Taking One Small Step Forward: Finding Low-Frequentcy Items in Data Streams

Michael Shell
School of Electrical and
Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Email: http://www.michaelshell.org/contact.html

Homer Simpson
Twentieth Century Fox
Springfield, USA
Email: homer@thesimpsons.com

James Kirk
and Montgomery Scott
Starfleet Academy
San Francisco, California 96678-2391
Telephone: (800) 555–1212
Fax: (888) 555–1212

***Abstract*—We propose an one-pass algorithm, called BFSS, which can identify items in a data stream with frequencies less than a user specified support. Our algorithm is simple and have provably small memory footprints related to domain of data streams. Although the output is approximate, we can guarantee no false negatives and provably few false positives. Given a little modification, algorithm BFSS can be improved to SBFSS, which can handle low-frequency items detection over data streams form large domain with acceptable and bounded false negatives and false positives using a proper space.**

I. Introduction

In many real-world applications, information such as web click data, stock ticker data, sensor network data, phone call records, and traffic monitoring data appear in the form of data streams. Online monitoring of data streams has emerged as an important research undertaking. Estimating the frequency of the items on these streams is an important aggregation and summary technique for both stream mining and data management systems with a broad range of applications. A variety of algorithms have been proposed to identify frequent items from data streams, *Sticky Sampling* [] and *Space Saving* [] etc.

However, to the best of our knowledge, there are no algorithms identifying low-frequency items over data streams. Low-frequency items, similar to the defination of frequent items, are items frequencies of which are less than a specified threshold S.

Zipf's law refers to the fact that many types of data studied in the physical and social sciences can be approximated with a Zipfian distribution, one of a family of related discrete power law distributions. Fig.1 describes relation between frequency rank and frequency of items that follow a power law distribution over data streams, and we can find that the distinct number of low-frequency items is much larger than the distinct number of frequent items, so under the restriction of limited memory it is much more difficult to identify low-frequency items than to identify frequent items.

Nowadays, Low-frequency items over data streams are becoming more and more important because of the rich information they contain which can be easily understood through the observation of entropy of data streams []. We take one small step forward to maintain low-frequency items over data streams approximately and try to fill up the blank of low-frequency items mining over data streams in this paper.

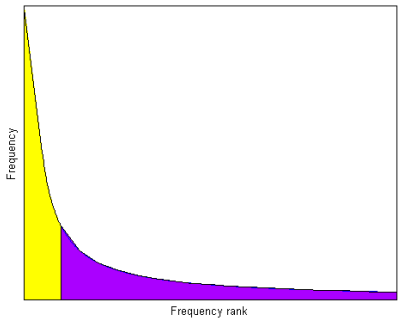

Fig. 1: Rank-frequency distribution

A. Motivating Examples

1) Individual requirements mining: In such an era that information technology develops rapidly, it is much easier for us to get access to various of knowledge and information and we need through a few mouse clicks, for example, we can shop online with the help of e-commerce site, such as Amazon and Taobao etc. We can search whatever we are interested in through search engines, such as Google and Bing etc.

Our requirements are popular most times, for example, buying a regular water glass online or searching the information about a tourist attraction etc, and these popular requirements can be easily met because almost every e-commerce site sell all linds of water glasses and nearly every search engines provide links to popular tourist attractions worldwide.

However, we are no longer saiesfied with responses only to popular demands. For example, it is not so easy for us to buy embroidery stitches online or search information about a nameless small village in China, because these are individual requirements, and some e-commerce sites or search engines may neglect these requirements, for example, i can't find product information about embroidery stitches at Amazon while Taobao does.

Yusuf Mehdi, SVP of Microsoft, once said at the meeting of Search Engine Strategies in 2010 that major reason why

Bing got begind with Google is neglecting "long tail" of search flow items of which appear a few times or even once, and it is of great importance for them to analyze these low-frequency items. So it is individual requirements or in other words low-frequency demands, in some sense, that really make a difference rather than popular demands in areas like e-commerce and SE, and identifying them is the first step to do deep analysis on them.

