Finding Frequent Items in Sliding Windows with Multinomially-Distributed Item Frequencies*
(University of Waterloo Technical Report CS-2004-06, January 2004)

Lukasz Golab, David DeHaan, Alejandro López-Ortiz Erik D. Demaine

University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
{*lgolab,dedehaan,alopez-o*}*@uwaterloo.ca*

M.I.T.
Cambridge, MA, USA 02139
edemaine@mit.edu

Abstract

This paper presents algorithms for identifying frequent items within a sliding window in the data stream computational model, assuming that item types conform to a multinomial distribution. We begin by introducing the drifting data distribution model, which assumes that item type frequencies approximately follow the same distribution in every instance of the sliding window, but gradual changes in the parameters of the distribution are allowed across the lifetime of the stream. Under this model and given the assumption of multinomially-distributed item frequencies, we show how to determine the true frequencies of items using space that is only a fraction of the sliding window size. We then use these approximate frequencies to identify items that exceed a given frequency threshold. Our algorithms are shown to outperform classical inference based on random sampling from the sliding window.

1 Introduction

A data stream is a real-time, continuous, ordered sequence of items generated by sources such as sensor networks, Internet traffic flow, credit card transaction logs, or on-line financial tickers. In the last several years, it has been shown that the unique properties of data streams—virtually unbounded length, fast arrival rate, and lack of system control over the order in which items arrive—generate many interesting research problems in algorithm analysis and data management; see [9] for a recent survey. One such problem concerns maintaining statistics over the recently arrived portion of a stream in order to identify emerging trends. This may be accomplished by monitoring item types or categories that occur frequently, either above some threshold or in the top k positions when ranked by frequency. To emphasize recent data, applications typically impose a sliding window on the stream and ignore all values that fall outside the window. *Time-based* windows are defined in terms of the last t time units and *count-based* windows are defined in terms of the last N items seen.

If the entire sliding window fits in memory, then computing statistics over the window is trivial. However, there are applications where either the stream arrives arrives so fast that useful sliding windows do not fit in memory (e.g. monitoring Internet traffic on a high-speed link), or computations are carried out locally by embedded devices with severe memory constraints (e.g. sensors). In such cases, the sliding window must somehow be summarized and query results must be approximated on the basis of the available summary information. One solution is to divide the sliding window into sub-windows, called *basic windows* [14], and store only a summary in each basic window. When the newest basic window fills up, it is appended to the sliding window, the oldest basic window is removed, and statistics over the sliding window are recomputed.

The basic window approach applies to *distributive* and *algebraic* aggregates [10], where the data set may be divided into partitions (basic windows), and the final answer may be computed by pre-aggregating the partitions. However, finding frequently occurring items is a *holistic* aggregate [10] because the summary information needed for pre-aggregating the partitions is proportional to the size of the partitions. In other words, there is no obvious rule for merging the basic window summaries in order to obtain a list of frequent items in the sliding window, unless each basic window stores frequency counts for all of its items. For instance, if

*This research is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

each basic window only stores counts of its top three categories, we would completely ignore a frequent item type that consistently ranks fourth in each basic window (false negatives). Furthermore, an item type that appears on the top-k list in one basic window may not appear elsewhere in the window (false positives).

In previous work [8], we showed that false positives and negatives resulting from the above concerns are rare if item types in the sliding window conform to a power-law like distribution, in which case we expect several very frequent categories that will be counted in every basic window. In this paper, we will show that the basic window approach may also be used to find frequent items in sliding windows if item types conform to a multinomial distribution in each instance of the window. Our specific contributions are as follows.

- We classify distribution models for sliding windows and propose a drifting data model, which, combined with the assumption of multinomially-distributed item types, allows us to use the basic window technique for identifying frequent items.
- We propose three algorithms, MoreFrequentItem, MostFrequentItem and OverThreshold, for identifying frequent items within a stream adhering to our data model. The first two identify the most frequent item in the window, but estimating frequency or even bounding the error in the identification is shown to become infeasible as the number of item types grows. OverThreshold identifies all items over a specified threshold frequency and may be used for frequency prediction with bounded error dependent upon the allocated memory.
- We demonstrate that MoreFrequentItem and OverThreshold outperform classical inference based on random sampling in terms of identifying items over a fixed threshold.

1.1 Background

Finding frequent items is a well-studied problem in the infinite stream model, where the objective is to examine the stream on-line and continuously maintain a set of frequent items using limited memory; see Section 2 of [4] for an overview. Three approaches (and combinations thereof) have appeared in the literature: sampling, counting, and hashing. Random sampling alone is usually insufficient for answering frequent item queries because it may result in large variance when the sampled frequency is used as the estimator of the actual frequency. Counting methods usually allocate a limited number of counters to keep track of selected item frequencies (e.g. [5, 12]), but ensuring that all the frequently occurring items are indeed being monitored is difficult. Clearly, we cannot monitor all the items as this may require space proportional to the window size—consider an instance of a sliding window where one item type occurs twice and the others once. A hybrid sampling-counting approach is a possible solution, where random sampling is used only to select which items are to be assigned counters; once an item is selected, all of its occurrences are counted [6, 7]. Similarly, a hashing-counting solution does not rely on sampling to decide whether to keep a counter for an item type, but instead hashes each item's key to d hash tables and adds a new counter only if all d buckets to which a particular element hashes are large [6].

