Finding Subcube Heavy Hitters in Digital Analytics Streams

Extended Abstract

Branislav Kveton
Adobe Research
kveton@adobe.com

S. Muthukrishnan
Rutgers University
muthu@cs.rutgers.edu

Hoa T. Vu
University of Massachusetts
hvu@cs.umass.edu

Yikun Xian
Rutgers University
siriusxyk@gmail.com

ABSTRACT

Digital analytics consists of online user activity streams (e.g., web browsing activity, commercial site activity, apps and social behavior, and response to ads). These data streams typically have high dimensionality, and an important problem is to find frequent joint values of subsets of dimensions.

We address this subcube heavy-hitters problem. Formally, the data stream consists of d-dimensional items. A *k-dimensional subcube* T is a subset of k distinct coordinates and a *subcube heavy hitter query* $\text{Query}(T, v)$ outputs

- YES if $f_T(v) \geq \gamma$
- NO if $f_T(v) < \gamma/4$

where f_T is the ratio of the number of stream items whose coordinates T have joint values v. The *all subcube heavy hitters query* $\text{AllQuery}(T)$ outputs all joint values v that return YES to $\text{Query}(T, v)$. The problem is to answer these queries correctly for all T and v.

Recently, Liberty et al. showed that any constant-pass streaming algorithm answering even a special case of this query needs $\Omega(d^2/\gamma)$ bits of memory for large k, ignoring the poly-logarithmic factors. Our main contribution is to circumvent this quadratic bottleneck via a model-based approach. In particular, we assume that the dimensions are related to each other via the Naive Bayes model. We present a new two-pass, $\tilde{O}(d/\gamma)$-space algorithm for our problem, and a fast algorithm for answering $\text{AllQuery}(T)$ in $O(k/\gamma^2)$ time.

We also perform empirical study with synthetic datasets as well as real datasets from Adobe and Yandex. We show that our algorithm achieves the least error in finding the subcube heavy hitters compared to a one-pass variant or natural sampling based approach. Our work shows the potential of model-based approach to data stream analysis.

KEYWORDS

algorithms, data streams, heavy hitters, graphical models

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Branislav Kveton, S. Muthukrishnan, Hoa T. Vu, and Yikun Xian. 2017. Finding Subcube Heavy Hitters in Digital Analytics Streams: Extended Abstract. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Most companies see transactions with items sold, time, store location, price, etc. that arrive over time. Modern online companies see streams of user web activities that typically have components of user information including ID (e.g. cookies), hardware (e.g., device), software (e.g., browser, OS), and contents such as web properties, apps. Activity streams also include events (e.g., impressions, views, clicks, purchases) and event attributes (e.g., product id, price, geolocation, time). Even classical IP traffic streams have many dimensions including source and destination IP addresses, port numbers and other features of an IP connection such as application type. Furthermore, in applications such as Natural Language Processing, streams of documents can be thought of as streams of a large number of bigrams or multi-grams over word combinations [6]. As these examples show, analytics data streams with 100's and 1000's of dimensions arise in many applications. Motivated by this, we study the problem of finding *heavy hitters* on data streams focusing on d, the number of dimensions, as a parameter. Given d one sees in practice, d^2 in space usage is prohibitive, for solving the heavy hitters problem on such streams.

Formally, let us start with a one-dimensional stream $x_1, \ldots x_m$ of items where each $x_i \in [n] := \{1, 2, \ldots, n\}$. We can look at the *count* $c(y) = |\{i : x_i = y\}|$ or *frequency ratio* $f(y) = \frac{c(y)}{m}$. A *heavy hitter* item y is one with $c(y) \geq \gamma m$ or equivalently $f(y) \geq \gamma$, for some constant γ. The standard *data stream model* is that we maintain data structures of size polylog(m, n) and determine if y is a heavy hitter with probability of success at least 3/4, that is, if $f(y) \geq \gamma$ output YES and output NO if $f(y) < \gamma/4$ for all y.[1] We note that if $\gamma/4 \leq f(y) < \gamma$, then either answer is acceptable.