2) Data distribution estimation:

B. Our Contributions

We introduce and state the proplem of low-frequency items detection over data streams which are of great significance in areas such as individual requirements mining and data distribution estimation, and to the best of our knowledge there is no relative research up to now.

In this paper, we propose *BFSS*, which extends the classic frequent items detection algorithm *Space Saving* to maintain both frequent and low-frequency items in a data stream approximately. The basic idea of our solution is as follows: each item in a data stream is either a frequent one or a low-frequency one once the threshold $\phi[\in (0,1)]$ is set, so we can maintain low-frequency items by filtering the frequent items out. A major problem we have to deal with is to maintain an itemset items in which appear in the data stream, and this can be done approximately using a Bloom filter size of which is related to the size of domain of data streams. *BFSS* gurantee no false negatives and provably few false positives using small memory footprints.

However, size of a Bloom filter must increase with the size of domain in order to keep a low false positive rate, and here comes a problem: what if the domain is too large to be handled by a Bloom filter in a limited space? Inspired by the solution presented in [], we propose *SBFSS* which extends *BFSS* to deal with data streams from large domains, and *SBFSS* gurantee acceptable and bounded false negatives and false positives using a proper space.

C. Roadmap

In Section 2, we present the problem statement and some background on existing approches which deal with the problem of frequent items detection. Our solutions are presented and discussed in Section 3. In Section 4, we discuss how our algorithms can be used in practice. In Section 5, we experimentally evaluate out methods. Conclusions are given in Section 6.

II. PRELIMINARIES

This section presents the problem statement and some representative algorithms solvimg ϵ-*Deficient Frequent Elements* [sticky sampling] which will be formally defined below.Table I summarizes the major notation in this paper.

A. Problem Statement

Consider an input stream $S = e_1, e_2, ..., e_N$ of current length N, which arrives item by item. Let each item e_i belong to a universe set $A = \{a_1, a_2, ..., a_M\}$ of size M representing the input stream's domain. Let $f_S(a)$ denote the number of occurrences of a in S

TABLE I: Major Notation Used in the Paper

Notation	Meaning
BFSS	The name of our first algorithm
SBFSS	The name of our second algorithm
S	The input data stream
A	The domain of S
M	The size of A
M'	The number of distinct elements in S
ϕ	The user specified support threshold
D	The synopsis used in *Space-Saving*
BF	The Bloom filter used in *BFSS*
K	The size of *BF*
C	The number of counters in D
N	The number of elements in the input stream
H	The number of hash functions
h_i	The ith hash function
$f_S(e)$	The frequency of element e in S
FPs	The elements in S with frequency no less than $\lfloor \phi N \rfloor$ wrongly output
FNs	The elements in S with frequency less than $\lfloor \phi N \rfloor$ wrongly neglected
$(e, f(e), \Delta(e))$	The form of each counter in D
E	The set of elements monitored in D
min	The minimum value of $f(e)$ in D

Our problem ϕ-*Bounded Low-Frequency Elements* can be stated as follows: given a data stream S along with its domain A and a user specified threshold $\phi[\in (0,1)]$, output the subset $I(S,\phi) \subset A$ of symbols defined as $I(S,\phi) = \{a \in A : 0 < f_S(a) \leq \lfloor \phi N \rfloor\}$.

At any point of time, *BFSS* output a list of items with the following guarantees:

1) All elements whose true frequency is no more than $\lfloor \phi N \rfloor$ are output. There are *no false negatives*.
2) No element whose true frequency exceeds $2\lfloor \phi N \rfloor$ is output.