A simple method for adapting any of the above algorithms to the sliding window model is to store frequent item sets for each basic window, and somehow merge the individual sets to obtain an approximate list of frequent items over the entire sliding window. This approach was used in [1, 8], but, as already mentioned, in the general case we are not guaranteed to obtain meaningful results by merging the frequent item sets from individual basic windows. Maintaining a windowed histogram [13] is another possibility, but it may not be possible to store separate counts for each frequently occurring type. We know of no previous work in the frequent item problem over data streams with multinomially-distributed item frequencies, nor are we aware of previous research in modeling changes in distribution across the lifetime of a data stream.

1.2 Organization

The remainder of this paper is organized as follows. Section 2 introduces our distribution model and assumptions, Section 3 describes the MoreFrequentItem and MostFrequentItem algorithms, Section 4 introduces the algorithm OverThreshold, Section 5 compares the prediction error of our algorithms with random sampling and inference for proportions, Section 6 discusses possible extensions of our work, and Section 7 concludes the paper.

2 Distribution Models for Data Streams and Sliding Windows

We begin by defining a series of item type distribution models for data streams and explain their consequences in the sliding window model. The most general model, which we call *adversarial*, assumes no under-

lying distribution; an adversary is free to choose the category of each item in the stream. In this model, the space complexity of the frequent item problem is proportional to the size of the data set. Hence, it is intractable to maintain a set of frequent items in limited memory over the entire stream, but it is feasible to do so over a sliding window that fits in the allowed memory. The basic window model may be used to maintain distributive and algebraic statistics over sliding windows in limited space, even in the most general adversarial model. For holistic aggregates in the adversarial model, we conjecture that the best possible strategy is to re-execute a non-streaming algorithm periodically over the state of the sliding window at the given time.

In the *stochastic* distribution model, a stream is assumed to contain an arbitrary probability distribution of category frequencies with the characteristic that the order in which the categories occur is uniformly random. The parameters and type of the distribution are assumed to remain fixed across the lifetime of the stream. The *fixed stochastic* model is a variation that explicitly specifies the type of the distribution. All of the previous work on finding frequent items in data streams assumes a stochastic model (the algorithm presented in [8] was in fact designed to work in the adversarial model, but performs well in practice only in the fixed stochastic model with a power law distribution).

In the stochastic model, maintaining a sliding window measures the variance of the distribution in the current window relative to the expected frequencies. To account for permanent changes in the expected frequencies, we propose the *drifting* distribution model, in which the relative frequencies of the categories are assumed to vary over the lifetime of the stream, provided that they vary sufficiently slowly that for any given sliding window of size N excerpted from the stream, with high probability the window could have been generated by a stochastic model. The *fixed drifting* model can also be defined by specifying the type of distribution. The drifting model allows us to assume a stochastic model within a single sliding window without introducing significant error.

To illustrate the drifting model, consider a category whose relative frequency $p(t)$ is a function of time. As shown in Figure 1, for a (time-based) window of size N, we require that $p(t)$ vary by at most δ in all intervals $[t - N + 1, \ldots, t]$, for a "small" value of δ. That is, we allow small variations in the dispersion of the categories within the sliding window, in turn allowing the frequency of each category to change over the lifetime of the stream. This definition is somewhat similar to *Lipschitz* functions of bounded variation (see, e.g. [3]). A function $f : \mathcal{R} \to \mathcal{R}$ is called a *Lipschitz* function if there exists a constant $M > 0$ such that $|f(y) - f(x)| \leq M|y - x|$ for all $x, y \in \mathcal{R}$.

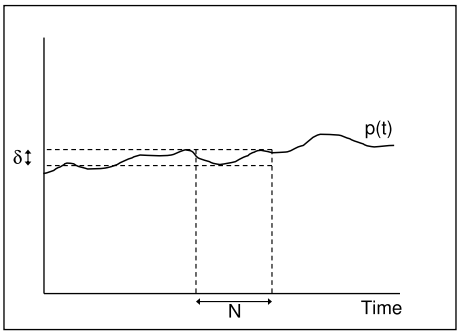


Figure 1: Pictorial representation of the drifting distribution model.

In this paper, we propose algorithms that identify frequently occurring items in sliding windows in the fixed stochastic and fixed drifting models, given a multinomial distribution of item types. The input is assumed to consist of a multiplexed stream containing many distinct categories generated from a multinomial distribution, with the relative probabilities unknown a priori and possibly changing slowly across the lifetime of the stream. We assume that the variation in the underlying relative frequencies across a window of size N does not contribute to the approximation error. The available storage is only a fraction of the sliding window size and items may be seen only once in order of arrival. We anticipate that the stream arrives at a high rate, so only a constant number of operations (amortized) is allowed for each item.

