On Biased Reservoir Sampling in the Presence of Stream Evolution.

Conference Paper · January 2006

Source: DBLP

CITATIONS
100

READS
479

1 author:

Charu C. Aggarwal
IBM
486 PUBLICATIONS **23,068** CITATIONS

SEE PROFILE

On Biased Reservoir Sampling in the Presence of Stream Evolution

Charu C. Aggarwal
IBM T. J. Watson Research Center
19 Skyline Drive
Hawhorne, NY 10532, USA
charu@us.ibm.com

ABSTRACT

The method of reservoir based sampling is often used to pick an unbiased sample from a data stream. A large portion of the unbiased sample may become less relevant over time because of evolution. An analytical or mining task (eg. query estimation) which is specific to only the sample points from a recent time-horizon may provide a very inaccurate result. This is because the size of the relevant sample reduces with the horizon itself. On the other hand, this is precisely the most important case for data stream algorithms, since recent history is frequently analyzed. In such cases, we show that an effective solution is to bias the sample with the use of temporal bias functions. The maintenance of such a sample is non-trivial, since it needs to be *dynamically maintained, without knowing the total number of points in advance.* We prove some interesting theoretical properties of a large class of memory-less bias functions, which allow for an efficient implementation of the sampling algorithm. We also show that the inclusion of bias in the sampling process introduces a *maximum requirement* on the reservoir size. This is a nice property since it shows that it may often be possible to maintain the maximum *relevant* sample with limited storage requirements. We not only illustrate the advantages of the method for the problem of query estimation, but also show that the approach has applicability to broader data mining problems such as evolution analysis and classification.

1. INTRODUCTION

In recent years, the problem of synopsis maintenance [3, 7, 8, 11, 12, 13, 14, 16, 17] has been studied in great detail because of its application to problems such as query estimation [4, 8, 17] in data streams. Many synopsis methods such as sampling, wavelets, histograms and sketches [2, 9, 10, 11, 15] are designed for use with specific applications such as approximate query answering. A comprehensive survey of stream synopsis construction algorithms may be found in [2]. An important class of stream synopsis construction methods is that of reservoir sampling [16]. The method of sampling has great appeal because it generates a sample of the original multi-dimensional data representation. Therefore, it can be used with arbitrary data mining applications with little changes to the underlying methods and algorithms.

In the case of a fixed data set of *known* size N, it is trivial to construct a sample of size n, since all points have an inclusion probability of n/N. However, a data stream is a continuous process, and it is not known in advance how many points may elapse before an analyst may need to use a representative sample. In other words, the base data size N is not known in advance. Therefore, one needs to maintain a *dynamic sample* or *reservoir*, which reflects the current history in an unbiased way. The reservoir is maintained by probabilistic insertions and deletions on arrival of new stream points. We note that as the length of the stream increases, an unbiased sample contains a larger and larger fraction of points from the distant history of the stream. In reservoir sampling, this is also reflected by the fact that the probability of successive insertion of new points reduces with progression of the stream [16]. A completely unbiased sample is often undesirable from an application point of view, since the data stream may evolve, and the vast majority of the points in the sample may represent the stale history of the stream. For example, if an application is queried for the statistics for the past hour of stream arrivals, then for a data stream which has been running over one year, only about 0.01% of an unbiased sample may be relevant. The imposition of range selectivity or other constraints on the query will reduce the relevant estimated sample further. In many cases, this may return a null or wildly inaccurate result. In general, this also means that *the quality of the result for the same query will only degrade with progression of the stream*, as a smaller and smaller portion of the sample remains relevant with time. This is also the most important case for stream analytics, since the same query over recent behavior may be repeatedly used with progression of the stream. This point has been noted in the context of the design of exponential histograms and other methods for tracking decaying stream statistics [5, 6], though these methods are not applicable to the general problem of drawing a biased reservoir from the data stream.

One solution is to use a sliding window approach for restricting the horizon of the sample [3]. However, the use of a pure sliding window to pick a sample of the immedi-

VLDB '06, September 12-15, 2006, Seoul, Korea.

ately preceding points may represent another extreme and rather unstable solution. This is because one may not wish to completely lose the entire history of past stream data. While analytical techniques such as query estimation may be performed more frequently for recent time horizons, distant historical behavior may also be queried periodically. We show that a practical solution is to use a temporal bias function in order to regulate the choice of the stream sample. Such a solution helps in cases where it is desirable to obtain both biased and unbiased results. For example, in some data mining applications, it may be desirable to bias the result to represent more recent behavior of the stream [1]. In other applications such as query estimation, while it may be desirable to obtain unbiased query results, it is more critical and challenging to obtain accurate results for queries over recent horizons. We will see that the biased sampling method allows us to achieve both goals. We will derive the following set of results:

- In general, it is non-trivial to extend reservoir maintenance algorithms to the biased case. In fact, it is an open problem to determine whether reservoir maintenance can be achieved in one-pass with arbitrary bias functions. However, we theoretically show that in the case of an important class of memory-less bias functions (exponential bias functions), the reservoir maintenance algorithm reduces to a form which is simple to implement in a one-pass approach.

- The inclusion of a bias function imposes a *maximum requirement* on the sample size. Any sample satisfying the bias requirements will not have size larger than a function of N. This function of N defines a maximum requirement on the reservoir size which is significantly less than N. In the case of the memory-less bias functions, we will show that this maximum sample size is independent of N and is therefore bounded above by a *constant* even for an infinitely long data stream. This is a nice property, since it means that the maximum relevant sample may be maintained in main memory in many practical scenarios.

- We will theoretically analyze the accuracy of the approach on the problem of query estimation. This will clarify the advantages of the biased sampling approach.

- In the experimental section, we will not only illustrate the advantages of the method for the problem of query estimation, but also show that the approach has applicability to broader data mining problems such as classification.

This paper is organized as follows. In the next section, we will discuss the concept of maximum reservoir requirements, and present a simple maintenance algorithm for memory-less functions. In section 3, we will extend this method to the space constrained case. The analysis of the method for query estimation is presented in section 4. The algorithm is experimentally tested for the query estimation and classification problems in section 5. The conclusions and summary are presented in section 6.

2. STREAM EVOLUTION AND RESERVOIR REQUIREMENTS

Before discussing our biased sampling method, we will introduce the simple reservoir maintenance algorithm discussed in [16]. In this method, we continuously maintain a reservoir of size n from the data stream. The first n points in the data stream are added to the reservoir for initialization. Subsequently, when the $(t+1)$-th point from the data stream is received, it is added to the reservoir with probability $n/(t+1)$. This point replaces a randomly chosen point in the reservoir. We note that the probability value $n/(t+1)$ reduces with stream progression. The reservoir maintenance algorithm satisfies the following property:

PROPERTY 2.1. *After t points in the data stream have been processed, the probability of any point in the stream belonging to the sample of size n is equal to n/t.*

The proof that this sampling approach maintains the unbiased character of the reservoir is straightforward, and uses induction on t. After the $(t+1)$-th point is processed, its probability of being included in the reservoir is $n/(t+1)$. The probability of any of the last t points being included in the reservoir is defined by the sum of the probabilities of inclusion together with the event of the $(t+1)$-th point being added (or not) to the reservoir. By using this relationship in conjunction with the inductive assumption, the corresponding result can also be proved for the first t points.

