

Charu C. Aggarwal  
T J Watson Research Center  
IBM Corporation  
Hawthorne, NY  
USA

## On Biased Reservoir Sampling in the Presence of Stream Evolution

VLDB Conference, Seoul, South Korea, 2006

# Synopsis Construction in Data Streams

- Synopsis maintenance is an important problem in massive volume applications such as data streams.
- Many synopsis methods such as wavelets, histograms and sketches are designed for use with specific applications such as approximate query answering.
- An important class of stream synopsis construction methods is that of reservoir sampling (Vitter 1985).
- Great appeal because it generates a sample of the original multi-dimensional data representation.
- Can be used with arbitrary data mining applications with little changes to the underlying algorithms.

# Reservoir Sampling (Vitter 1985)

- 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.
- The base data size  $N$  is not known in advance.
- A reservoir or *dynamic sample* is maintained by probabilistic insertions and deletions on arrival of new stream points.
- **Challenge:** Probabilistic insertions and deletions always need to maintain unbiased sample.

# Reservoir Sampling

- 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.
- **Note:** Probability of insertion reduces with stream progression.
- **Property:** The reservoir sampling method maintains an unbiased sample of the history of the data stream (proof by induction).

# Observations

- In an evolving data stream only the more recent data may be relevant for many queries.
- 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.

## Observations

- In general, *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.

# Potential Solutions

- One solution is to use a sliding window approach for restricting the horizon of the sample.
- The use of a pure sliding window to pick a sample of the immediately 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.

# Biased Reservoir Sampling

- 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.
- In some data mining applications, it may be desirable to bias the result to represent more recent behavior of the stream.
- In other applications such as query estimation, while it may be desirable to obtain unbiased query results, it is more critical to obtain accurate results for queries over recent horizons.
- The biased sampling method allows us to achieve both goals.

# Contributions

- 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.
- 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  $N$ .

# Contributions

- 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.
- We will theoretically analyze the accuracy of the approach on the problem of query estimation.
- Test the method for the problem of query estimation and data mining problems.

# Bias Function

- 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  $f(r, t)$ .
- The probability  $p(r, t)$  of the  $r$ th point belonging to the reservoir at the time of arrival of the  $t$ th point 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.

## Biased Sample

- **Definition:** 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  $f(r, t)$ .
- 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.

# Challenges

- In the case of unbiased maintenance algorithms, we only need to perform a single insertion and deletion operation periodically on the reservoir.
- In the case of arbitrary functions, the entire set of points within the current sample may need to be 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.

# Memoryless Bias Functions

- The *exponential bias* function is defined as follows:

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

- The parameter  $\lambda$  defines the bias rate and typically lies in the range  $[0, 1]$  with very small values.
- 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.
- Memory-less bias functions are natural, and also allow for an extremely efficient extension of the reservoir sampling method.

# Maximum Reservoir Requirements

- **Result:** The maximum reservoir requirement  $R(t)$  for a random sample (without duplicates) from a stream of length  $t$  which satisfies the bias function  $f(r, t)$  is given by:

$$R(t) \leq \sum_{i=1}^t f(i, t) / f(t, t) \quad (2)$$

- **Proof Sketch:**

- Derive expression for probability  $p(r, t)$  in terms of reservoir size  $n$  and bias function  $f(r, t)$ .

$$p(r, t) = n \cdot f(r, t) / \left( \sum_{i=1}^t f(i, t) \right) \quad (3)$$

- Since  $p(r, t)$  is a probability, it is at most 1.
- Set  $r = t$  to obtain result.

# Maximum Reservoir Requirement for Exponential Bias Functions

- The maximum reservoir requirement  $R(t)$  for a random sample (without duplicates) from a stream of length  $t$  which satisfies the exponential bias function  $f(r, t) = e^{-\lambda(t-r)}$  is given by:

$$R(t) \leq (1 - e^{-\lambda t}) / (1 - e^{-\lambda}) \quad (4)$$

- **Proof Sketch:** Easy to show by instantiating result for general bias functions.

# Constant Upper bound for Exponential Bias Functions

- **Result:** 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  $1/(1 - e^{-\lambda})$ .
- **Approximation for small values of  $\lambda$ :** The maximum reservoir requirement  $R(t)$  for a random sample (without duplicates) 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  $1/\lambda$ .