3 Identifying the Most Frequent Item

In this section, we present an algorithm for identifying the most frequent item in a sliding window that conforms to a fixed stochastic or fixed drifting model with a multinomial distribution. We show that the algorithm works well when only two categories are present, but becomes computationally expensive as the number of categories increases. In the next section, we modify the algorithm to instead identify items occurring above a given threshold, and we show that solving this problem is significantly easier. We begin by considering count-based windows and extend our approach to time-based windows in Section 6.1.

3.1 Two Categories

In the simplest case of two item types (binomial distribution), call them x and y, whose actual rela-

tive frequencies p_x and p_y sum to one, we wish to determine which of the two types occurs in the window with higher frequency and give an estimate for that frequency. Using the basic window approach, a simple exact algorithm is to store counters that contain the difference of the number of x-items versus the number of y-items in each basic window. Summing the counters over all basic windows gives the difference in the observed counts, from which percentage frequencies may be obtained if the size of the sliding window is known. In what follows, we show that if the sliding window conforms to a binomial distribution, we need only record the identity of the more frequent item in each basic window to estimate item frequencies.

3.1.1 A Simple Algorithm

The following algorithm divides the sliding window of size N into a set of n equally-sized basic windows, each of which is summarized by an entry in a queue. Statistics are refreshed every $b = N/n$ items.

Algorithm MoreFrequentItem

1. Initialize global counters f_x and f_y to zero
2. Repeat:
   (a) Initialize local counters l_x and l_y to zero
   (b) For each element e in the next b elements:
   If e is of type x, increment l_x,
   Otherwise, increment l_y
   (c) Add a summary containing the type of the "winner" (larger local counter) to the back of queue Q, and increment the corresponding global counter
   (d) If $sizeOf(Q) > N/b$:
      i. Remove the summary from the front of Q and decrement the corresponding global counter
      ii. Output the identity and value of the larger global counter

Since a single bit can identify the "winner" between two categories, algorithm MoreFrequentItem requires $O(N/b)$ space and $\Theta(1)$ amortized time.

Each time a basic window is filled, MoreFrequentItem outputs the identity of the item expected to be more frequent in the sliding window. Suppose that the output item is x. The algorithm also supplies a frequency f_x of the basic windows dominated by x. However, it is not immediately clear how f_x is related to the actual relative frequency p_x of item x.

Proposition 1. Consider the random variable w defined as follows

$$w = \begin{cases} 1 & \text{if } x \text{ is more frequent in a basic window} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Then, w constitutes a Bernoulli variable.

Proof. The probability of success is the same for all basic windows as they all have the same size. Success occurs with probability B_x equal to the probability that type x is more frequent than type y within the basic window. That is, B_x is the probability that type x occurs $\lceil \frac{b}{2} \rceil$ or more times in a basic window of size b, given by Equation (2); recall that item types x and y conform to a binomial distribution, where the probability that a given element e belongs to type x is p_x and the probability that e belongs to type y is $p_y = 1 - p_x$. Failure occurs with probability $1 - B_x$.

$$B_x = \sum_{i=\lceil \frac{b}{2} \rceil}^{b} \binom{b}{i} p_x^i (1 - p_x)^{b-i} \tag{2}$$

□

Corollary 1. Since the probability of type x winning in any one basic window is independent of its probability of winning in any other basic window, the sum of n Bernoulli variables as defined in Equation (1) is a binomial variable with parameters n and B_x.

The frequency f_x output by MoreFrequentItem may be used to calculate an observed relative frequency $\hat{B_x}$ that x is the winner of a basic window:

$$\hat{B_x} = f_x / n. \tag{3}$$

This value can then be substituted in Equation (2) in order to obtain $\hat{p_x}$, the expected relative frequency of item x. Unfortunately, Equation (2) cannot be solved in closed-form for $\hat{p_x}$, so numerical methods must be used in order to obtain a value for $\hat{p_x}$ for a given $\hat{B_x}$.

3.1.2 Bounding the Error

We will make use of the following result due to Hoeffding [11]. Consider a sample of n items from a binomial distribution and an observed frequency of f. The following is Hoeffding's bound on the deviation of the observed frequency from the true frequency p.

$$\mathbf{Pr}\left\{ \frac{f}{n} - p \geq \Delta \right\} \leq e^{-2n\Delta^2} \tag{4}$$

We assume that the numerical methods used to obtain $\hat{p_x}$ from $\hat{B_x}$ are not a significant source of error; therefore, the primary source of error stems from the quality of $\hat{B_x}$ as an estimate for B_x. Now, B_x is a

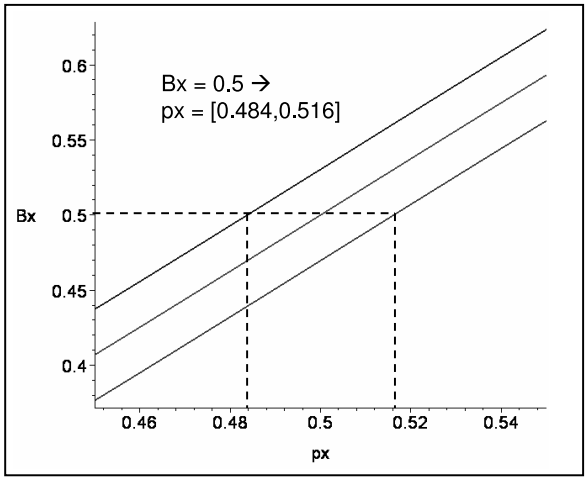


Figure 2: Solving for $\hat{p_x}$ numerically.