Detecting heavy hitters on data streams is a fundamental problem ihat arises in guises such as finding elephant flows and network attacks in networking, finding hot trends in databases, finding frequent patterns in data mining, finding largest coefficients in signal analysis, and so on. Therefore, the heavy hitter problem has been studied extensively in theory, databases, networking and signal

[1] The error can be narrowed to any ϵ and success probability can be amplified to $1 - \delta$ for any parameters α, ϵ with cost $1/\epsilon$, $\log(1/\delta)$ as needed, and we omit these factors in the discussions.

processing literature. See [2] for an early survey and [15] for a recent survey.

Subcube Heavy Hitter Problems. Our focus is on modern data streams such as in analytics cases, with d dimensions, for large d. The data stream consists of d-dimensional items $\mathbf{x}_1, \cdots, \mathbf{x}_m$. In particular, $\mathbf{x}_i = (\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,d})$ and each $\mathbf{x}_{i,j} \in [n]$. The number of items whose jth coordinate has value y is denoted by $c_j(y)$, i.e., $c_j(y) = \left|\{i : \mathbf{x}_{i,j} = y\}\right|$. We define the random variable X_j as jth coordinate's value of a random data stream item. Therefore,

$$f_j(y) = \Pr\left[X_j = y\right] = \frac{c_j(y)}{m} .$$

A *k-dimensional subcube* T is a subset of k distinct coordinates $\{T_1, \cdots, T_k\} \subseteq [d]$ where $T_1 < T_2 < \cdots < T_k$. We say $\dim(T) = k$ and use X_T to denote the random variable of the joint values of the coordinates T of a random item.

Similarly, we refer to the joint values of the coordinates T of $\mathbf{x}_i$ as $\mathbf{x}_{i,T}$. Suppose $v = (v_1, \cdots, v_k) \in [n]^k$, i.e., v is indexed as $[n] \times [n] \times \ldots \times [n]$, we write $X_T = v$ to denote $X_{T_1} = v_1, \cdots, X_{T_k} = v_k$. In the same fashion, we define $c_T(v) := \left|\{i : \mathbf{x}_{i,T} = v\}\right|$ and $f_T(v) := c_T(v)/m = \Pr\left[X_T = v\right]$.

We are now ready to define our problems. They take k, γ as parameters and the stream as the input and build data structures to answer:

- *Subcube Heavy Hitter*: Query(T, v), where $\dim(T) = k$ and $v \in [n]^k$, returns an estimate if $f_T(v) \geq \gamma$. Specifically, output YES if $f_T(v) \geq \gamma$ and NO if $f_T(v) < \gamma/4$. If $\gamma/4 \leq f_T(v) < \gamma$, then either output is acceptable. The required success probability *for all* $T \in \binom{d}{k}$ and $v \in [n]^k$ is at least 3/4.
- *All Subcube Heavy Hitters*: AllQuery(T), where $\dim(T) = k$, outputs all joint values v that return YES to Query(T, v). This is conditioned on the algorithm used for Query(T, v).

It is important to emphasize that the stream is presented (in a single pass or constant passes) to the algorithm before the algorithm receives any query.

Subcube heavy hitters are relevant wherever one dimensional heavy hitters have found applications: combination of source and destination IP addresses forms the subcube heavy hitters that detect network attacks; combination of stores, sales quarters and nature of products forms the subcube heavy hitters that might be the pattern of interest in the data, etc. Given the omnipresence of multiple dimensions in digital analytics, arguably, subcube heavy hitters limn the significant data properties far more than the single dimensional view.

Related Work. The problem we address is directly related to frequent itemset mining studied in the data mining community. In frequent itemset mining, each dimension is binary ($n = 2$), and we consider Query(T, v) where $v = (1, \ldots, 1) := \mathbf{U}_k$. It is known that counting all maximal subcubes T that have a frequent itemset, i.e., $f_T(\mathbf{U}_k) \geq \gamma$, is #$P$-complete [17]. Furthermore, finding even a single T of maximal size such that $f_T(\mathbf{U}_k) \geq \gamma$ is NP-hard [7, 10]. Recently, Liberty et al. showed that any constant-pass streaming algorithm answering Query$(T, \mathbf{U}_k)$ needs to use $\Omega(kd/\gamma \cdot \log(d/k))$ bits of memory [10]. In the worst case, this is $\Omega(d^2/\gamma)$ for large k, ignoring the poly-logarithmic factors. For this specific problem, sampling algorithms will nearly meet their lower bound for space. Our problem is more general, with arbitrary n and v.