# Implications of Constant Upper Bound

- For unbiased sampling, reservoir size may be as large as stream itself- no longer necessary for biased sampling!
- The constant upper bound shows that maximum reservoir size is not sensitive to how long the points from the stream are being received.
- Provides an estimate of the maximum sampling requirement.
- We can maintain the *maximum theoretical reservoir size* if sufficient main memory is available.

# Simple Reservoir Sampling Algorithm

- We start off with an empty reservoir with capacity  $n = \lceil 1/\lambda \rceil$ , 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.

# Simple Reservoir Sampling Algorithm

- 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.

# Simple Reservoir Sampling Algorithm

- **Result:** The simple reservoir sampling algorithm, when implemented across different reservoir sizes, results in an exponential bias of sampling, and the parameter of this bias is the inverse of the reservoir size.
- **Proof Sketch:** Use induction.

# Observations

- One observation about this policy is that it is extremely simple to implement in practice: 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  $\lambda$ .
- The only difference is in the choice of the reservoir size which depends on  $\lambda$ .
- 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.

## Strong Space Constraints

- The dependence of the reservoir size on the application-specific parameter  $\lambda$  can sometimes be a constraint.
- What do we do when the available space for sampling is less than the reservoir size implied by this application specific parameter?
- How do we work with a smaller reservoir while retaining the properties of the reservoir sampling method?

## Reducing Reservoir Size

- Note that the insertion policy of the simple reservoir sampling algorithm is deterministic.
- What happens when we retain the same deletion policy, but we also introduce a constant insertion probability  $p_{in}$ ?

## Reducing Reservoir Size

- **Result:** Using a constant insertion probability of  $p_{in}$  with parameter  $\lambda$  retains exponential bias but reduces the reservoir size by a factor  $p_{in}$  (to  $p_{in}/\lambda$ ).
- Thus, if the deterministic policy results in a reservoir size  $1/\lambda > M$ , where  $M$  is the available space, we can use  $p_{in} = M \cdot \lambda$ , and a reservoir size of  $M$ .
- **Proof of Correctness:** By induction.

# Dilemma

- We start off with an empty reservoir.
- **Result:** The expected number of points in the stream to be processed before filling up the reservoir completely is  $O(n \cdot \log(n)/p_{in})$ .
- For small values of  $p_{in}$ , the fillup time for the reservoir can be quite large.
- Means that we have to work with unnecessarily small samples for a while.
- We solved the reservoir size problem, but ended up with an initialization problem instead!

# Solution

- The reservoir size is proportional to the insertion probability  $p_{in}$  for fixed  $\lambda$ .
- 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  $p_{in}/\lambda$  available.
- Note that the value of  $F(t)$  is computed with respect to the size of this fictitious reservoir for the deletion policy.
- 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.

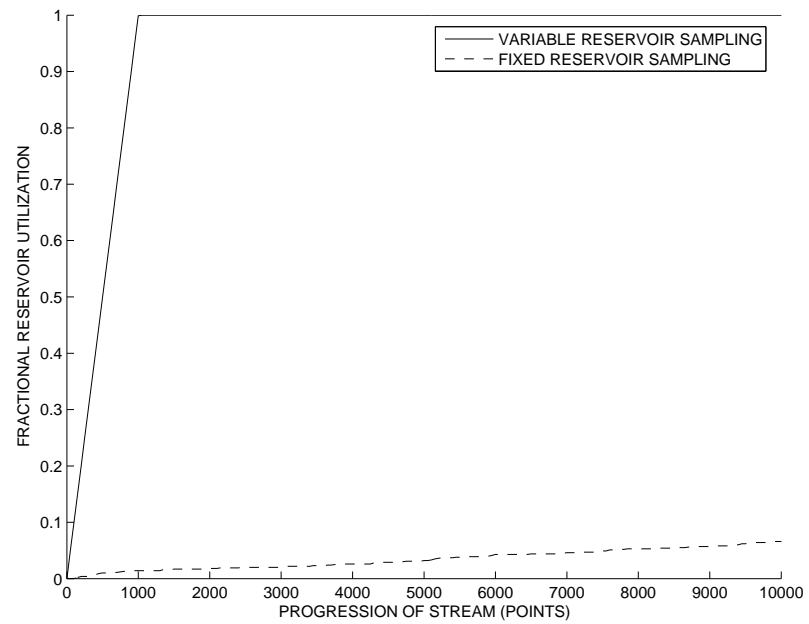
## Result

- If we apply the reservoir sampling algorithm for any number of iterations with parameters  $p_{in}$  and  $\lambda$ , eject a fraction  $1 - (p'_{in}/p_{in})$  of the points, and then apply it again for any number of iterations with parameters  $p'_{in}$  and  $\lambda$ , the resulting set of points satisfies the exponential bias function with parameter  $\lambda$ .