binomial random variable (by Corollary 1, f_x is a binomial random variable, and B_x is simply a normalized form of f_x). Using the Hoeffding bound along with a symmetry argument gives the following.

$$\mathbf{Pr}\left\{(\hat{B_x} - \Delta) \leq B_x \leq (\hat{B_x} + \Delta)\right\} > 1 - 2e^{-2n\Delta^2} \quad (5)$$

The right-hand side is the confidence level, so by setting it equal to the desired confidence (e.g. 0.95) we can solve for Δ (note that n is fixed by the choice of b). Because B_x in Equation (2) increases monotonically with p_x, we can find lower and upper bounds for p_x by numerically computing solutions to Equation (2) for the points $B_x = (\hat{B_x} - \Delta)$ and $B_x = (\hat{B_x} + \Delta)$, respectively. This process is illustrated in Figure 2, with the three lines corresponding to $\hat{B_x} - \Delta$, $\hat{B_x}$, and $\hat{B_x} + \Delta$. As shown, a $\hat{B_x}$ value of 0.5 generates a 95% confidence interval for p_x of $[0.484, 0.516]$.

It should be noted that because the basic window size b occurs in the bounds of the summation in Equation (2), the choice of b has a large impact on the error in predicting p_x. As b increases, the following behaviour may be observed.

1. The prediction error Δ surrounding $\hat{B_x}$ increases because n, the number of basic windows used to make the prediction, decreases.

2. The graph of B_x vs. p_x degrades from a linear function to a step function centered at $p_x = 0.5$.

Figure 3 demonstrates the effect of changing b for a window of size $N = 10000$. It shows the curve $\hat{B_x}$ as a function of p_x, along with the curves $\hat{B_x} - \Delta$ and $\hat{B_x} + \Delta$ that bound the 95% confidence region. The three graphs demonstrate the following values of b: (a) 5 (b) 50 (c) 500.

The observation that $\hat{B_x}$ as a function of p_x degrades to a step function with increasing b is crucial for characterizing the effect of basic window size on prediction error. For small values of b, algorithm MoreFrequentItem predicts a wide range of values for p_x, whereas for large values of b, the useful prediction range for p_x is very small. However, the prediction error immediately about the point $p_x = 0.5$ remains tight as b grows. The net effect is that as the choice of basic window size ranges from 1 to N, MoreFrequentItem's usefulness as a frequency predictor diminishes, but its accuracy as a Boolean test for identifying the majority item remains. Since the algorithm's space usage is inversely proportional to b, we conclude that there is a direct tradeoff between space and the accuracy of the frequency prediction, but the simple identification of the majority item does not fully illustrate this tradeoff.

3.2 Multiple Item Types

Algorithm MoreFrequentItem from the previous section may be modified as follows to identify the most frequent item among d categories.

Algorithm MostFrequentItem
Repeat:

1. For each element e in the next b elements:
   If a local counter exists for the type of element e, increment it,
   Otherwise, create a new local counter for this element type and set it equal to 1

2. Add a summary S containing the type of the "winner" (largest local counter) to the back of queue Q

3. Delete all local counters

4. If a global counter exists for the type named in S:
   Increment the global counter,
   Otherwise, create a new global counter for this element type and set it equal to 1

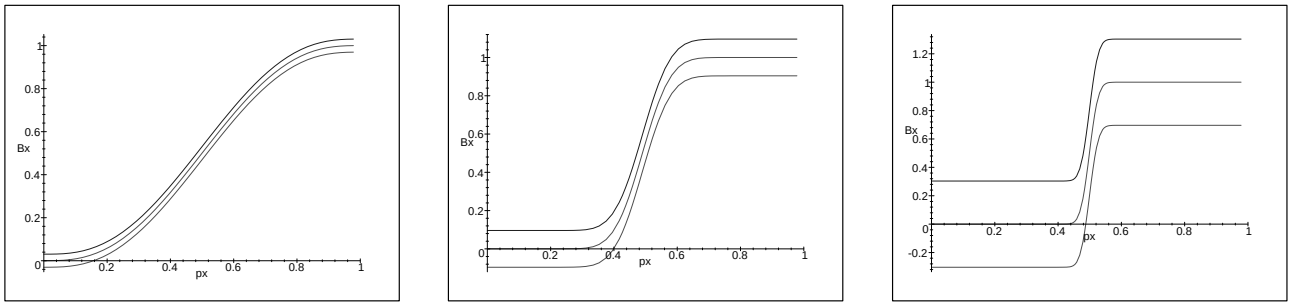
5. If $sizeOf(Q) > N/b$:

   (a) Remove the summary from the front of Q and decrement the corresponding global counter

   (b) Delete the counter if its size reaches zero

   (c) Output the identity and value of the largest global counter

The space requirement of algorithm MostFrequentItem consists of two parts: the working space



(a) (b) (c)

Figure 3: Effect of basic window size on inference error for $N = 10000$ and (a) $b = 5$, (b) $b = 50$, and (c) $b = 500$.

needed to create a summary for the current basic window, and the storage space needed for the summaries of the basic windows. In the worst case, the working space requires $\min(b, d)$ local counters of size $\log b$. For storage, there are N/b summaries each requiring $\log d$ bits. There are also at most N/b global counters of size $\log(N/b)$. This gives a total space bound of $O(\min(b, d)\log b + \frac{N}{b}\log d + \frac{N}{b}\log\frac{N}{b})$. The time complexity of MostFrequentItem is $O(b)$ for each pass through the outer loop. Since each pass consumes b arriving elements, this gives $O(1)$ amortized time per element.