Our Contributions. Clearly, the case $k = 1$ can be solved by building one of the many known single dimensional data structures for the heavy hitters problem on each of the d dimension; the $k = d$ case can be thought of as a giant single dimensional problem by linearizing the space of all values in $[n]^k$; for any other k, there are $\binom{d}{k}$ distinct choices for subcube T, and these could be treated as separate one-dimensional problems by linearizing each of the subcubes. In general, this entails $\binom{d}{k}$ and $\log(n^d)$ cost in space or time bounds over the one-dimensional case, which we seek to avoid. Also, our problem can be reduced to the binary case by unary encoding each dimension by n bits, and solving frequent itemset mining: the query then has kn dimensions. The resulting bound will have an additional n factor which is large.

To start with, we observe that the reservoir sampling approach [14] solves subcube heavy hitters problems for arbitrary dimensions and query vectors v. Our analysis shows that the space we use is within poly-logarithmic factors of the lower bound shown in [10] for binary dimensions and query vector $\mathbf{U}_k$, which is a special case of our problem. Therefore, the subcube heavy hitters problem can be solved using $\tilde{O}(kd/\gamma)$ space. However, this is $\Omega(d^2)$ in worst case.

Our main contribution is to avoid this quadratic bottleneck for finding subcube heavy hitters. We adopt the notion that there is an underlying probabilistic model behind the data, and in the spirit of the Naive Bayes model, we assume that the dimensions are nearly (not exactly) mutually independent given an observable latent dimension. This could be considered as a low rank factorization of the dimensions. In particular, one could formalize this assumption by bounding the total variational distance between the data's joint distribution and that derived from the Naive Bayes estimation. This assumption is common in statistical data analysis and highly prevalent in machine learning. Following this modeling, we make two main contributions:

- We present a two-pass, $\tilde{O}(d/\gamma)$-space streaming algorithm for answering Query(T, v). This improves upon the kd factor in the space complexity from sampling, without assumptions, to just d with the Naive Bayes assumption, which would make this algorithm practical for large k. Our algorithm uses sketching in each dimension in one pass to detect heavy hitters, and then needs a second pass to precisely estimate the frequencies of the heavy hitters to compute the overall joint probability accurately.
- We present a fast algorithm for answering AllQuery(T) in $O(k/\gamma^2)$ time. The naive procedure would take exponential time $\Omega((1/\gamma)^k)$ by considering the Cartesian product of the heavy hitters in each dimension. Our approach, on the other hand, uses the structure of the Naive Bayes assumption to iteratively construct the subcube heavy hitters one dimension at a time.

Our work develops the direction of model-based data stream analysis. Model-based data analysis has been effective in other areas. For example, in compressed sensing, realistic signal models

that include dependencies between values and locations of the signal coefficients improve upon unconstrained cases [5]. In statistics, using tree constrained models of multidimensional data sometimes improves point and density estimation. In high dimensional distribution testing, model based approach has also been studied to overcome the curse of dimensionality [4].

In the data stream model, McGregor and Vu [12] studied the problem of evaluating Bayesian Networks. In another work, Kveton et al. [8] assumed a tree graphical model and designed a one-pass algorithm that estimates the joint frequency; their work however only solved the $k = d$ case for the joint frequency estimation problem. Our model is a bit different and more importantly, we solve the subcube heavy hitters problem (addressing all the $\binom{d}{k}$ subcubes) which prior work does not solve. In following such a direction, we have extended the fundamental heavy hitters problem to higher dimensional data. Given that many implementations already exist for the sketches we use for one-dimensional heavy hitters as a blackbox, our algorithms are therefore easily implementable.