## Observations

- The factor  $q$  by which the reservoir is reduced in each iteration does not affect the method.
- A particularly useful choice of parameters is to start off with  $p_{in} = 1$ , to pick  $q = 1/n_{max}$ , and eject exactly one point in each such phase.
- Extremely fast fill up: almost like deterministic policy!

# Effects of Approach on Reservoir Initialization



# Applications

- Can be directly used for applications in which bias is required.
- What if we want unbiased results from a given application?
- Reservoir sampling provides this flexibility.
- Assume function of the form:

$$G(t) = \sum_{i=1}^t c_i \cdot h(\overline{X}_i) \quad (5)$$

# Query Resolution

- Function  $G(t)$  is general enough to work for *count* and *sum* queries.
- Define random variable  $H(t)$  as follows:

$$H(t) = \sum_{r=1}^t (\mathcal{I}_{r,t} \cdot c_r \cdot h(\bar{X}_r)) / p(r, t)$$

- $\mathcal{I}_{r,t}$  is an indicator function depending upon whether or not the  $r$ -th point is included in reservoir at  $t$ .
- $H(t)$  can be easily estimated from the reservoir.

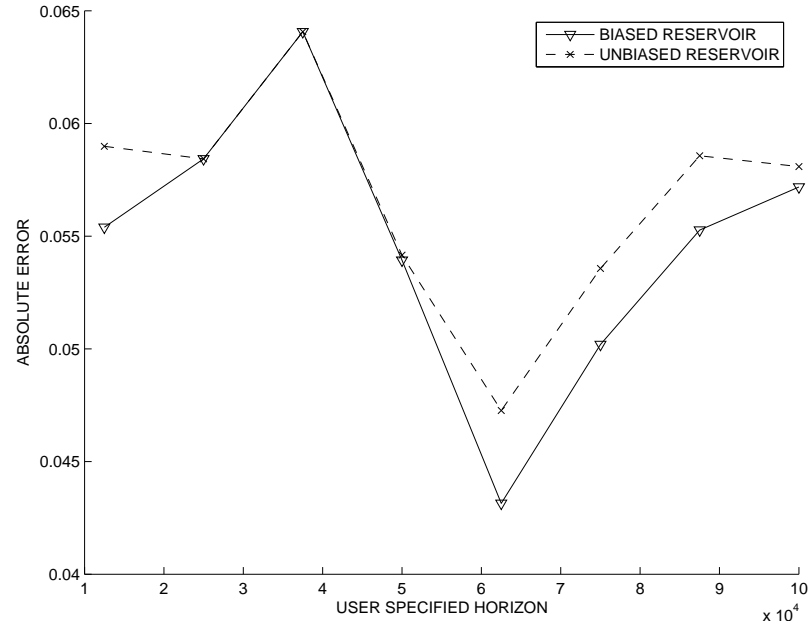
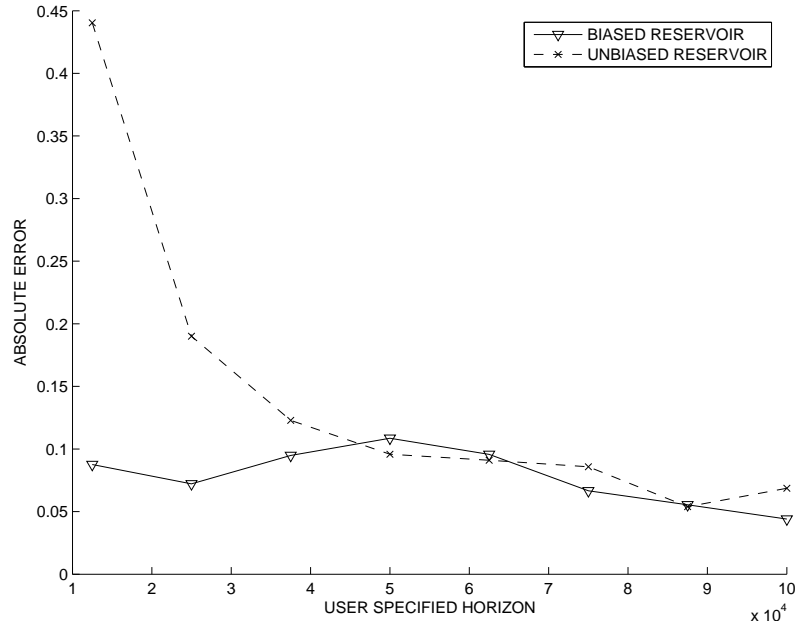
# Results