One interesting characteristic of this maintenance algorithm is that it is extremely efficient to implement in practice. When new points in the stream arrive, we only need to decide whether or not to insert into the current sample array which represents the reservoir. The sample array can then be overwritten at a random position. Next, we will define the biased formulation of the sampling problem.

The bias function associated with the r-th data point at the time of arrival of the t-th point ($r \leq t$) is given by $f(r,t)$ and is related to the probability $p(r,t)$ of the r-th point belonging to the reservoir at the time of arrival of the t-th point. Specifically, $p(r,t)$ is proportional to $f(r,t)$. The function $f(r,t)$ is monotonically decreasing with t (for fixed r) and monotonically increasing with r (for fixed t). Therefore, the use of a bias function ensures that recent points have higher probability of being represented in the sample reservoir. Next, we define the concept of a bias-sensitive sample $S(t)$, which is defined by the bias function $f(r,t)$.

DEFINITION 2.1. *Let $f(r,t)$ be the bias function for the r-th point at the arrival of the t-th point. A* biased *sample $S(t)$ at the time of arrival of the t-th point in the stream is defined as a sample such that the relative probability $p(r,t)$ of the r-th point belonging to the sample $S(t)$ (of size n) is proportional to $f(r,t)$.*

We note that for the case of general functions $f(r,t)$, it is an open problem to determine if maintenance algorithms can be implemented in one pass. In the case of unbiased

maintenance algorithms, we only need to perform a single insertion and deletion operation periodically on the reservoir. However, in the case of arbitrary functions, the entire set of points within the current sample may need to re-distributed in order to reflect the changes in the function $f(r,t)$ over different values of t. For a sample $S(t)$ this requires $\Omega(|S(t)|) = \Omega(n)$ operations, *for every point in the stream* irrespective of whether or not insertions are made. For modestly large sample sizes n, this can be fairly inefficient in practice. Furthermore, the re-distribution process is unlikely to maintain a constant size for the reservoir during stream progression.

In this paper, we will leverage and exploit some interesting properties of a class of memory-less bias functions. The *exponential bias* function is defined as follows:

$$f(r,t) = e^{-\lambda(t-r)} \tag{1}$$

The parameter λ defines the bias rate and typically lies in the range $[0,1]$ with very small values. In general, this parameter λ is chosen in an application specific way, and is expressed in terms of the inverse of the number of data points after which the (relative) probability of inclusion in the reservoir reduces by factor of $1/e$. A choice of $\lambda = 0$ represents the unbiased case. The exponential bias function defines the class of *memory-less functions* in which the future probability of retaining a current point in the reservoir is independent of its past history or arrival time. While biased reservoir sampling is non-trivial and may not be efficiently implementable in the general case, we will show that this class of memory-less functions have some interesting properties which allow an extremely simple and efficient implementation. First, we will examine the issue of *reservoir requirements* both for general bias functions as well as the exponential function.

In the case of traditional unbiased sampling, one may maintain a sample as large as the entire stream itself. This is not possible in a practical setting, because of the high volume of the data stream. However, in the case of biased sampling, the introduction of bias results in some upper bounds on the maximum *necessary* sample size. Since $p(r,t)$ is proportional to $f(r,t)$, the following is true for some constant C:

$$p(r,t) = C \cdot f(r,t) \qquad \forall r = \{1 \ldots t\} \tag{2}$$

Let X_i be the (binary) indicator variable representing whether or not the point i is included in the reservoir sample after the arrival of data point t. Then, the expected sample size $S(t)$ is defined as the sum of the expected values of the indicator variables.

$$E[|S(t)|] = \sum_{i=1}^{t} E[X_i] = \sum_{i=1}^{t} p(i,t) \tag{3}$$

$$= C \cdot \sum_{i=1}^{t} f(i,t) \tag{4}$$

Since we are examining only those sampling policies which maintain the constant size n of the sample, the value of $|S(t)|$ will always be deterministically equal to n. Therefore, we have:

$$n = C \cdot \sum_{i=1}^{t} f(i,t) \tag{5}$$

We can derive an expression for C from the above relationship. By using this expression for C, we can derive the expression for $p(r,t)$.

$$p(r,t) = n \cdot f(r,t)/(\sum_{i=1}^{t} f(i,t)) \tag{6}$$

Since $p(r,t)$ is a probability, we have:

$$p(r,t) \leq 1 \quad \forall r \in \{1 \ldots t\} \tag{7}$$

By substituting the value of $p(r,t)$, we get:

$$n \cdot f(r,t)/(\sum_{i=1}^{t} f(i,t)) \leq 1 \quad \forall r \in \{1 \ldots t\} \tag{8}$$

$$n \leq (\sum_{i=1}^{t} f(i,t))/f(r,t) \tag{9}$$

Since the function $f(r,t)$ is monotonically increasing with r, the tightest possible bound is obtained by choosing $r = t$. Therefore, we have:

$$n \leq \sum_{i=1}^{t} f(i,t)/f(t,t) \tag{10}$$

The above relationship represents the maximum reservoir requirement at the time of arrival of the t-th point. Note that each of the t terms in the right hand side of above expression is at most 1, and therefore the maximum reservoir requirement in a biased sample is often significantly less than the number of points in the stream. We summarize the requirement as follows:

THEOREM 2.1. *The maximum reservoir requirement $R(t)$ for a random sample from a stream of length t which satisfies the bias function $f(r,t)$ is given by:*

$$R(t) \leq \sum_{i=1}^{t} f(i,t)/f(t,t) \tag{11}$$

For the particular case of the exponential function with parameter λ, we can derive the following result:

LEMMA 2.1. *The maximum reservoir requirement $R(t)$ for a random sample from a stream of length t which satisfies the exponential bias function $f(r,t) = e^{-\lambda(t-r)}$ is given by:*

$$R(t) \leq (1 - e^{-\lambda t})/(1 - e^{-\lambda}) \tag{12}$$

PROOF. We instantiate the result of Theorem 2.1 to the case of an exponential bias function. Therefore, we have:

$$R(t) \leq \sum_{i=1}^{t} e^{-\lambda(i-1)} \tag{13}$$

$$= (1 - e^{-\lambda t})/(1 - e^{-\lambda}) \tag{14}$$

□

We note that the above constraint on the reservoir requirement is bounded above the constant $1/(1-e^{-\lambda})$ irrespective

of the length of the stream t. Furthermore, for a long stream, this bound is tight since we have:

$$\lim_{t\Rightarrow\infty} R(t) \leq \lim_{t\Rightarrow\infty}(1-e^{-\lambda t})/(1-e^{-\lambda}) \quad (15)$$

$$= 1/(1-e^{-\lambda}) \quad (16)$$

We summarize this result as follows:

COROLLARY 2.1. *The maximum reservoir requirement $R(t)$ for a random sample from a stream of length t which satisfies the exponential bias function $f(r,t) = e^{-\lambda(t-r)}$ is bounded above by the constant $1/(1-e^{-\lambda})$.*

We note that the parameter λ is expressed in terms of the inverse of the number of data points after which the probability reduces by factor of $1/e$. In most stable applications, we assume that the function reduces slowly over several thousands of points. Since this is also the most interesting case with the highest reservoir requirements, we assume that λ is much smaller than 1. Therefore, we can approximate $e^{-\lambda} \approx 1-\lambda$. Therefore, we have the following approximation:

APPROXIMATION 2.1. *The maximum reservoir requirement $R(t)$ for a random sample from a stream of length t which satisfies the exponential bias function $f(r,t) = e^{-\lambda(t-r)}$ is approximately bounded above by the constant $1/\lambda$.*

When the value of $1/\lambda$ is much smaller than the space constraints of the sampling algorithm, it is possible to hold the entire relevant sample within the required space constraints. This is a nice property, since we are then assured of the most robust sample satisfying that bias function. We propose the following algorithm to achieve this goal.