As for data streams from large domains which means M is so large that *BFSS* can not handle in mempry, *SBFSS* output a list of items using proper space under the restriction of limited memory with acceptable false negatives and false positives, where a false positive is an item with frequency more than $\lfloor \phi N \rfloor$ wrongly output, and a false negative is an item with frequency no more than $\lfloor \phi N \rfloor$ wrongly neglected.

B. Related Work

As far as we know that there are no related algorithms addressing this problem, however, the methods we propose are based on algorithm solving ϵ-*Deficient Frequent Elements* which can be described as follows: given a input stream S of current length N and a support threshold $\phi \in (0,1)$, return the items whose frequencies are guaranteed to be no smaller than $\lfloor (\phi - \epsilon)N \rfloor$ deterministically or with a probability of at least $1 - \delta$, where $\epsilon \in (0,1)$ is a user-defined error and $\delta \in (0,1)$ is a probability of failure, so we examine several algorithms solving ϵ-*Deficient Frequent Elements*.

research can be divided into two groups: *counter-based* techniques and *sketch-based* techniques.

Counter-Based Techniques keep an individual counter for each element in the monitored set, a subset of A. The counter of a monitored element, e_i, is updated when e_i occurs in the stream. If there is no counter kept for the observed ID, it is either disregarded, or some algorithm-dependent action is taken.

Two representative algorithms *Sticky Sampling* and *Lossy Counting* were proposed in []. The algorithms cut the stream into rounds, and they prune some potential low-requency items at the edge of each round. Though simple and intuitive, they suffer from zeroing too many counters at rounds boundaries, and thus, they free space before it is really needed. In addition, answering a frequent elements query entails scanning all counters.

The algorithm *Space-Saving* , the one we are based at, was proposed in []. The algorithm maintains a synopsis which keeps all counters in an order according to the value of each counter's monitoring frequency plus maximum possible error. For a non-monitored item, the counter with the smallest counts, *min*, is assigned to monitor it, with the items monitoring frequency $f(e)$ set to 1 and its maximal possible error $\Delta(e)$ set to *min*. Since $min \leq \epsilon N$ (this follows because of the choice of the number of counters), the operation amounts to replacing an old, potentially infrequent item with a new, hopefully frequent item. This strategy keeps the item information until the very end when space is absolutely needed, and it leads to the high accuracy of *Space-Saving*. Experiments done in [overview] showed *Space-Saving* outperforms other Counter-Based techniques in recall/precision tests.

Sketch-Based Techniques do not monitor a subset of elements, rather provide, with less stringent guarantees, frequency estimation for all elements using bitmaps of counters. Usually, each element is hashed into the space of counters using a family of hash functions, and the hashed-to counters are updated for every hit of this element. Those representative counters are then queried for the element frequency with less accuracy, due to hashing collisions.

The *Count-Min Sketch* algorithm of Cormode and Muthukrishnan [Count-Min] maintains an array of $d \times w$ counters, and pairwise independent hash functions h_j map items onto $[w]$ for each row. Each update is mapped onto d entries in the array, each of which is incremented. The Markov inequality is used to show that the estimate for each j overestimates by less than n/w, and repeating d times reduces the probability of error exponentially.

The *hCount* algorithm was proposed in []. The data structure and algorithms used in *Count-Min Sketch* and *hCount* shared the similarity, but were simultaneously and independently investigated with different focuses.

III. OUR ALGORITHMS

In this section, we will discuss our approaches *BFSS* and *SBFSS* in detail.

A. The Challenges in ϕ-Bounded Low-Frequency Elements

ϕ-Bounded Low-Frequency Elements has two main challenges due to the different features between frequent items and low-frequency elements over data streams.

The long tail phenomenon. It can be easily proved that there are at most $\lceil 1/\phi \rceil$ frequent elements whose frequency is more than $\lfloor \phi N \rfloor$ in any data stream, however, there is no provable upper bound of the number of low-frequency elements whose frequency is less than $\lfloor \phi N \rfloor$. In fact, our experiments on both real and synthetic data show that low-frequency elements occupy most of the distinct elements appear in data streams, and it is almost impossible to maintain all low-frequency elements in memory. Another observation is that their frequencies are very low and close as well, and it may consume much space to separate low-frequency elements from frequent elements especially when ϕ is very small.