The largest weakness of this algorithm lies in the intractability of using the output value f_i in order to estimate the relative frequency p_i of the most frequent item i. In fact, even just bounding the error on the identity of i is intractable for large d. Consider the case of three item types x, y, and z. In the case of two item types, B_x in Equation (2) was constructed by summing the probabilities of all possible cases where x was in majority within a basic window. These cases were easily identified as exactly those where x occurred at least $\lceil\frac{b}{2}\rceil$ times. However, in the case of three categories, the test $count(x) \geq \frac{b}{3}$ is a necessary but not sufficient criterion for identifying a majority by x, because x's majority also depends on its count being greater than both y and z. This gives rise to the equation

$$B_x = \sum_{i=\lceil\frac{b}{3}\rceil}^{b} \binom{b}{i} p_x^i \left\{ \sum_{j=0}^{b-i} \binom{b-i}{j} p_y^j p_z^{b-(i+j)} - \sum_{j=i+1}^{b-i} \binom{b-i}{j} \left[p_y^j p_z^{1-(i+j)} + p_y^{1-(i+j)} p_z^j \right] \right\} \quad (6)$$

with analogous equations existing for B_y and B_z. In order to compute p_x given estimates for B_x, B_y and B_z, we must solve a non-linear system of two equations and two unknowns (the third equation is eliminated by rewriting p_z in terms of p_x and p_y).

In the general case of d item types, to estimate p_i we must solve a non-linear system of $d-1$ equations and $d-1$ unknowns, where the number of terms within each equation grows combinatorially in d. Even if we restrict the problem to bounding the prediction error in the identification of i as the most frequent item, we cannot translate the width of the Hoeffding-bounded error surrounding $\hat{B}_i$ to a range surrounding $\hat{p}_i$ without solving the entire system.

Because the *Most Frequent Item Problem* is a simplification of the more general *Top-k Problem*, the above results demonstrate that it is infeasible to extrapolate a solution to the top-k problem with bounded error using only a set of sub-solutions (top-k lists for portions of the total window) and the assumption of a multinomial distribution of item categories.

4 Identifying Over-Threshold Items

The complexity involved in using algorithm MostFrequentItem is due to the interdependence among item types inherent in the concept of a winner for each basic window. Because of the dependencies involved in the creation of the stored top-k lists, we cannot use these lists to solve for the relative frequency of one type without simultaneously solving the entire system. If we wish to solve for the frequencies of only selected categories, we must eliminate the dependencies within the frequent item lists in individual basic windows. One way to introduce independence is to replace the concept of *winner* (implying comparison among peers) with *achiever* (implying comparison against an external standard). As a consequence, rather than each basic window resulting in exactly one winner,

each basic window may result in the recognition of zero or more achievers. The following algorithm employs a user-defined threshold $1/m$ to create an items-over-threshold list for each basic window.

Algorithm OVERTHRESHOLD
Repeat:

1. For each element e in the next b elements:
   If a local counter exists for the type of element e, increment it,
   Otherwise, create a new local counter for this element type and set it equal to 1
2. Add a summary S containing the element types of all local counters $\geq b/m$ to the back of queue Q
3. Delete all local counters
4. For each type named in S:
   If a global counter exists for this type, increment it,
   Otherwise, create a new global counter for this element type and set it equal to 1
5. If $sizeOf(Q) > N/b$:
   (a) Remove the summary S' from the front of Q
   (b) Decrement the global counters for all element types named in S'
   (c) If a counter is decremented to zero, delete it
   (d) Output the identity and value of all global counters greater than some threshold τ

The space complexity of algorithm OVERTHRESHOLD is at most m times worse than MOSTFREQUENTITEM, with a worst case bound of $O(\min(b,d)\log b + \frac{mN}{b}\log d + \frac{mN}{b}\log\frac{N}{b})$ where d is the total number of item types in the system. The time complexity is $O(\min(m,b)+b)$ per iteration of the outer loop, which still yields $O(1)$ amortized time.

We now proceed to resolve two issues related to algorithm OVERTHRESHOLD. Firstly, we identify the relation between the frequency f_x output by the algorithm and the true relative frequency p_x. Secondly, we investigate how to calculate the required value of τ used in step 5(d) of the algorithm. We first note that, as in section 3.1, we can define a Bernoulli random variable

$$w = \begin{cases} 1 & \text{if } count(x) \geq b/m \text{ in a basic window} \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

whose probability of success B_x is given by the sum of the probabilities for all scenarios where x exceeds the threshold. In the construction of Equation (2) in section 3.1 we exploited the fact that "majority" in the case of two categories is equivalent to surpassing a threshold of $\lceil b/2 \rceil$. In case of an arbitrary threshold, our new equation for B_x becomes the following.

$$B_x = \sum_{i=\lceil \frac{b}{m} \rceil}^{b} \binom{b}{i} p_x^i (1-p_x)^{b-i} \quad (8)$$

Observe that unlike Equation (6), this equation relates B_x to p_x without dependence on the relative frequencies of any other items.