ALGORITHM 2.1. *We start off with an empty reservoir with capacity $n = [1/\lambda]$, and use the following replacement policy to gradually fill up the reservoir. Let us assume that at the time of (just before) the arrival of the t-th point, the fraction of the reservoir filled is $F(t) \in [0,1]$. When the $(t+1)$-th point arrives. we deterministically add it to the reservoir. However, we do not necessarily delete one of the old points in the reservoir. We flip a coin with success probability $F(t)$. In the event of a success, we randomly pick one of the old points in the reservoir, and replace its position in the sample array by the incoming $(t+1)$-th point. In the event of a failure, we do not delete any of old points and simply add the $(t+1)$-th point to the reservoir. In the latter case, the number of points in the reservoir (current sample size) increases by 1.*

One observation about this policy is that it is extremely simple to implement in practice, and is no more difficult to implement than the unbiased policy. Another interesting observation is that the insertion and deletion policies are parameter free, and are *exactly the same* across all choices of the bias parameter λ. The only difference is in the choice of the reservoir size. Therefore, assuming that the policy of Algorithm 2.1 does indeed achieve exponential bias with parameter λ (we prove this slightly later), we observe the following:

OBSERVATION 2.1. *The replacement policy of Algorithm 2.1, when implemented across different reservoir sizes, results in an exponential bias of sampling, and the parameter of this bias is decided by the reservoir size.*

In practice, the reservoir size should be chosen on the basis of the application specific parameter λ, and not the other way around. We note that because of the dependence of ejections on the value of $F(t)$, the reservoir fills up rapidly at first. As the value of $F(t)$ approaches 1, further additions to the reservoir slow down. When the reservoir is completely full, we have a deterministic insertion policy along with a deletion policy which is equivalent to that of equi-probability sampling. It remains to show that the above replacement policy results in a dynamic sample of size n which always satisfies the exponential bias behavior with parameter $\lambda = 1/n$.

THEOREM 2.2. *When Algorithm 2.1 is applied, the probability of the r-th point in the stream at time t being included in a reservoir of maximum size n is approximately equal to $f(r,t) = e^{-(t-r)/n} = e^{-\lambda(t-r)}$.*

PROOF. We note that in Algorithm 2.1, we first flip a coin with success probability $F(t)$ and in the event of a success, we eject any of the $n \cdot F(t)$ points in reservoir with equal probability of $1/(n \cdot F(t))$. Therefore, the probability of a point in the reservoir being ejected in a given iteration is $F(t) \cdot (1/(n \cdot F(t))) = 1/n$. Since insertions are deterministic, the r-th point in the reservoir remains at time t, if it does not get ejected in $t-r$ iterations. The probability of this is $(1-1/n)^{t-r} = ((1-1/n)^n)^{(t-r)/n}$. For large values of n, the inner $(1-1/n)^n$ term is approximately equal to $1/e$. On substitution, the result follows. □

3. SAMPLING WITH STRONG SPACE CONSTRAINTS

The algorithm in the previous section discussed the case when the available space exceeds the maximum reservoir requirements $1/\lambda$. What happens when the space is constrained, and the available reservoir size is less than this value? This can happen in very fast data streams in which even a small fraction of the stream can greatly exceed main memory limitations, or in the case when there are thousands of independent streams, and the amount of space allocated for each is relatively small. Such cases are also the most important scenarios in real applications. We will show that with some subtle changes to Algorithm 2.1, it is possible to handle the case when the space availability is significantly less than the reservoir requirement. We note that in the case of Algorithm 2.1, the insertion policy is deterministic, and therefore the corresponding insertion probability p_{in} is effectively 1. By reducing this insertion probability p_{in}, it is possible to store a sample with the same bias function within a smaller reservoir. As we will discuss later, we will pick the value of p_{in} depending upon application specific parameters λ and n. We propose the following modified version of Algorithm 2.1:

ALGORITHM 3.1. *We start off with an empty reservoir with capacity $n = p_{in}/\lambda$, and use the following replacement*

policy to gradually fill up the reservoir. Let us assume that at the time of (just before) the arrival of the t-th point, the fraction of the reservoir filled is $F(t) \in [0, 1]$. When the $(t+1)$-th point arrives. we add it to the reservoir with insertion probability p_{in}. However, we do not necessarily delete one of the old points in the reservoir. We flip a coin with success probability $F(t)$. In the event of a success, we randomly pick one of the points in the reservoir, and replace its position in the sample array by the incoming $(t+1)$-th point. In the event of a failure, we do not delete any of old points and simply add the $(t+1)$-th point to the reservoir. In the latter case, the number of points in the reservoir (current sample size) increases by 1.

We note that this modified algorithm has a lower insertion probability, and a corresponding reduced reservoir requirement. In this case, the value of λ and n are decided by application specific constraints, and the value of p_{in} is set to $n \cdot \lambda$. We show that the sample from the modified algorithm shows the same bias distribution, but with a reduced reservoir size.

THEOREM 3.1. *When Algorithm 3.1 is applied, the probability of the r-th point in the stream at time t being included in a reservoir of maximum size n is approximately equal to* $p_{in} \cdot f(r, t) = p_{in} \cdot e^{-(t-r) \cdot p_{in}/n} = p_{in} \cdot e^{-\lambda(t-r)}$.

PROOF. A point is inserted into the reservoir with probability p_{in} at the r-th iteration. Then in each of the next $(t-r)$ iterations, this point may be ejected or retained. The point is retained if either no insertion is made (probability $(1 - p_{in})$), or if an insertion is made, but an ejection does not take place (probability $p_{in} \cdot (1 - 1/n)$). The sum of the probabilities of these two disjoint events is $1 - p_{in}/n$. Therefore, the probability that the r-th point is (first inserted and then) retained after the t-th iteration is equal to $p_{in} \cdot (1 - p_{in}/n)^{(t-r)} = p_{in} \cdot ((1 - p_{in}/n)^{n/p_{in}})^{p_{in} \cdot (t-r)/n}$. For large values of n/p_{in}, the inner $(1 - p_{in}/n)^{n/p_{in}}$ term is approximately equal to $1/e$. On substitution, the result follows. □

One subtle difference between the results of Theorems 2.2 and 3.1 is that while the sampling probabilities maintain their proportional behavior to $f(r, t)$ in accordance with Definition 2.1, we have an added proportionality factor p_{in} to represent the fact that not all points that could be included in the reservoir do end up being included because of space constraints.

Since the concept of reservoir based sampling is premised on its dynamic use during the collection process, we would like to always have a reservoir which is as much filled to capacity as possible. We note that the start up phase of filling the reservoir to capacity can take a while, especially for low choices of the insertion probability p_r. Any data mining or database operation during this initial phase would need to use the (smaller) unfilled reservoir. While the unfilled reservoir does satisfy the bias requirements of Definition 2.1, it provides a smaller sample size than necessary. Therefore, we will examine the theoretical nature of this initial behavior, and propose a simple solution to speed up the process of filling up the reservoir. The following result quantifies the number of points to be processed in order to fill up the reservoir.