Background on the Naive Bayes model and its use in our context. The Naive Bayes Model [13] is a Bayesian network over d features $X_1, \ldots, X_d$ and the class variable Y. This model represents a joint probability distribution of the form

$$\begin{aligned}&\Pr[X_1 = x_1, \ldots, X_d = x_d, Y = y] \\ &= \Pr[Y = y] \prod_{j=1}^{d} \Pr\left[X_j = x_j \mid Y = y\right] ,\end{aligned}$$

which means that the values of the features are conditionally independent given the value of the class variable. The simplicity of the Naive Bayes model makes it a popular choice in text processing and information retrieval [9, 11], with state-of-the-art performance in spam filtering [1], text classification [9], and others.

Empirical Study. We perform detailed experimental study of subcube heavy hitters. We use synthetic datasets where we generate data that confirms to the Naive Bayes model, and also use real data set from Yandex (Search) and Adobe (Marketing Cloud) which give multidimensional analytics streams. We experiment with not only the reservoir sampling based algorithm as a benchmark that works without modeling assumptions, and our two-pass subcube heavy hitters algorithm that improves upon it for data that satisfies the model, we also adopt our approach to give a simpler one-pass algorithm for which theoretical guarantees is weaker. Our experiments show substantial improvement of the model-based algorithms over the benchmark for synthetic as well as real data sets, and further show the benefits of the second pass.

2 THE SAMPLING APPROACH

In this section, we show that sampling solves the problem efficiently compared to running one-dimensional heavy hitters for all $\binom{n}{k}$ subcubes independently. Furthermore, this upper bound matches the lower bound in [10] up to poly-logarithmic factors.

Algorithm details. The algorithm samples $m' = \tilde{O}(\gamma^{-1}kd)$ random items $\mathbf{z}_1, \ldots, \mathbf{z}_{m'}$ from the stream using Reservoir sampling [14].

(1) Sample $m' = \tilde{O}(\gamma^{-1}kd)$ random items S.
(2) Output YES to Query(T, v) if and only if $\hat{f}_T(v) \geq \gamma/2$.

Figure 1: Sampling algorithm

Let $S = \{\mathbf{z}_1, \ldots, \mathbf{z}_{m'}\}$ be the sample set. Given Query(T, v), we output YES if and only if the sample frequency of v, denoted by $\hat{f}_T(v)$, is at least $\gamma/2$. Specifically,

$$\hat{f}_T(v) := \frac{|\{\mathbf{x}_i : \mathbf{x}_i \in S \text{ and } \mathbf{x}_{i,T} = v\}|}{m'} .$$

The detailed algorithm is in Figure 1. For all subcubes T and joint values v of T, the expected sample frequency $\hat{f}_T(v)$ is $f_T(v)$. Intuitively, if v is a frequent joint values, then according to Chernoff bound, its sample frequency $\hat{f}_T(v) \approx f_T(v)$ with high probability; otherwise, $\hat{f}_T(v)$ stays small.

Let us fix a k-dimensional subcube T and suppose that for all $v \in [n]^k$, we have

$$\hat{f}_T(v) = f_T(v) \pm \frac{\max\{\gamma, f_T(v)\}}{4} . \tag{1}$$

It is then straightforward to see that if $f_T(v) \leq \gamma/4$, then $\hat{f}_T(v) < \gamma/2$ and if $f_T(v) \geq \gamma$, then $\hat{f}_T(v) \geq 3\gamma/4 > \gamma/2$. Hence, one could output YES for all v where $\hat{f}_T(v) \geq \gamma/2$, and output NO otherwise.

Lemma 2.1. *(Chernoff bound) Let $X_1, \cdots, X_n$ be independent or negatively correlated binary random variables. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathrm{E}[X]$. Then,*

$$\Pr[|X - \mu| \geq \epsilon\mu] \leq \exp(-\max\{\epsilon^2, \epsilon\}\mu/3) .$$

Recall that $S = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_{m'}\}$ is the sample set returned by the algorithm. For a fixed $v \in [n]^k$, we use Z_i as the indicator variable for the event $\mathbf{z}_{i,T} = v$. Since we sample without replacement, the random variables Z_i are negatively correlated.

