_s - k$ times on bucket k ($k = 1, 2, ..., m_s$). Denote the resulting subset of bucket k by s_k. Observe that s_k is a good, or relatively good, representative of bucket k, therefore, the union of the resulting subsets, $\cup_k s_k$, can be viewed as a good representative of the data stream, which contains approximately N_S transactions in total and emphasizes recent data. Contrary to Sticky Sampling that samples less and less frequently, our approach to get the reduction of the "frozen" stream samples more and more frequently along the time dimension until it selects all $\frac{N_S}{2}$ transactions from the most recent bucket.

In reality, however, it can be prohibitively expensive to map the transactions into conceptual buckets and compute the representative subset of each bucket whenever a new transaction arrives. The idea behind DSR is to simulate the ideal scenario while avoiding frequent halving. To this end, a working buffer that can hold N_S transactions is utilized to receive and generate the reduced representative of the data stream. The buffer size N_S should be as large as possible. Initially, the buffer is empty. Once it is filled with N_S transactions, if new data come, the buffer is halved by EA and new data are inserted to the buffer. Observe that the older the data in the buffer, the more halving by EA they have experienced. Whenever a user requests a representative of the stream, all the transactions in the buffer are returned to the user.

3.3 Discussion of DSR

The advantages of DSR include:

1. Representative tuples are selected from a data stream at different rates that favor recent data. Therefore DSR is more sensitive to recent changes in the stream than Sticky Sampling and Lossy Counting.

2. Unlike traditional tools on streaming data, DSR attempts to generate a representative sample, instead of collecting statistics such as counts. In this way, we offer more flexibility since users can decide what operations they would like to perform on the representative subset.

3. Since each halving operation by EA is relatively cost-effective, when applying DSR to frequent itemset identification, we don't have to periodically generate and keep the frequent itemsets from the buffer, and thus avoid the expensive frequent itemset identifications by Lossy Counting based approach.

A potential problem with DSR concerns the stability of analytical results computed from the sample. After the working buffer is halved, the number of transactions in the buffer will be changed from N_S to $\frac{N_S}{2}$. If two users request data immediately prior to and after the halving operation, respectively, these users could receive data that vary dramatically and hence get very different analytical results even when the actual data stream remains stable. We can use a variant of DSR, or continuous DSR, to solve this problem. Unlike DSR, where the buffer is halved when it is filled, continuous DSR halves a smaller chunk of the buffer at a time. It works in the following manner: After the buffer is filled, transactions are divided into $\frac{N_S}{2n_s}$ chunks in the increasing order that they are received. Each chunk contains $2n_s$ transactions. After the working buffer is filled and n_s new transactions arrive, where $n_s << N_S$ and is a predefined fixed number, EA is called to half the first chunk that contains the oldest $2n_s$ transactions. The n_s new transactions then replace the n_s old transactions that are evicted. When another n_s new transactions come, the second chunk is halved and the new transactions replace the newly evicted transactions. The procedure continues until all $\frac{N_S}{2n_s}$ chunks are halved. At this time, $\frac{N_S}{2}$ of the transactions in the buffer have been replaced by new transactions. Then all transactions in the buffer are re-assigned to $\frac{N_S}{2n_s}$ chunks, each of which holding $2n_s$ transactions. The cycle continues so that except the initial warm-up period, the buffer is always full. Because $n_s \ll N_S$, no matter how close in time two users request the data, their results would not be drastically affected by the fluctuations caused by halving operations.

There are some open issues surrounding the choice of discrepancy function to use in the DSR context. On a static database, our experiments have indicated that in many cases the discrepancy function based on single item frequencies could lead to acceptable error on the frequencies of higher level itemsets. However, in case of streaming data, the error introduced by EA will likely accumulate and therefore the ε-bound could easily be exceeded.

It is also not entirely clear how to evaluate the goodness of a representative subset obtained by DSR. Recall that our goal is to favor recent data. More generally, we might want to put weights on the data. The measurement of the error associated with the reduced data stream should reflect the weighting scheme. We are currently investigating appropriate measurements for such weighted stream reduction.

4 Conclusion and Future Directions

In this paper we proposed and compared two algorithms for sampling-based association mining. Analysis of the comparison results led to insights about when each algorithm is appropriate. We plan to perform more exhaustive experiments involving additional data sets and other variants of both FAST and EA.

We are currently working at modifying EA so that it can keep in memory at any time only the transactions that belong to the final sample. Such a modification would make EA an even more attractive alternative for extremely large databases, where we can afford to keep in memory only a small sample of the database. We are also exploring other modifications that would give the user finer control over the final sample size.

We also proposed some potential extensions of the EA algorithm to permit maintenance of a sample of streaming data, and are currently investigating the details of such extensions.

Overall, the FAST and EA data-reduction methods provide powerful, complementary tools for scaling existing mining algorithms to massive databases. Our results provide the user with new options for trading off processing speed and accuracy of results.

References

[1] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *Proceedings of ACM SIGMOD International Conference on management of Data*, 2000.

[2] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000.

[3] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1993.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of International Conference on Very Large Databases (VLDB)*, 1994.

[5] N. Alon and J. H. Spencer. *The probabilistic method*. Wiley Interscience, New York, 1992.

[6] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceburg cubes. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1999.

[7] B. Chazelle. *The discrepancy method*. Cambridge University Press, Cambridge, United Kingdom, 2000.

[8] B. Chen, P. Haas, and P. Scheuermann. A new two-phase sampling based algorithm for discovering association rules. In *Proceedings of ACM SIGKDD International Conference on Knoledge Discovery and Data Mining*, 2002.

[9] C. Estan and G. Verghese. New directions in traffic measurement and accounting. In *ACM SIGCOMM Internet Measurement Workshop*, 2001.
[10] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. Ullman. Computing iceburg queries efficently. In *Proceedings of International Conference on Very Large Databases (VLDB)*, 1998.
[11] J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of iceburg cubes with complex measures. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2001.
[12] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2000.
[13] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of International Conference on Very Large Databases (VLDB)*, 2002.
[14] R. Winter and K. Auerbach. The big time: 1998 winter VLDB survey. *Database Programming Design*, August 1998.