To address the first issue, note that f_x induces a value $\hat{B_x} = f_x/n$ which is an approximation for the true B_x of Equation (8). The frequencies $\{f_1, \ldots, f_d\}$ follow a multinomial distribution with parameters n, B_1, B_2, $\ldots$, B_d, so the marginal distribution for f_x (and hence $\hat{B_x}$) follows a binomial distribution. Therefore, we can directly apply the Hoeffding bound from Equation (5) to quantify the error in this approximation. The result is that the observations made in section 3.1.2 regarding the effect of b on the shape of the curve and the error in prediction all directly apply, with the generalization that the step function centers around $p_x = 1/m$ rather than $p_x = 1/2$. Figure 4 demonstrates the three curves (corresponding to $\hat{B_x} - \Delta$, $\hat{B_x}$, and $\hat{B_x} + \Delta$ for the 95% confidence level) associated with the values $N = 10000$, $b = 50$, and $m = 10$.

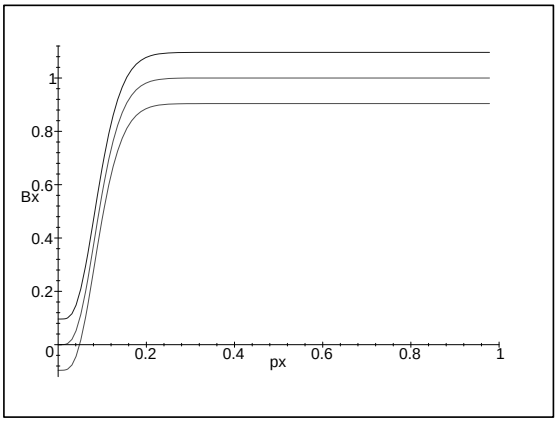


Figure 4: Effect of threshold parameter on inference error.

The accuracy of frequency prediction centers around the relative threshold $1/m$, therefore $1/m$ should be chosen very close to the actual desired reporting threshold. Assume that $1/m$ is the desired threshold. Then, the value for τ should be the expected value for B_x when p_x=$1/m$, which can be calculated by substituting $1/m$ for p_x in Equation (8). This value for τ gives the most likely list of types that have a relative

frequency over $1/m$; however, the solution may contain either false negatives (high frequency types not identified) or false positives (low frequency types incorrectly identified). By adding (subtracting) the value Δ to (from) τ, we can guarantee with the confidence level associated with Δ that the solution contains no false positives (negatives), with the tradeoff that the solution is more likely to contain false negatives (positives).

5 Comparison with Random Sampling

We are interested in comparing the accuracy in identifying high-frequency categories between our algorithms and classical inference for proportions. Let $\hat{p}$ be the sample proportion (observed count divided by the sample size n). The interval within which the true proportion p lies may be calculated as follows.

$$p \in [\hat{p} - z^* S, \hat{p} + z^* S] \tag{9}$$

The value of z^* is the percentile of the standard normal distribution that corresponds to a given confidence level ($z^* = 1.960$ for 95% confidence, $z^* = 2.576$ for 99% confidence). S is the standard error of the sample given by the following equation.

$$S = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \sqrt{\frac{N-n}{N-1}} \tag{10}$$

The second term is the finite population correction factor because we are sampling from a population size equal to the sliding window size N (recall our assumption that each sliding window conforms to a multinomial distribution). This inference method relies on the normal approximation to binomial distributions and may be used if $np \geq 5$ and $n(1-p) \geq 5$.

In our experiments, the error metric is taken to be the maximum expected error when the sample proportion is equal to the threshold at the 95% confidence level. For instance, in the two-category majority case, we compare the range of p that algorithm MoreFrequentItem returns when $\hat{B}_x = 0.5$ with the confidence interval predicted by Equations (9) and (10) for $\hat{p} = 0.5$. We fix N, the size of the sliding window, to be 10000, and investigate the consequences of increasing the basic window size b, (or equivalently, decreasing k, the number of basic windows). To ensure fairness, we allow the random sampling algorithm to utilize the same amount of memory that our algorithms require in the worst case. Furthermore, we undercharge the random sampling algorithm by ignoring the space costs associated with maintaining a windowed random sample (see [2] for more details regarding these costs).

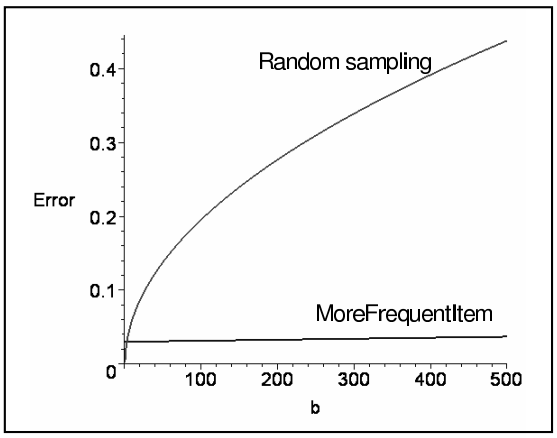


Figure 5: Prediction error of algorithm MoreFrequentItem and random sampling.