Lemma 2.2. *For all k-dimensional subcubes T and joint values $v \in [n]^k$, with probability at least 0.9,*

$$\hat{f}_T(v) = f_T(v) \pm \frac{\max\{\gamma, f_T(v)\}}{4} .$$

Proof. Let $m' = c\gamma^{-1}\log(d^k \cdot n^k)$ for some sufficiently large constant c. We first consider a fixed $v \in [n]^k$ and define the random variables Z_i as above. Specifically, $Z_i = 1$ if $\mathbf{z}_{i,T} = v$. Suppose $f_T(v) \geq \gamma$. Appealing to Lemma 2.1, we have

$$\begin{aligned}&\Pr\left[\left|\left(\sum_{i=1}^{m'} \frac{Z_i}{m'}\right) - f_T(v)\right| \geq \frac{f_T(v)}{4}\right] \\ &= \Pr\left[\left|\hat{f}_T(v) - f_T(v)\right| \geq \frac{f_T(v)}{4}\right] \\ &\leq \exp\left(-\frac{f_T(v)m'}{12}\right) \leq \frac{1}{10d^k n^k} .\end{aligned}$$

On the other hand, if $f_T(v) < \gamma/4$, then

$$\Pr\left[\left|\hat{f}_T(v) - f_T(v)\right| \geq \frac{\gamma}{4}\right] \leq \exp\left(-\left(\frac{\gamma}{4f_T(v)}\right) f_T(v)\frac{m'}{3}\right) \leq \frac{1}{10d^k n^k} .$$

Therefore, by taking the union bound over all $\binom{d}{k} \cdot n^k \leq d^k \cdot n^k$ possible combinations of k-dimensional subcubes and the corresponding joint values $v \in [n]^k$, we deduce the claim. □

The above lemma showed that Eq. 1 holds and we therefore could answer all Query(T, v) correctly with probability at least 0.9 for all $v \in [n]^k$ and k-dimensional subcubes T. Because storing each sample $\mathbf{z}_i$ requires $\tilde{O}(d)$ bits of space, the algorithm uses $\tilde{O}(dk\gamma^{-1})$ bits of space. We summarize the result as follows.

Theorem 2.3. *There exists a one-pass algorithm that uses $\tilde{O}(dk\gamma^{-1})$ space and solves k-dimensional subcube heavy hitters. Furthermore,* Query(T, v) *takes $\tilde{O}(dk\gamma^{-1})$ time and* AllQuery(T) *takes $\tilde{O}\left(\left(dk\gamma^{-1}\right)^2\right)$ time.*

3 WARM-UP: THE NEAR-INDEPENDENCE CASE

The near-independence assumption. Suppose the random variables representing the dimensions are *near* independent. We show that there is a 2-pass algorithm that uses less space and has faster query time. At a high level, we make the assumption that the joint probability is approximately factorized for all subcubes $T \subseteq [d]$:

$$f_T(v) \approx f_{T_1}(v_1) f_{T_2}(v_2) \cdots f_{T_k}(v_k) .$$

More formally, we assume that the total variation distance is bounded by a small quantity α as follows. If $\dim(T) = h$, then

$$\max_{v \in [n]^h} \left| f_T(v) - \prod_{i=1}^{h} f_{T_i}(v_i) \right| < \alpha .$$

We emphasize that this distance is bounded for all subcubes T. We refer to the above as the *near-independence* assumption. Furthermore, we assume that α is reasonable with respect to γ that controls the heavy hitters. For example, $\alpha < \gamma/10$ will suffice.

We observe that if $f_T(v) \geq \gamma$, then $\prod_{i=1}^{k} f_{T_i}(v_i) \geq \gamma - \gamma/10 \geq \gamma/2$ and if $f_T(v) < \gamma/4$, then $\prod_{i=1}^{k} f_{T_i}(v_i) < \gamma/4 + \gamma/10 \leq \gamma/2$. We therefore want to output YES to Query(T, v) if and only if $\prod_{i=1}^{k} f_{T_i}(v_i) \geq \gamma/2$. For convenience, we define $\lambda := \gamma/2$.