The unpredictability. A basic and common idea of *Counter-Based Techniques* is to discard potential infrequent elements dynamically, and it is based on the fact that potential infrequent elements will never become frequent elements if they don't appear afterwards, however, this fact no longer applies to low-frequency elements in data streams because potential elements will possibly become low-frequency elements if they don't appear afterwards. The unpredictability of low-frequency elements makes it difficult to maintain them directly like frequent elements.

B. The BFSS Algorithm

In consideration of the challenges in *ϕ-Bounded Low-Frequency Elements*, we tried to solve the problem indirectly by filtering frequent items out which is the underlying idea of *BFSS*.

Two algorithms are proposed for updating and outputting final result separately. Algorithm 1 maintains a Bloom filter BF of size K with H uniformly independent hash functions $\{h_1(x), ..., h_H(x)\}$ and a synopsis D with M counters. Each of these H hash function maps an element from A to $[0, ..., K-1]$. Initially each bit of BF is set to 0 and D has M empty counters. Each newly arrived element in the stream is mapped to H bits in BF by the H hash functions and we set the H bits to 1. Then if we observe an element that is monitored in D, we just increment $f(e)$. If we observe an element, e_{new}, that is not monitored in D, handle it depending on whether there is an empty counter in D. If there is one, we just allocate it to e_{new} and set $f(e_{new})$ to 1 and set $\Delta(e_{new})$ to 0. If D is full, we just replace the element that currently has the least hits, min, with e_{new}. Assign $f(e_{new})$ the value $min + 1$ and assign $\Delta(e_{new})$ the value min.

Algorithm 2 checks and outputs the element with frequency less than a user-specified threshold ϕ. For each element in A, we first check whether it is in BF. If the element is not in BF, it must not be a low-frequency item because it never appeared in the stream. If the element is in BF but not in D, it must be a low-frequency item and we output it, and we will give the reason later. If the element appears both in BF and D, we identify the element as a low-frequency element if $f(e) < \lfloor \phi N \rfloor + \Delta(e)$ with high probability. The time requirement of Algorithm 2 is linear to the range of universe. It is acceptable when the frequency of the requests is not high.

C. Analysis of BFSS

In this section, we present a theoretical analysis of *BFSS* described in Section III-B. We analyze the *FNs* and *FPs*, space complexity, and time complexity.

1) ***Analysis of FNs:*** In this section, we will prove that there are no *FNs* in the output of *BFSS*. The proof is based on Lemma 1 to Lemma 5, and the detailed proof of Lemma 1 to Lemma 3 can be found in [].

Algorithm 1 BFSS Update Algorithm

Require: Stream S, support threshold ϕ

$N = 0, c = 0, C = 1/\phi, K = \lambda, H = \mu$; {$N$: length of stream; m: the number of counters used in D; C: the maximum number of counters in D; K: size of Bloom filter; H: number of hash functions;The value of λ and μ will be discussed in detail later.}
Initially synopsis D has M empty counters, each with the form $(e, f(e), \Delta(e))$
for $i = 0$ to $K - 1$ **do**
  $BF[i] = 0$
end for
for each item e of stream S **do**
  for $i = 1$ to H **do**
    $BF[h_i(e)] = 1$
  end for
  if e is monitored in D **then**
    $f(e) = f(e) + 1$;
  else if $c < C$ **then**
    Assign a new counter $(e, 1, 0)$ to it
    $c = c + 1$
  else
    Let e_m be the element with least hits, min
    Replace e_m with e in D
    $f(e) = min + 1, \Delta(e) = min$
  end if
  $N = N + 1$;
end for