- $E[H(t)] = G(t)$
- $Var[H(t)] = \sum_{r=1}^t K(r, t)$   
where  $K(r, t) = c_r^2 \cdot h(\bar{X}_r)^2 \cdot (1/p(r, t) - 1)$
- **Key Observation:** 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.

# Experimental Results

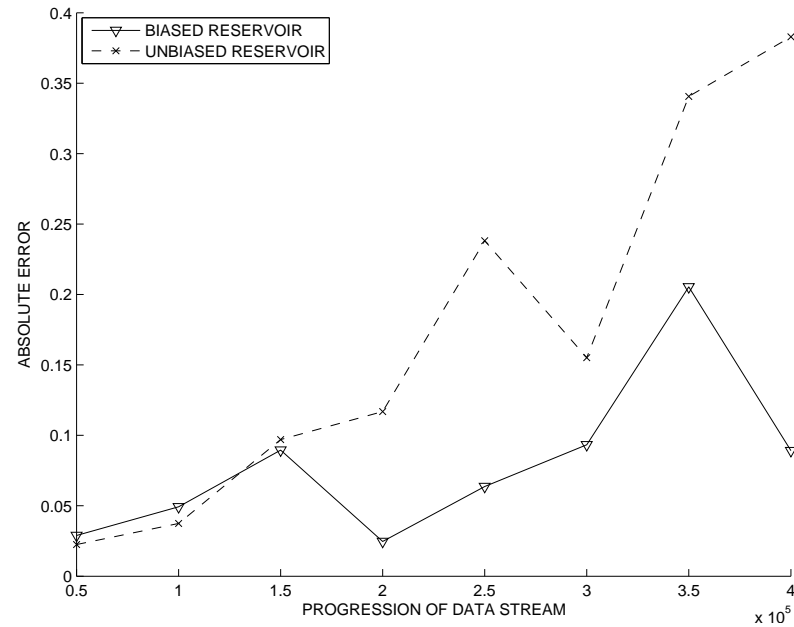
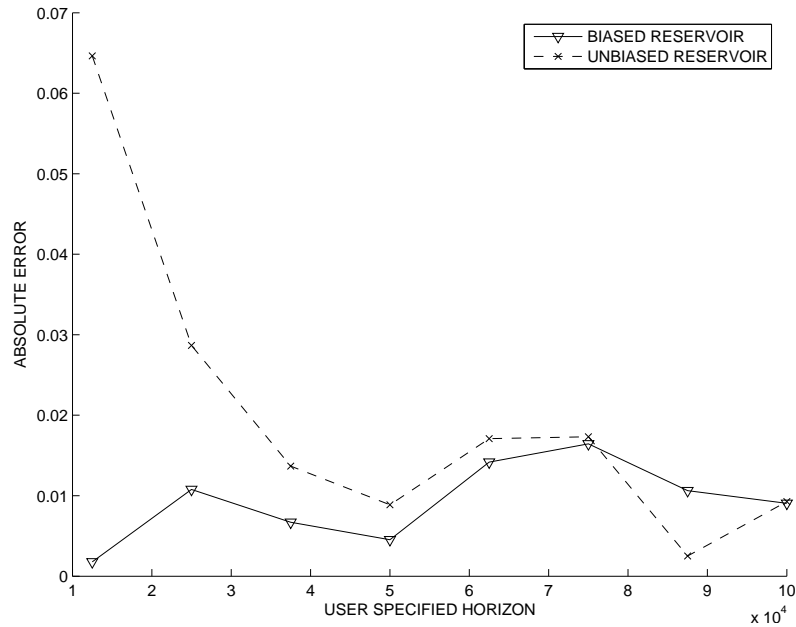
- Network Intrusion Data Set
- Clustered Synthetic Data Set with Drifting Clusters
- Applications (1) Query Processing (2) Classification (3) Evolution Analysis