THEOREM 3.2. *The expected number of points in the stream to be processed before filling up the reservoir completely is* $O(n \cdot log(n)/p_{in})$.

PROOF. Let q be the current number of points in the reservoir. The probability of adding a point to the reservoir is $p_{in} \cdot (n-q)/n$. Then, the expected number of stream points to be processed to add the next point to the reservoir is $n/(p_{in} \cdot (n-q))$. Then, the total expected number of points to be processed is $\sum_{q=0}^{n-1} n/(p_{in} \cdot (n-q)) = (n/p_{in}) \cdot \sum_{q=1}^{n} (1/q)$. This is approximately equal to $O(n \cdot \log(n)/p_{in})$. □

On examining the proof of the above theorem in detail, it is easy to see that most of the processing is incurred in filling up the last few points of the reservoir. In fact, a minor modification of the above proof shows that in order to fill up the reservoir to a fraction f, the number of points to be processed are linear in reservoir size.

COROLLARY 3.1. *The expected number of points in the stream to be processed before filling up the reservoir to a target fraction f is at most* $O(n \cdot log(1/f)/p_{in})$.

This is a nice property, since it shows that a processing phase which is linear in reservoir size can fill the reservoir size to a pre-defined fraction. Nevertheless, the number of points to be processed are still inversely related to p_{in}. The value of $1/p_{in}$ can be rather large in space constrained applications, and this will result in a sub-optimally full reservoir for a long initial period. How do we solve this problem?

The reservoir size is proportional to the insertion probability p_{in} for fixed λ. When the reservoir is relatively empty, we do not need to use a reservoir within the memory constraints. Rather, we can use a larger value of p_{in}, and "pretend" that we have a fictitious reservoir of size p_{in}/λ available. Note that the value of $F(t)$ is computed with respect to the size of this fictitious reservoir. As soon as this fictitious reservoir is full to the extent of the (true) space limitations, we eject a certain percentage of the points, reduce the value of p_{in} to p'_{in} and proceed. The key issue is that we are mixing the points from two different replacement policies. Would the mixed set continue to satisfy the conditions of Definition 2.1?

Note that Theorem 3.1 has sample bias probabilities proportional to $p_{in} \cdot f(r, t)$, and the varying value of p_{in} needs to be accounted for while mixing the points from two different policies. In order to account for the varying value of p_{in}, we eject a fraction $1 - p'_{in}/p_{in}$ of the points from the reservoir at the time of reducing the insertion probability from p_{in} to p'_{in}. This ensures that proportionality to $p'_{in} \cdot f(r, t)$ is maintained by all points in the reservoir even after mixing. This establishes the following result:

THEOREM 3.3. *If we apply Algorithm 3.1 for any number of iterations with parameters p_{in} and λ, eject a fraction* $1 -$

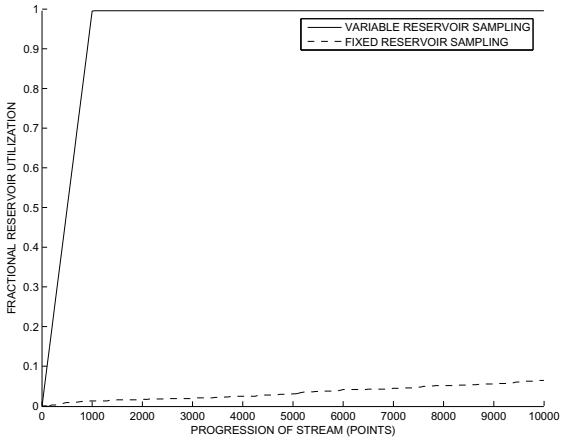


Figure 1: Effects of using variable reservoir sampling

(p'_{in}/p_{in}) of the points, and then apply Algorithm 3.1 for any number of iterations with parameters p'_t and λ, the resulting set of points satisfies the conditions of Definition 2.1 with $f(r,t) = e^{-\lambda(t-r)}$.

This suggests a natural algorithm for quickly building the reservoir and maintaining it at near full capacity. Let us assume that the total space availability is n_{max}. We start off with $p_{in} = 1$, and add points to the reservoir using Algorithm 3.1, until the space limitation n_{max} of the (true) reservoir size is reached. We multiply the value of p_{in} by a given fraction (say $1-q$), and eject a fraction q of the points. We note that the exact choice of reduction factor does not affect the correctness of the method because of Theorem 3.3. Then we continue the process of adding points to the reservoir according to Algorithm 3.1. We repeat with the process of successively adding points to the reservoir and reducing p_{in} until the value of the insertion probability is equal to the target probability of $n_{max} \cdot \lambda$. In the last iteration, the value of p_{in} may be reduced by less than the usual rate since we do not wish to reduce p_{in} to below $n_{max} \cdot \lambda$. A particularly useful choice of parameters is to pick $q = 1/n_{max}$, and eject exactly one point in each such phase. This ensures that the reservoir is almost full most of the time, and even in the worst case at most one point is missing. By using this approach, the reservoir can be filled at the rate of $p_{in} = 1$ for the first time, and subsequently maintained at almost full capacity thereafter. We will refer to this technique as *variable reservoir sampling* to illustrate the variable size of the fictitious reservoir over time. We will refer to our earlier strategy of using only the true reservoir as *fixed reservoir sampling*. We note that there is no theoretical difference between the two methods in terms of the probabilistic distribution of the sampled reservoir. The real difference is in terms of the startup times to fill the reservoir.

In order to illustrate the advantage of the variable reservoir sampling scheme, we tested it on the network intrusion data stream which is described later in the experimental section. We used a true reservoir size $n_{max} = 1000$ and $\lambda = 10^{-5}$. At each filling of the reservoir, p_{in} was reduced by $1/n_{max}$ so that exactly one slot in the reservoir was empty. The results are illustrated in Figure 1. The X-axis shows the progression of the stream in terms of the number of points and the Y-axis shows the percentage of fullness of the (true) reservoir size. An immediate observation is that the variable reservoir sampling scheme is able to fill the 1000-point reservoir to capacity after processing only 1001 data points. Subsequently, the reservoir is maintained at full capacity. On the other hand, the fixed reservoir sampling scheme contains only 14 points after processing 1000 points in the data stream. After processing 10,000 points (right end of chart), the reservoir in the fixed sampling scheme contains only 66 points. We have not shown the chart beyond 10,000 points in order to retain the clarity of both curves[1] in Figure 1. We briefly mention the behavior of the fixed scheme outside this range. After 100,000 points have been processed, the reservoir still contains only 634 data points. In fact, even after processing the entire stream of 494020 points, the reservoir is still not full and contains 986 data points. We note that the variable scheme provides larger samples in the initial phase of processing. Clearly, for samples with the same distribution, a larger reservoir utilization provides more robust results.