Algorithm 2 BFSS Query Algorithm

Require: *BF*, D, ϕ, A, N {A: domain of stream}
Ensure: low-frequency elements with threshold ϕ

$V = |A|$, $flag = true$; {V: size of A; *flag*: indicate whether an element is in *BF* or not}
for $i = 0$ to $V - 1$ **do**
  $flag = true$
  for $j = 1$ to H **do**
    if $BF[h_j(A[i])] == 0$ **then**
      $flag = false$
      break;
    end if
  end for
  if $flag == true$ **then**
    if $A[i]$ is monitored in D **then**
      if $f(A[i]) \leq \lfloor \phi N \rfloor + \Delta(A[i])$ **then**
        output $A[i]$ as a low-frequency element
      end if
    else
      output $A[i]$ as a low-frequency element
    end if
  end if
end for

Lemma 1: $N = \sum_{\forall i | e_i \in E}(f(e_i))$

Proof: Every hit in S increments only one counter by 1 among the M counters which can be easily proved when D has empty counters. It is true even when a replacement happens, i.e., the observed element e was not previously monitored and D has no empty counters, and it replaces another element e_m. This is because we add $f(e_m)$ to $f(e)$ and increment $f(e)$ by 1. Therefore, at any time, the sum of all counters is equal to the length of the stream observed so far. ■

Lemma 2: Among all counters in D, the minimum counter value, min, is no greater than $\lfloor \frac{N}{m} \rfloor$.

Proof: Lemma 1 can be written as:

$$min = \frac{N - \sum_{\forall i | e_i \in E}(f(e_i) - min)}{m} \tag{1}$$

All the items in the summation of Equation 1 are non negative because all counters are no smaller than min, hence $min \leq \lfloor \frac{N}{m} \rfloor$. ■

Lemma 3: For any element $e \in E$, $0 \leq \Delta(e) \leq min$.

Proof: From Algorithm 1, $\Delta(e)$ is non-negative because any observed element is always given the benefit of doubt. $\Delta(e)$ is always assigned the value of the minimum counter at the time e started being observed. Since the value of the minimum counter monotonically increases over time until it reaches the current min, then for all monitored elements $\Delta(e) \leq min$. ■

Lemma 4: An element e with $f_S(e) > min$, must be monitored in D.

Proof: The proof is given in [], however, we think the proof is not very rigorous because it is based on the fact that the true frequency of e must be no more than its estimated frequency, i.e. $f_S(e) \leq f(e)$, but this fact should be based on Lemma 4.

My proof is also by contradiction. Assume $e \notin E$. Then, it was evicted previously, and we assume that it had been monitored in $i(> 0)$ time slots, and e appeared $n_j (0 < j \leq i)$ times in the jth time slot. n_j satisfies:

$$\sum_{j=1}^{i} n_j = f_S(e) \tag{2}$$

We assume that $\Delta(e_j)(> 0)$ denotes the error estimation assigned to e at the start of the jth time slot, and we have the following inequality because the minimum counter value increases monotonically:

$$\begin{gathered} \Delta(e_1)+n_1 \leq \Delta(e_2) \\ \Delta(e_2)+n_2 \leq \Delta(e_3) \\ \cdots \\ \Delta(e_{i-1})+n_{i-1} \leq \Delta(e_i) \\ \Delta(e_i)+n_i \leq min \end{gathered} \tag{3}$$

After adding up the left and right sides of inequality group 3, we can get:

$$\Delta(e_1) + \sum_{j=1}^{i} n_j \leq min \tag{4}$$

We can get $f_S(e) \leq min$ from equation 2 and inequation 4. This contradicts the condition $f_S(e) > min$.