Algorithm details. We make following simple but useful observation. The observation allows us to argue that if v is a heavy hitter in the subcube T and if T' is a subcube of T, then $v_{T'}$ is a heavy hitter in the subcube T'. More formally,

Lemma 3.1. *For all h-dimensional subcubes T, where h is arbitrary, we have*

$$\prod_{i=1}^{h} f_{T_i}(v_i) \geq \lambda \implies \prod_{i \in \mathcal{V}} f_{T_i}(v_i) \geq \lambda$$

for all $\mathcal{V} \subseteq [h]$ (in other words, $\{T_i : i \in \mathcal{V}\}$ is a subcube of T).

(1) 1st pass: For each coordinate $i \in [d]$, use the Count-Min sketch to find H_i.
(2) 2nd pass: For each $i \in [d]$, compute $f_i(x)$ exactly for all $x \in H_i$ to obtain S_i.
(3) Output YES to Query(T, v) if and only if all $v_i \in S_{T_i}$ and $\prod_{i=1}^{k} f_{T_i}(v_i) \geq \lambda$.

Figure 2: Algorithm under the near-independence assumption

Therefore, if $\prod_{i=1}^{k} f_{T_i}(v_i) \geq \lambda$, then we must have $f_{T_1}(v_1) \geq \lambda$, $f_{T_2}(v_2) \geq \lambda$, and so on. To this end, by using (for example) the Count-Min sketch [3], with high probability, we could find a set H_i such that if $f_i(x) \geq \lambda/2$, then $x \in H_i$ and if $f_i(x) < \lambda/4$, then $x \notin H_i$. With another pass over the stream, for each $x \in H_i$, we compute $f_i(x)$ exactly to obtain

$$S_i := \{x \in [n] : f_i(x) \geq \lambda\}$$

for all $i \in [n]$.

Appealing to Lemma 3.1, if $\prod_{i=1}^{k} f_{T_i}(v_i) \geq \lambda$, then $v_i \in S_i$ for all i. Hence, we output YES to Query(T, v) if and only if all $v_i \in S_{T_i}$ and $\prod_{i=1}^{k} f_{T_i}(v_i) \geq \lambda$. The detailed algorithm is in Figure 2.

Theorem 3.2. *There exists a 2-pass algorithm that uses $\tilde{O}(d\gamma^{-1})$ space and solves subcube heavy hitters under the near-independence assumption. The time to answer* Query(T, v) *and* AllQuery(T) *are $\tilde{O}(k)$ and $O(k\gamma^{-1})$ respectively where $k = \dim(T)$.*

Proof. The first pass uses $\tilde{O}(d\lambda^{-1})$ space since the Count-Min data structure uses $\tilde{O}(\lambda^{-1})$ space for each coordinate $i \in [d]$. We observe that because size of each H_i is $O(\lambda^{-1})$, second pass also uses $\tilde{O}(d\lambda^{-1})$ space. [2]

We have already established correctness of the algorithm for an arbitrary Query(T, v) under the near-independence assumption above. Next, we exhibit a fast algorithm to answer AllQuery(T). We note that naively checking all combinations $(v_1, \cdots, v_k) \in S_{T_1} \times S_{T_2} \times \cdots \times S_{T_k}$ takes exponential $\Omega(\lambda^{-k})$ time in the worst case.

Our approach figures out the frequent joint values gradually and takes advantage of the near-independence assumption. In particular, define

$$W_j := \{v \in [n]^j : f_{T_1}(v_1) \cdots f_{T_j}(v_j) \geq \lambda\} .$$

The goal becomes finding W_k. Note that $W_1 = S_1$ is obtained directly by the algorithm. We shall show that it is possible to construct W_{j+1} from W_j in $\tilde{O}(\lambda^{-1})$ time which in turn means that we are able to find W_k in $\tilde{O}(k\lambda^{-1})$ time. We define $T_{1:j} := \{T_1, \cdots, T_j\}$ and $v_{1:j} = (v_1, v_2, \ldots, v_j)$.