4. APPLICATION TO QUERY SELECTIVITY ESTIMATION

In this section, we discuss how this technique may be used for query estimation. Let us consider a query which computes a linear function of the points $\overline{X_1} \ldots X_t$ in the stream. Let us assume that the function that we wish to compute is given by the following expression:

$$G(t) = \sum_{i=1}^{t} c_i \cdot h(\overline{X_i}) \tag{17}$$

The above relationship represents a simple linearly separable function of the underlying data values. We note that the *count* , and *sum* queries are specific instantiations of the above expression. For the case of the *count function*, we have $h(\overline{X_i}) = 1$, $c_i = 1$ and for the case of the *sum function*, we have $h(\overline{X_i}) = \overline{X_i}$, $c_i = 1$. We note that by using different choices of $h(\overline{X_i})$, it is possible to perform different kinds of queries. For example, one may choose $h(\overline{X_i})$ to be 1 or 0, depending upon whether or not it lies in a range (range query). In many scenarios, it may be desirable to perform the query only over the recent history of the data stream. In such cases, one may choose $h(\overline{X_i})$ to be 1 or 0 depending upon whether or not it lies in a user-defined horizon. Let us define the indicator function $\mathcal{I}_{r,t}$ to be 1 or 0, depending upon whether or not the r-th point is included in the sample reservoir at the t-th iteration. Now, let us define the random variable $H(t)$ as follows:

$$H(t) = \sum_{r=1}^{t} (\mathcal{I}_{r,t} \cdot c_r \cdot h(\overline{X_r}))/p(r,t) \tag{18}$$

We make the following observation about the relationship between $G(t)$ and the expected value of the random variable $H(t)$.

OBSERVATION 4.1. $E[H(t)] = G(t)$

[1]By extending the chart to the full stream of almost half a million points, the curve for the variable scheme practically overlaps with the Y-axis, whereas the reservoir in the fixed scheme still contains only 986 points.

This observation is easy to prove by taking the expected value of both sides in Equation 18, and observing that $E[\mathcal{I}_{r,t}] = p(r,t)$. The relationship of Equation 18 provides a natural way to estimate $G(t)$ by instantiating the indicator function in Equation 18.

Let us assume that the points in sample of size n were obtained at iterations $i_1 \ldots i_n$ at the t-th iteration. Then, the probability that the point i_k is included in the sample is given by $p(i_k, t)$. Then, by using the expression in Equation 18, it follows that the *realized* value of $H(t)$ for this particular sample is equal to $\sum_{q=1}^{n} c_{i_q} \cdot h(\overline{X_{i_q}})/p(i_q, t)$. This realized value of $H(t)$ can be estimated as the expected value of $H(t)$, and is therefore also an estimation for $G(t)$. The error of the estimation can be estimated as $Var(H(t))$ which denotes the variance of the random variable $H(t)$.

LEMMA 4.1. *$Var[H(t)] = \sum_{r=1}^{t} K(r,t)$ where $K(r,t) = c_r^2 \cdot h(\overline{X}_r)^2 \cdot (1/p(r,t) - 1)$*

PROOF. From Equation 18, we get:

$$Var[H(t)] = Var[\sum_{r=1}^{t} (\mathcal{I}_{r,t} \cdot c_r \cdot h(\overline{X_r}))/p(r,t)] \quad (19)$$

By using independence of different values of $\mathcal{I}_{r,t}$, we can decompose the variance of sum on the right hand side into the sum of variances. At the same time, we can bring the constants out of the expression. Therefore, we have:

$$Var[H(t)] = \sum_{r=1}^{t} (c_r^2 \cdot h(\overline{X_r})^2)/p(r,t)^2 \cdot Var[\mathcal{I}_{r,t}] \quad (20)$$

Now, we note that the indicator function $\mathcal{I}(r,t)$ is a bernoulli random variable with probability $p(r,t)$. The variance of this bernoulli random variable is given by $Var[\mathcal{I}_{r,t}] = p(r,t) \cdot (1 - p(r,t))$. By substituting for $Var[\mathcal{I}(r,t)]$ in Equation 20, we obtain the desired result. □

The key observation here is that the value of $K(r,t)$ is dominated by the behavior of $1/p(r,t)$ which is relatively small for larger values of r. However, for recent horizon queries, the value of c_r is 0 for smaller values of r. This reduces the overall error for recent horizon queries.

We note that the above-mentioned example of query estimation is not the only application of the biased sampling approach. The technique can be used for any data mining approach which uses sampling in conjunction with the gradual reduction in the importance of data points for mining purposes. For example, some recent methods for clustering [1] use such a bias factor in order to decide the relative importance of data points. The biased sample can be used as the base data set to simulate such a scenario. The advantage of using a sampling approach (over direct mining techniques on the data stream) is that we can use any blackbox mining algorithm over the smaller sample. In general, many data mining algorithms require multiple passes in conjunction with parameter tuning in order to obtain the best results. In other cases, an end-user may wish to apply the data mining algorithm in an exploratory way over different choices of parameters. This is possible with a sample, but not with the original stream. The use of sampling frees us from the strait-jacket of a one-pass method. In the next section, we will experimentally illustrate the advantages of the biased sampling method on some real applications.

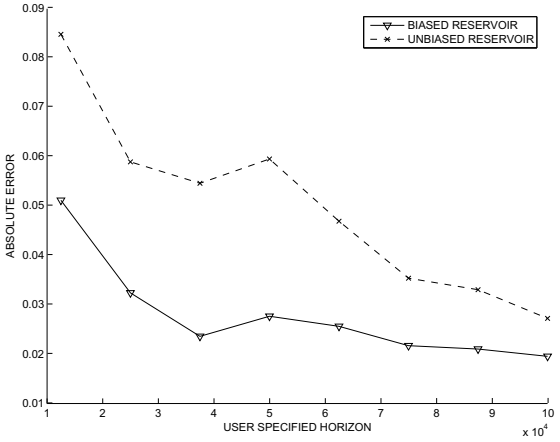


Figure 2: Sum Query Estimation Accuracy with User-defined horizon (Network Intrusion Data Set)

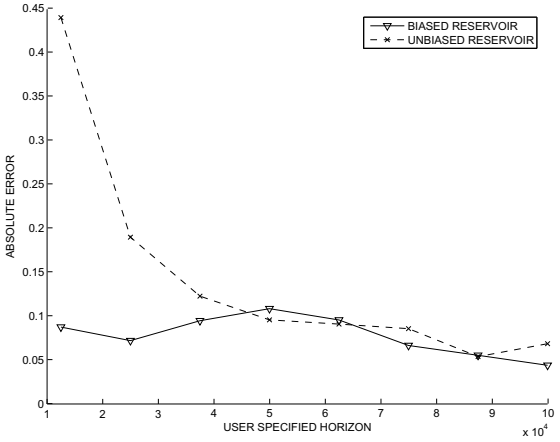


Figure 3: Sum Query Estimation Accuracy with User-defined horizon (Synthetic Data Set)

5. EXPERIMENTAL RESULTS

In this section, we will discuss a number of experimental results illustrating the effectiveness of the method for both query estimation and a number of data mining problems such as classification and evolution analysis. We will also examine the effects of evolution on the quality of the representative reservoir in both the unbiased and biased sampling strategy. We will show that the biased sampling strategy is an effective approach in a variety of scenarios.

5.1 Data Sets

We used both real and synthetic data sets. The real data set was the KDD CUP 1999 data set from the UCI machine learning repository. The data set was converted into a stream using the same methodology discussed in [1]. We normalized the data stream, so that the variance along each dimension was one unit.