In fact, Lemma 4 can also be stated as: For any element $e \notin E$, $f_S(e) \leq min$. ■

Lemma 5: For any element $e \in E$, $f(e)-\Delta(e) \leq f_S(e) \leq f(e) \leq f(e)-\Delta(e)+min$

Proof: From Lemma 3, we can easily get $f(e) \leq f(e)-\Delta(e)+min$. $f(e)-\Delta(e)$ is the true frequency of e since it was lastly observed, so $f(e)-\Delta(e) \leq f_S(e)$. From Lemma 4, we can find that $\Delta(e)$ over estimated the frequency of e before it was observed ,and it clearly indicates $f_S(e) \leq f(e)$. From the above, we can prove Lemma 5. ■

Theorem 1: There are no *FNs* in the output of *BFSS*.

Proof: We only hava to prove that the elements we don't output contain no low-frequency elements. From Algorithm 1, only two kinds of elements are not ouput: i) elements filtered out by the *BF*. ii) elements monitored in D with $f(e)-\Delta(e) > \lfloor \phi N \rfloor$. The first kind of elements are obviously not low-frequency elements because they never appeared in S. From Lemma 5, we know that the elements monitored in D with $f(e)-\Delta(e) > \lfloor \phi N \rfloor$, must satisfy $f_S(e) > \lfloor \phi N \rfloor$. ■

*2) **Anysis of FPs**:* In this section, we will give a theoretically strict bound of expectation of number of *FPs* in output of *BFSS* regardless of data distribution of S, and a tighter bound can be derived for data stream under power law distribution.

Lemma 6: Any element e with $f_S(e) > 2\lfloor \phi N \rfloor$, must not be output.

Proof: From Lemma 4, we know that any element e with $f_S(e) > 2\lfloor \phi N \rfloor$ must be monitored in D, i.e. $e \in E$. Then from Lemma 5, we can get that elements monitored in D must satisfy $f_S(e) \leq f(e)$, and that is to say any element e monitored in D with $f_S(e) > 2\lfloor \phi N \rfloor$, must satisfy $f(e) > 2\lfloor \phi N \rfloor$. At last from Algorithm 1, we can prove that any element e with $f_S(e) > 2\lfloor \phi N \rfloor$, must not be output. ■

Lemma 7: The probability of a false positive of *BF* is no more than

$$(1-(1-\frac{1}{K})^{HM})^H \tag{5}$$

Proof: First, a false positive of *BF* means an element in A not appearing in S but not filtered out by *BF*. Observe that after inserting M keys into a table of size K, the probability that a particular bit is still 0 is exactly

$$(1-\frac{1}{K})^{HM} \tag{6}$$

Hence the probability of a false positive in this situation is $(1-(1-\frac{1}{K})^{HM})^H$. However, we know that M denotes the size of A which is the domain of S, so the number of distinct elements in S must be no more than M, i.e. $M' \leq M$, and further we have $(1-(1-\frac{1}{K})^{HM'})^H \leq (1-(1-\frac{1}{K})^{HM})^H$. ■

Theorem 2: Assuming no specific data distribution, the expectation of the number of *FPs* in the output of *BFSS* is no more than

$$M(1-(1-\frac{1}{K})^{HM})^H + \frac{1}{\phi} \tag{7}$$

Proof: We denote the expectation of the number of *FPs* in the output of *BFSS* as $E(\#FPs)$. From Algorithm 1, we can observe that two kinds of elements contribute to *FPs*: i) false positives of *BF*; ii) elements with $f_S(e) > \lfloor \phi N \rfloor$ wrongly output. For the first case, we define the independent 0-1 random variables $x_i (1 \leq i \leq M)$ for each element in A, and the value of x_i depends on whether a_i appeared in S. If a_i appeared in S, then $x_i = 0$; If not, then with a probability of $(1-(1-\frac{1}{K})^{HM'})^H$, $x_i = 1$; From the defination of x_i, we can find that the expectation of the number of the false positives of *BF* equals $E(\sum_{i=1}^{M} x_i)$, i.e. $(M-M')(1-(1-\frac{1}{K})^{HM'})^H$. ■

3) : Subsubsection text here.

IV. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.