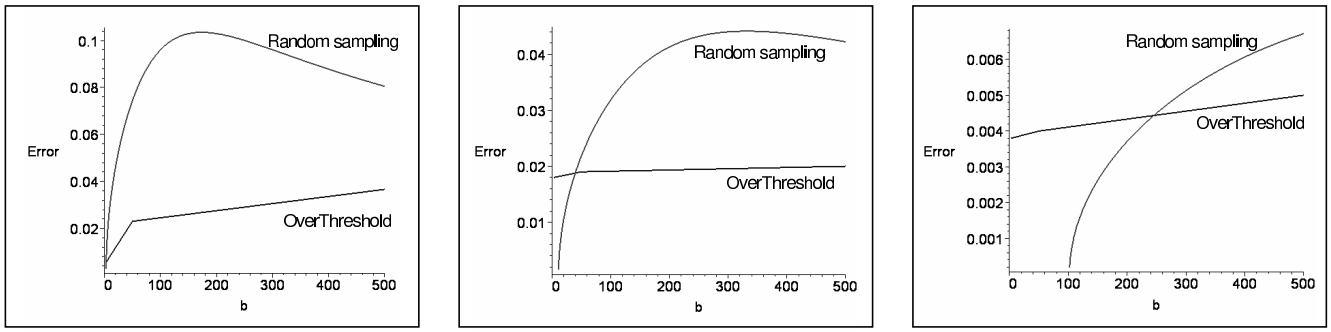
5.1 Algorithm MoreFrequentItem

First, we test algorithm MoreFrequentItem in the role of an identifier of the majority between two categories. Figure 5 compares the error (i.e. the length of the interval within which the true value of p_x lies with 95% confidence) as a function of b for algorithm MoreFrequentItem and classical inference from a random sample. Our algorithm outperforms classical inference for all values of b. For instance, if $b = 100$, the algorithm's error is only one-fifth of the error in random sampling. As the value of b approaches 400, our algorithm's advantage in minimizing the error reaches one order of magnitude. As seen in Figure 3 in Section 3, increasing b has little effect on the approximation error of MoreFrequentItem at the decision point, while at the same time reducing the space requirements (and increasing the refresh delay). In contrast, random sampling performs increasingly poorly as b gets large because the ratio of the sample size to the population size decreases.

5.2 Algorithm OverThreshold

We compare the worst-case performance of algorithm OverThreshold with classical inference for many categories with three threshold values: 0.5, 0.1, and 0.01. The value of N remains fixed at 10000 and the confidence level is still 95%. We assume that the number of distinct items d is at least as large as b. Results are shown in Figure 6 for threshold values of (a) 0.5 (b) 0.1 and (c) 0.01.

We first note that the error in classical inference is no longer a monotonically increasing function of b. This is so because the space complexity of algorithm Over-



(a) (b) (c)

Figure 6: Prediction error of algorithm OverThreshold and random sampling.

THRESHOLD depends on b (in the worst case, we need to store the entire current basic window in memory since we assumed that $d \geq b$) and on $\frac{mN}{b}$ (the number of synopses stored times the maximum number of items that may possibly exceed the given threshold of $\frac{1}{m}$). Thus, as b increases, our algorithm must allocate more working storage for the current basic window, which allows the classical inference algorithm to use a larger sample size. This explains why the error in random sampling eventually begins to decrease as b increases, as seen in Figures 6(a) and 6(b).

Our second observation deals with the degradation in the worst-case performance of algorithm OVERTHRESHOLD (relative to random sampling) for very small threshold values. Clearly, a smaller threshold value allows more items to exceed the threshold in a given basic window, thereby increasing the upper bound on the sizes of our synopses. Nevertheless, as seen in Figure 6(a), our algorithm outperforms random sampling when the threshold is 0.5. In Figure 6(b), we see that when the threshold is lowered to 0.1, our algorithm performs better for $b > 25$. In Figure 6(c), further decreasing the threshold to 0.01 leads to a value of $b > 250$ for which algorithm OVERTHRESHOLD is more precise than random sampling.

It should be noted that these results represent the worst-case behaviour of algorithm OVERTHRESHOLD, where the maximal number of items exceeds the threshold in a given basic window, and must be recorded. Relaxing this condition leads to a relative improvement in the performance of our algorithm versus random sampling. In the "best" case of only two categories, we only require one counter in order to decide which item type was more frequent within the current basic window. Thus, the amount of memory available to store a random sample is smaller and our algorithm enjoys a greater relative advantage (recall Figure 5).

6 Possible Extensions

6.1 Time-based Sliding Windows

Our algorithms are applicable to time-based windows, where basic windows of possibly different sizes span equal time intervals, due to the following result from probability theory.