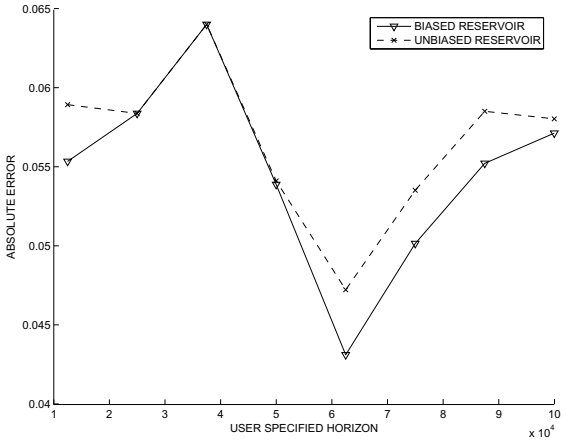


Figure 4: Count Query Estimation Accuracy with User-defined horizon (Network Intrusion Data Set)

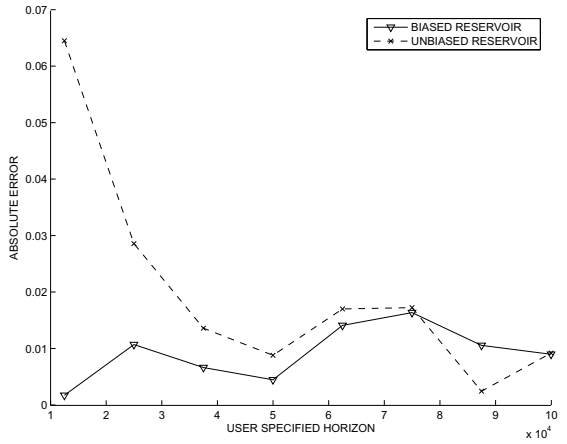


Figure 5: Range Selectivity Estimation Accuracy with User-defined horizon (Synthetic Data Set)

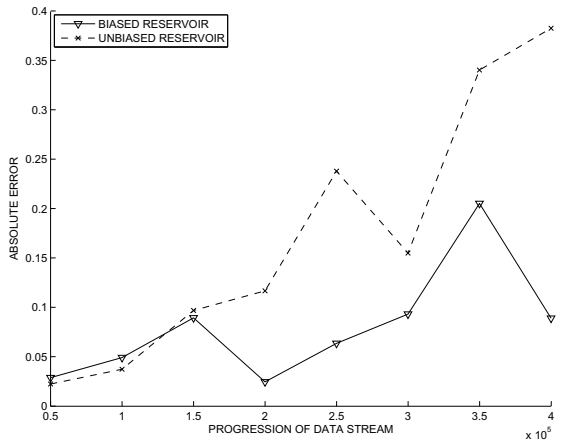


Figure 6: Error with stream Progression (Fixed Horizon $h = 10^4$)

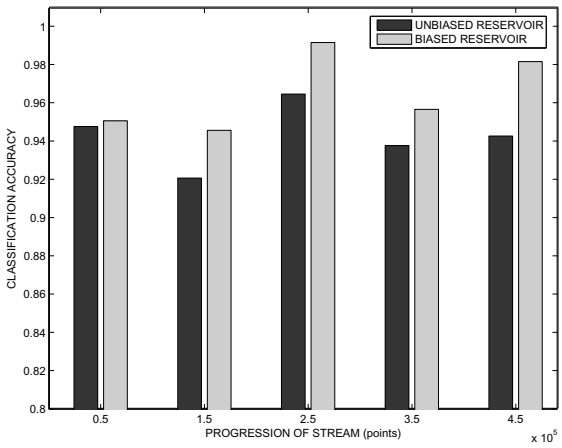


Figure 7: Classification Accuracy with Progression of Stream (Network Intrusion Data Set)

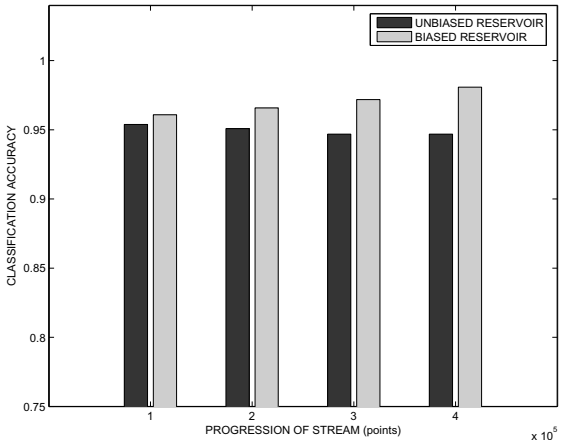


Figure 8: Classification Accuracy with Progression of Stream (Synthetic Data Set)

A 10-dimensional synthetic data set was generated using an evolving cluster scenario. In this case, we used $k = 4$ clusters which evolved continuously over time. We will use this interpretability for the case of the clustering application discussed later. The centers of each cluster were randomly chosen in the unit cube and the average radius of each cluster was set at 0.2. Thus, the individual points could lie outside the unit cube, and there was often a considerable level of overlap among the different clusters. The clusters were drawn from the gaussian distribution. After generation of each set of 100 data points, the center of each cluster moved by a random amount along each dimension by a number drawn in the range $[-0.05, 0.05]$. A continuous stream of $4 * 10^5$ data points was generated using this approach.

5.2 Query Estimation

In this section, we will discuss the application of this technique to the *count*, *sum* and *range selectivity* queries. In order to compare the effectiveness of biased and unbiased sampling, we used both an unbiased sample as well as a biased sample of exactly the same size. Since the paper is focussed on horizon-constrained queries, the timestamp needs to be maintained[2] for both the biased and unbiased reservoir. Therefore, the timestamp is not an additional overhead in the biased case. We note that horizon-driven queries are a practical choice in data streams because of the fact that users are often more interested in the recent historical behavior of the stream. Therefore, a user may only be interested in finding the corresponding statistics over a most recent horizon of length h. The queries were designed as follows:

- For the case of the *count* query, we wanted to find the distribution of the data points among the different classes for the data set in the past horizon of varying length h. For ease in analysis and interpretability of results, we represent h in the graphs in terms of the number of data points. The class-estimation query was applied only to the real network intrusion data set. We calculated the error in estimation of the query as the average absolute error in estimation of the fractional class distribution over the different classes. Thus, if f_i be the true fraction of class i, and f_i' be the fractional value, then the average error er over the l classes is defined as follows:

$$er = \sum_{i=1}^{l} |f_i - f_i|/l \tag{21}$$

  For the synthetic data set, we used a form of the count query which corresponds to range selectivity estimation. In this case, we estimated the fraction of all data points in a pre-determined horizon, for which a predefined set of dimensions lay in a user defined range.

- For the case of the *sum* queries, we wanted to estimate the average of all the data points in the past horizon h. This query was applied to both the real and synthetic data sets. As in the case of the count query, we calculate the average absolute error in estimation over the different dimensions.

In each case, we used a reservoir with 1000 data points, and $\lambda = 10^{-5}$. We compared the quality of the results of the biased sampling approach with those of the unbiased approach in [16]. In each case, we used a reservoir of exactly the same size in order to maintain the parity of the two schemes. The results for the sum query for the network intrusion and synthetic data sets are illustrated in Figures 2 and 3 respectively. In each case, we tested the scheme for variation in accuracy over different user horizons. On the X-axis, we have illustrated the user-defined horizon, and on the Y-axis, we have illustrated the average absolute error over the different dimensions. An immediate observation from Figures 2 and 3 is the effect of user-defined horizon on the accuracy of the two schemes.