# Query Estimation



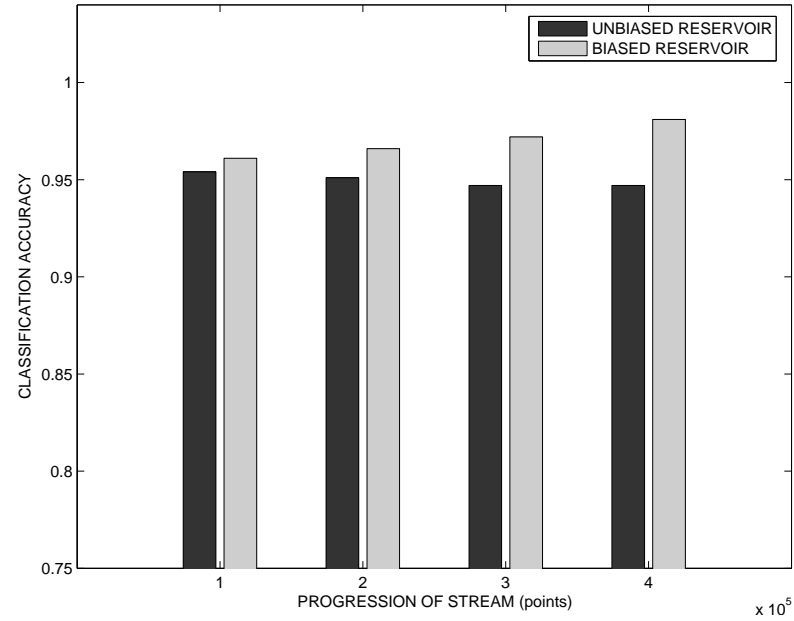
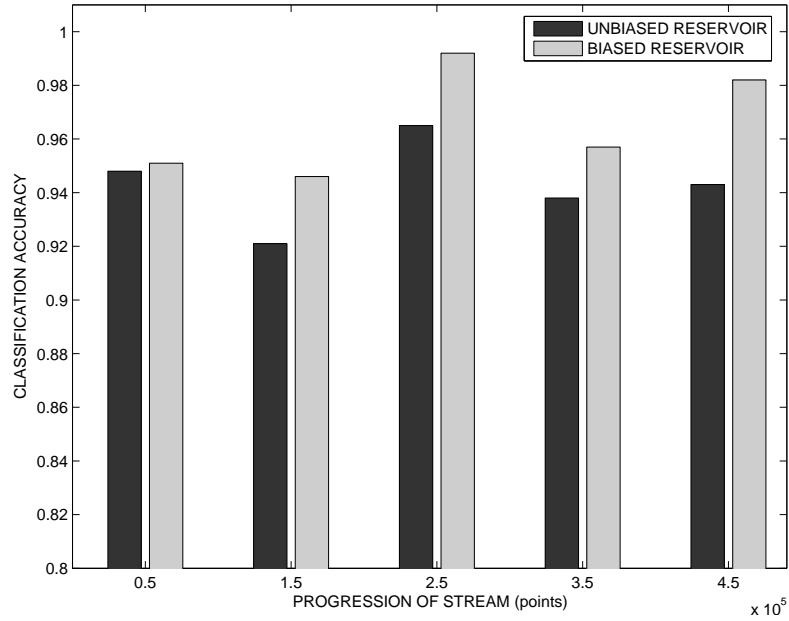
- Sum Query
- Count Query