Theorem 1. A Poisson trial a_i is a success with probability p_i and failure with probability $1 - p_i$. Suppose that A is the sum of n independent Poisson trials a_i with probabilities p_i for $1 \leq i \leq n$. Hoeffding's theorem states that A may be upper-bounded by a binomial random variable B with parameters n and $p = \frac{1}{n}\sum_{i=1}^{n} p_i$.

6.2 Top-k Estimation using Counts

Recall that algorithm OVERTHRESHOLD finds items that occur with frequencies exceeding a user-defined threshold. The following is a possible extension that computes a list of the k most frequent items. Consider the general case of d distinct flows and some threshold τ. In addition to storing the boolean information of whether or not an item exceeded τ in a given basic window, we also store the counts of all the items above τ. After computing the list of all the items that exceed the threshold in the entire window, if there are more than k such items, then we increase the threshold slightly and eliminate all the items whose counts do not exceed the new threshold. We continue this procedure until there are exactly k items left.

The above suggests a more general approach of assigning different thresholds for various item types. That is, for item types x, y, z and w, we could choose to include item x on our above-the-threshold lists only if its relative frequency is above 0.4 and include other items if their frequencies are above 0.35. This would

be an appropriate strategy if we knew that x is slightly more popular than the other item types. This method could be improved by incorporating feedback from recent sliding windows and deciding whether to increase or decrease thresholds for various items.

6.3 Reducing Space Usage

We propose two extensions of algorithm OVER-THRESHOLD that reduce space usage: randomly sampling items to be stored in the synopses and deleting parts of older synopses if a particular item has already exceeded the global threshold. In the first approach, if an item exceeds the threshold in a given basic window, we flip a biased coin and store the item with probability h and ignore it with probability $1 - h$. This scheme reduces space and does not affect the running time, but it introduces an additional source of error. This demonstrates an interesting tradeoff between using space in order to straighten out the error curve (as in Figure 3, improving the range of p_x that can be predicted) and using space to tighten the prediction error within the usable part of the curve.

The second improvement essentially eliminates redundant information and works as follows. Suppose that an item would have to occur on at least 20 out of 100 top-k lists in order to exceed a given threshold. Suppose further that flow x occurs on 60 such lists. If we removed every second occurrence of flow x from the top-k lists, we would still have 30 such occurrences and we would still conclude that x exceeds the threshold (although we could not even attempt an estimation of the true frequency of x). However, this would introduce error for skewed data as the window slides. A better solution would be to remove flow x from the 30 oldest lists on which it occurs, which does not introduce any error into future windows. In either case, this reduction in space comes at a cost of increased processing time to locate the oldest items to delete.

7 Conclusions

We proposed a classification of distribution models for sliding windows over data streams. We used one of our models—the drifting distribution—to develop algorithms for answering frequent item queries (and to some extent for inferring the actual frequencies of items) over multinomially-distributed sliding windows. Our algorithms were shown to outperform classical inference from a windowed random sample.

In future work, we intend to develop sliding window algorithms for other holistic aggregates and refine the drifting distribution model. For the former, we are interested in identifying any special cases that would enable incremental execution of other holistic aggregates (e.g. COUNT DISTINCT) using the basic window technique. For the latter, we want to verify that the drifting distribution model with a small value of δ (the maximum allowed deviation in the probability distribution over a given sliding window) does not add a significant error to our algorithms. Moreover, we would like to further divide the drifting distribution model according to the type of change in the distribution over time. For example, one type of change could be that each item's frequency changes slightly, and another could involve a small number of categories changing significantly and others remaining the same (though some windowed aggregates may be oblivious to the type of change in the distribution). Again, the goal is to find special cases where efficient incremental evaluation of complex statistics over the sliding window is possible.

References

[1] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. Stanford University Comp. Sci. Tech. Report 2003-72.

[2] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. *SODA'02*, pp. 633–634.

[3] A. Bruckner, J. Bruckner, and B. Thomson. *Real Analysis.* Prentice Hall, 1997.

[4] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. *PODS'03*, pp. 296–306.

[5] E. Demaine, A. Lopez-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. *ESA'02*, pp. 348–360.

[6] C. Estan and G. Varghese. New directions in traffic measurement and accounting. *SIGCOMM Internet Measurement Workshop '01*, pp. 75–80.

[7] P. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. *SIGMOD'98*, pp. 331–342.

[8] L. Golab, D. DeHaan, E. Demaine, A. Lopez-Ortiz, and J. I. Munro. Identifying frequent items in sliding windows over on-line packet streams. *SIGCOMM Internet Measurement Workshop '03*, pp. 173–178.

[9] L. Golab and M. T. Özsu. Issues in data stream management. *SIGMOD Record*, 32(2):5–14, 2003.

[10] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. *ICDE'96*, pp. 152–159.

[11] W. Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, 58:13–30, 1963.

[12] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. *VLDB'02*, pp. 346–357.

[13] L. Qiao, D. Agrawal, and A. El Abbadi. Supporting sliding window queries for continuous data streams. *SSDBM'03*, pp. 85–94.

[14] Y. Zhu and D. Shasha. StatStream: Statistical monitoring of thousands of data streams in real time. *VLDB'02*, pp. 358–369.