- For small user-defined horizons, the error of the unbiased scheme is very high. This is also the most critical case for data stream scenarios, since only a small portion of the stream is usually relevant. With the use of smaller horizons, an unbiased reservoir often does not provide a large enough portion of the relevant sample for robust estimation. For larger horizons, the two schemes are almost competitive, and their relative performance is often because of the random variations in the data. Furthermore, since larger horizons allow for inherently more robust estimations (of both schemes) anyway, the small variations for this case are not as critical. Thus, these results show that the biased sampling approach provides a critical advantage for horizons of smaller size.

- The error rate is more stable in the case of the biased scheme with increasing user horizon. As is evident from Figures 2 and 3, the error rate does not vary much with the user-defined horizon for the two schemes, when the biased sampling method is used. In fact, in the case of Figure 3, the biased sampling method maintains an almost stable error rate with user horizon.

The results for the *count* and *range selectivity* queries are illustrated in Figures 4 and 5 respectively. The *class estimation count query* for the network intrusion data set (Figure 4) shows considerable random variations because of the skewed nature of the class distributions over the data stream. Furthermore, the error rates are not averaged over multiple dimensions as in the case of sum queries. However, even in this case, the biased sampling approach consistently outperforms the unbiased reservoir sampling method of [16]. The behavior for the case of range selectivity queries (Figure 5) is even more interesting since the error rate of the biased sampling method remains robust with variation in the horizon length. On the other hand, the variation in the error rate of the unbiased method is very sudden with increasing horizon. For the case of larger horizons, the two schemes are roughly competitive. We note that a very slight advantage for larger horizons is to be expected for the unbiased

[2]Since evolving data streams have an inherently temporal component, the time-stamp would generally need to be maintained for most practical applications. In the event of an application in which the time-stamp does not need to be maintained in the unbiased case, the overhead is $1/(d+1)$ for data of dimensionality d.

method, though the differences are relatively small. Furthermore, since larger horizons contain a greater portion of the sample and the absolute accuracy rates are relatively robust, this case is not as critical. Therefore, acceptable accuracy is achieved by both methods for those cases. Therefore, the *overwhelming advantages of the biased method for the small horizon query* provides a clear advantage to the biased sampling method.

In order to elucidate this point further, we ran the sum query on the synthetic data set and used a fixed horizon $h = 10^4$, but performed the same query at different points in the progression of the stream. This shows the variation in results with *stream progression*, and is analogous to many real situations in which the queries may not change very much over time, but new points in the stream will always continue to arrive. The results are illustrated in Figure 6. An important observation is that *with progression of the stream the error rates of the unbiased sampling method deteriorate rapidly.* On the other hand, the (memory-less) behavior of the biased sampling method ensures that the stream continues to remain relevant over time. Therefore, the results do not deteriorate as much. In the unbiased case, a larger and larger fraction of the reservoir corresponds to the distant history of the stream over time. This distant history is not used in the estimation process, and therefore the size of the relevant sample reduces rapidly. As a result, the accuracy rate deteriorates with stream progression. We also note that in most practical applications, the parameters (such as horizon) for user-queries are likely to remain constant, whereas the stream is likely to remain as a continuous process. This would result in continuous deterioration of result quality for the unbiased sampling method. Our biased sampling method is designed to reduce this problem.

5.3 Data Mining and Evolution Analysis Applications

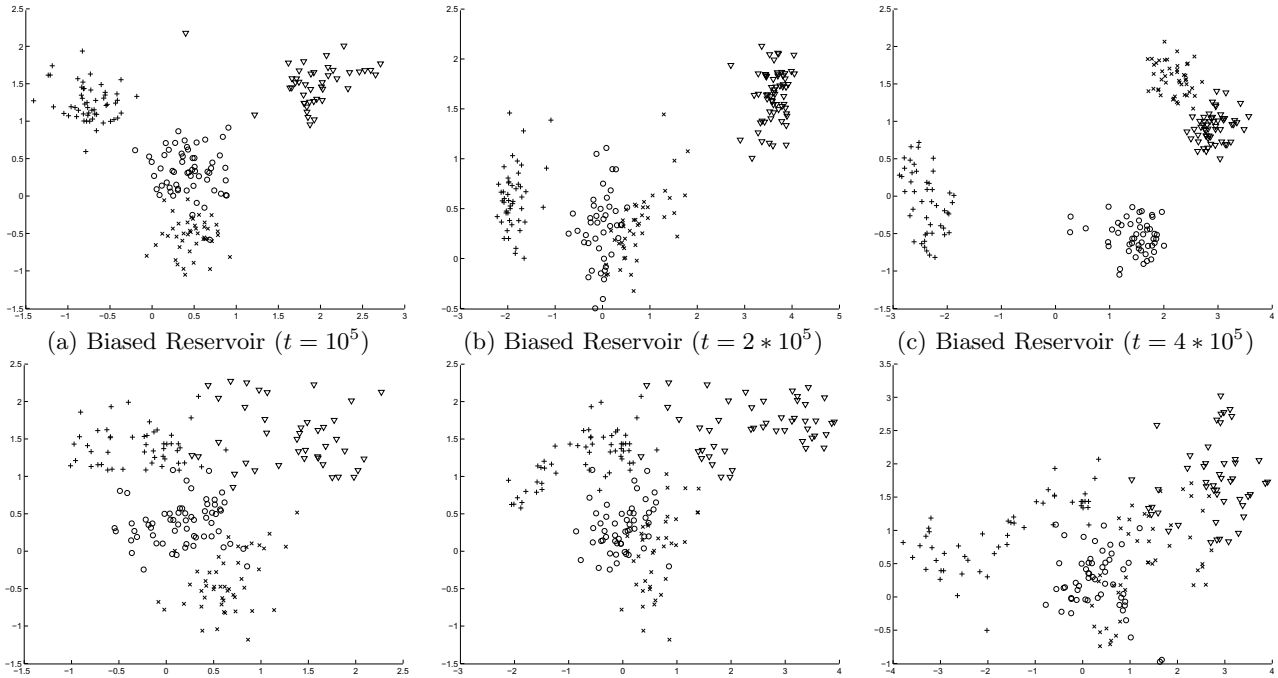
The applicability of the biased sampling method can be easily extended to wider classes of data mining problems such as classification and evolution analysis. We will show that the use of a biased reservoir preserves similar advantages as in the case of the query estimation method. In particular, the use of an unbiased reservoir causes deterioration of data mining results with the progression of the data stream, whereas the biased reservoir continues to maintain accurate results. We will also provide an intuitive understanding of the pattern of behavior of the results using some scatter plots of the evolution of the reservoirs with progression of the data stream.

In order to illustrate the effects of biased sampling method on data mining problems, we used the particular case of a nearest neighbor classifier. A nearest neighbor classifier is a typical application which requires sampling, because it is not possible to compare the test instance with every possible point in the history of the data stream. Therefore, a sample reservoir is used for comparison purposes. We used a reservoir of size 1000 and used $\lambda = 10^{-4}$. We tested the method for both the network intrusion and the synthetic data set. For the case of the synthetic data set, we used the cluster id as the class label. For each incoming data point, we first used the reservoir in order to classify it before reading its true label and updating the accuracy statistics. Then, we use the sampling policy to decide whether or not it should be added to the reservoir.