# Query Estimation



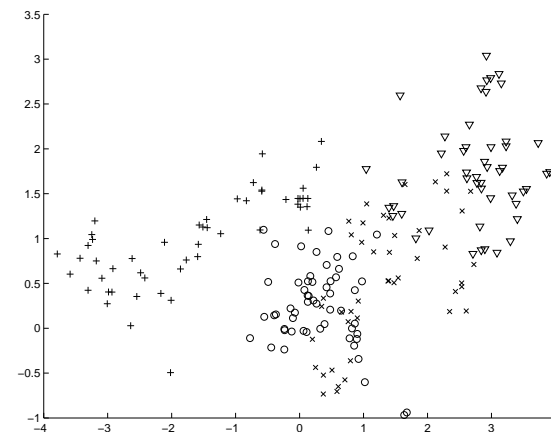
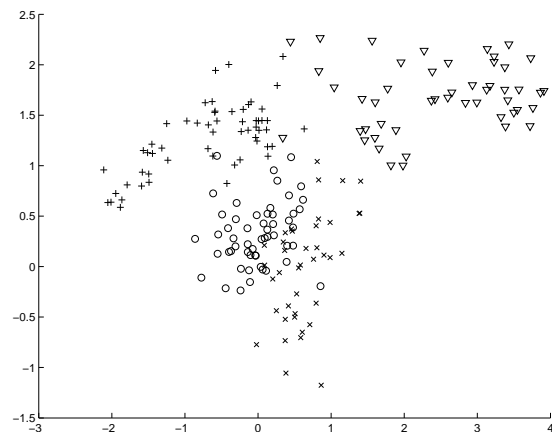
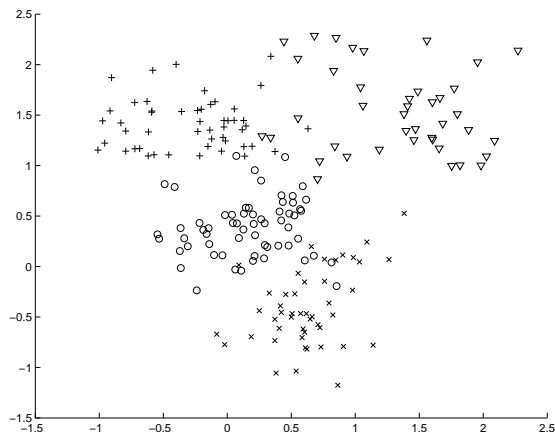
- Range Query
- Query Effectiveness with Stream Progression

# Classification



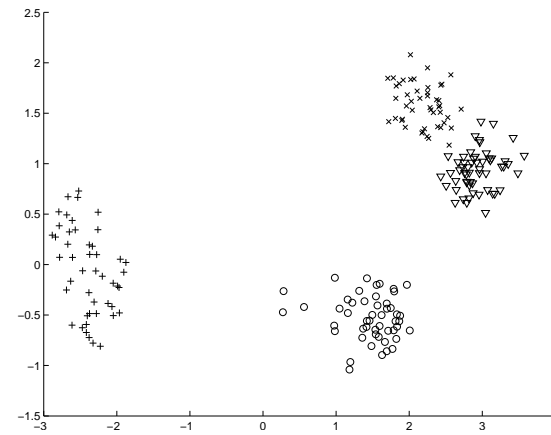
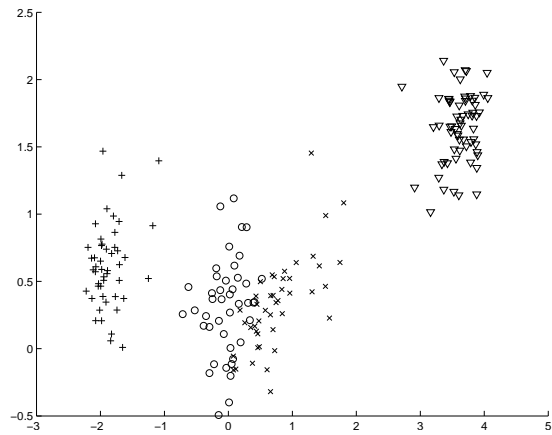
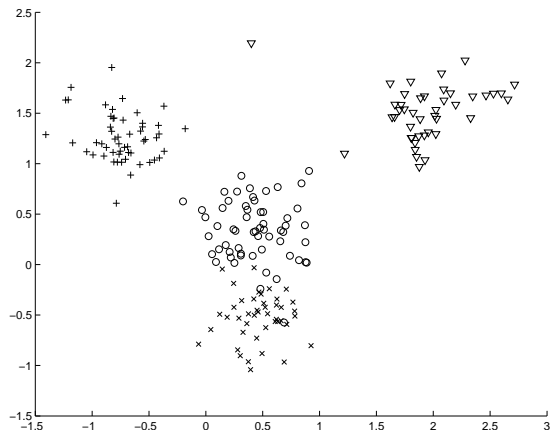
- Network Intrusion
- Synthetic Data

# Evolution Analysis



- Unbiased Reservoir Sampling

# Evolution Analysis



- Biased Reservoir Sampling

## Conclusions and Summary

- Technique for biased reservoir sampling.
- Quality does not degrade with progression of the stream.
- Useful for both biased and unbiased applications.