The classification results for the network intrusion data set are illustrated in Figure 7. We have illustrated the variation in classification accuracy with progression of the data stream. One observation from Figure 7 is that the use of either kind of reservoir provides almost similar classification accuracy at the beginning of the stream. However, with progression of the stream, the unbiased reservoir contains a larger and larger fraction of stale data points, and therefore the *relative* difference between the two methods increases. Because of random variations, this increase is not strictly monotonic. However, the overall trend shows a gradual increase in relative differences between the two methods.

This trend is even more obvious in the case of the classification results on the synthetic data set in Figure 8. The classes in the data sets correspond to evolving clusters which gradually drift apart from one another. As they drift apart, the data set becomes easier and easier to classify. This is reflected in increasing classification accuracy of the biased reservoir. However, the interesting observation is that the classification accuracy with the use of the unbiased reservoir actually *reduces* or remains stable with progression of the data stream. This is because the history of evolution of the different clusters contains a significant amount of overlap which is represented in the reservoir. With progression of the stream, recent points are given a smaller and smaller fraction of the representation. In many cases, stale points from the reservoir (belonging to the incorrect class) happen to be in closest proximity of the test instance. This results in a reduction of classification accuracy.

In order to illustrate this point, we constructed scatter plots of the evolution of the reservoir at different points in stream progression in Figure 9. The scatter plot was constructed on a projection of the first two dimensions. The points in the different clusters have been marked by different kinds of symbols such as a circle, cross, plus, and triangle. The charts in Figures 9(a), (b), and (c) illustrate the behavior of the evolution of the points in the *reservoir* at different points during stream progression. The charts in Figures 9(d), (e), and (f) illustrate the behavior of the *unbiased reservoir* at exactly the same points. By comparing the corresponding points in stream progression in pairwise fashion it is possible to understand the effect of evolution of the stream on the two kinds of reservoirs. It is clear from the progression in Figures 9(a), (b), and (c) that the clusters in the biased reservoir drift further and further apart with stream progression. This is reflective of the true behavior of the underlying data stream. On the other hand, the progression in Figures 9(d), (e), and (f) show greater diffusion and mixing of the points from different clusters with progression of the data stream. This is because the unbiased reservoir contains many points from the history of the stream which are no longer relevant to the current patterns in the data stream. Our earlier observations on classification accuracy variations with stream progression arise because of these reasons. The diffuse behavior of the distribution of points across different classes in the unbiased reservoir results in a lower classification accuracy. On the other hand, the biased reservoir maintains the sharp distinctions among different classes, and it therefore



(a) Biased Reservoir ($t = 10^5$) (b) Biased Reservoir ($t = 2 * 10^5$) (c) Biased Reservoir ($t = 4 * 10^5$)

(d) Unbiased Reservoir ($t = 10^5$) (e) Unbiased Reservoir ($t = 2 * 10^5$) (f) Unbiased Reservoir ($t = 4 * 10^5$)

Figure 9: Evolution of Reservoir Representation of Evolving Clusters (Synthetic Data Set)

continues to be effective with progression of the stream.

Similar advantages of the biased reservoir are likely to apply to a host of other aggregation based data mining algorithms such as clustering or frequent pattern mining. We expect that the biased reservoir sampling method is likely to be an effective tool to leverage the power of many compute-intensive data mining algorithms which can only be used in conjunction with smaller samples of the stream. In future work, we will examine the full power of this method on a variety of data mining problems.

6. CONCLUSIONS AND SUMMARY

In this paper, we proposed a new method for biased reservoir based sampling of data streams. This technique is especially relevant for continual streams which gradually evolve over time, as a result of which the old data becomes stale and is no longer relevant for a variety of data mining applications. While biased reservoir sampling is a very difficult problem (with the one pass constraint), we show that it is possible to design very efficient replacement algorithms for the important class of memory-less bias functions. In addition, the incorporation of bias results in upper bounds on reservoir sizes in many cases. In the special case of memory-less bias functions, we show that the maximum space requirement is constant even for an infinitely long data stream. This is a nice property, since it allows for easy implementation in a variety of space-constrained scenarios.

For the case in which the space-constraints are unusually tight, we designed replacement algorithms with very low startup times. This results in larger reservoir space utilization (and therefore greater robustness) during the initial phase. We discussed how to apply this technique to the problem of query estimation. We tested this method on both the query estimation problem and a variety of data mining techniques such as classification and evolution analysis. In each case, the biased reservoir method was more robust. Unlike the unbiased method, the behavior of the biased sample did not degrade with progression of the data stream across the different data mining and query estimation problems. This is also the most interesting case for a variety of practical scenarios.

7. REFERENCES

[1] C. Aggarwal, J. Han, J. Wang, and P. Yu. A Framework for High Dimensional Projected Clustering of Data Streams. *VLDB Conference Proceedings*, pp. 852–863, 2004.

[2] C. Aggarwal, and P. Yu. A Survey of Synopsis Construction Methods in Data Streams. *Data Streams: Models and Algorithms*, *Springer, ed. C. Aggarwal*, to appear 2007.

[3] B. Babcock, M. Datar, and R. Motwani. Sampling from a Moving Window over Streaming Data. *SODA Conference Proceedings*, pp. 633–634, 2002.

[4] S. Babu, and J. Widom. Continuous queries over data streams. *ACM SIGMOD Record Archives*, 30(3):109–120, 2001.

[5] E. Cohen, and M. Strauss. Maintaining Time-Decaying Stream Aggregates. *ACM PODS Conference Proceedings*, pp. 223-233, 2003.

[6] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining Stream Statistics over Sliding Windows. *SODA Conference Proceedings*, pp. 635–644, 2002.

[7] P. Gibbons, and Y. Mattias. New Sampling-Based Summary Statistics for Improving Approximate Query Answers. *ACM SIGMOD Conference Proceedings*, pp. 331–342, 1998.

[8] P. Gibbons. Distinct sampling for highly accurate answers to distinct value queries and event reports. *VLDB Conference Proceedings*, pp. 541–550, 2001.

[9] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. One-Pass Wavelet Decompositions of Data Streams. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):541–554, May 2003.

[10] P. Karras, and N. Mamoulis. One-pass wavelet synopsis for maximum error metrics. *VLDB Conference Proceedings*, pp. 421–432, 2005.

[11] Y. Matias, J. Vitter, and M. Wang. Wavelet-Based Histograms for Selectivity Estimation. *ACM SIGMOD Conference Proceedings*, pp. 448–459, 1998.

[12] J. Hellerstein, P. Haas, and H. Wang. Online Aggregation. *ACM SIGMOD Conference Proceedings*, pp. 171–182, 1997.

[13] G. Manku, S. Rajagopalan, and B. Lindsay. Random Sampling for Space Efficient Computation of order statistics in large datasets. *ACM SIGMOD Conference Proceedings*, pp. 251–262, 1999.

[14] G. Manku, and R. Motwani. Approximate Frequency Counts over Data Streams. *VLDB Conference Proceedings*, pp. 346–357, 2002.

[15] N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic multidimensional histograms. *ACM SIGMOD Conference Proceedings*, pp. 428–439, 2002.

[16] J. S. Vitter. Random Sampling with a Reservoir. *ACM Transactions on Mathematical Software*, Vol. 11(1):37–57, March 1985.

[17] Y. Wu, D. Agrawal, and A. Abbadi. Applying the Golden Rule of Sampling for Query Estimation. *ACM SIGMOD Conference Proceedings* pp. 449–460, 2001.