We note that $|W_j| \leq 5/4 \cdot \lambda^{-1}$. This is because if $\prod_{i=1}^{k} f_{T_j}(v_j) \geq \lambda$ then $f_{T_{1:j}}(v_{1:j}) \geq \lambda - \alpha \geq 4/5 \cdot \lambda$ according to the near-independence assumption. For each $(v_1, \ldots, v_j) \in W_j$, we collect all $v_{j+1} \in S_{j+1}$

[2] We note that guaranteeing correct H_i for all $i \in [d]$ with high probability only results in a $\log d$ factor in the space complexity.

such that

$$f_{T_{j+1}}(v_{j+1}) \geq \frac{\lambda}{\prod_{i=1}^{j} f_{T_i}(v_i)}$$

and put $(v_1, \cdots, v_{j+1})$ into W_{j+1}. Since $|W_j| \leq 5/4 \cdot \lambda^{-1}$ and $|S_{j+1}| \leq \lambda^{-1}$, this step obviously takes $O(\lambda^{-2})$ time. However, a tighter analysis yields $\tilde{O}(\lambda^{-1})$ time by observing that there could be at most $\lambda^{-1} \prod_{i=1}^{j} f_{T_i}(v_i)$ such v_{j+1} for each $(v_1, \cdots, v_j) \in W_j$. The time complexity of this step is therefore

$$\tilde{O}\left(\sum_{v' \in W_j} \lambda^{-1} \prod_{i=1}^{j} f_{T_i}(v'_i)\right) = \tilde{O}\left(\sum_{v' \in W_j} \lambda^{-1} (f_{T_{1:j}}(v') + \alpha)\right)$$
$$= \tilde{O}\left(\sum_{v' \in W_j} \lambda^{-1}\alpha + \sum_{v' \in W_j} \lambda^{-1} f_{T_{1:j}}(v')\right)$$
$$= \tilde{O}\left(\left|W_j\right| + \lambda^{-1}\right) = \tilde{O}\left(\lambda^{-1}\right) .$$

As the algorithm finds W_{j+1} given W_j in $\tilde{O}(\lambda^{-1})$ time[3], it therefore obtains W_k in $\tilde{O}(k\lambda^{-1}) = \tilde{O}(k\gamma^{-1})$ time. The correctness of this procedure follows directly from Lemma 3.1 since $(v_1, \ldots, v_{j+1}) \in W_{j+1}$ implies that $(v_1, \ldots, v_j) \in W_j$ and $v_{j+1} \in S_{j+1}$. Thus, by checking all combinations of $(v_1, \ldots, v_j) \in W_j$ and $v_{j+1} \in S_{j+1}$, we ensure that we construct W_{j+1} completely. □

4 THE NAIVE BAYES MODEL: THE GENERAL CASE

The Naive Bayes assumption. In this final section, we focus on the data stream inspired by the Naive Bayes model which is strictly more general than the near-independence assumption. In particular, we assume that the coordinates are near-independent given an extra $(d+1)$th observable coordinate that has a value in $\{1, \ldots, \ell\}$. As in typical in Naive Bayes analysis, we assume ℓ is a constant but perform the calculations in terms of ℓ so its role in the complexity of the problem is apparent. Informally, this model asserts that for all subcubes T where $\dim(T) = h$, joint values $v \in [n]^h$, and $z \in [\ell]$,

$$f_T(v) \approx \sum_{z \in [\ell]} f_{T_1 \mid d+1}(v_1 \mid z) \cdots f_{T_h \mid d+1}(v_d \mid z) f_{d+1}(z) .$$

where $f_{T_i \mid d+1}(v_i \mid z)$ is defined as the conditional probability $\Pr[X_i = v_i \mid X_{d+1} = z]$. The $(d+1)$th dimension is often referred to as the *latent dimension*. Similar to the previous section, we formalize our model-based assumption as follows. For all h-dimensional subcubes $T \subseteq [d]$ where $h \leq d$ is arbitrary,

$$\max_{v \in [n]^h} \left| f_T(v) - \sum_{z \in [\ell]} f_{d+1}(z) \prod_{i=1}^{h} f_{T_i \mid d+1}(v_i \mid z) \right| < \alpha .$$

Algorithm details. We again assume $\alpha < \gamma/10$. As argued in the previous section, it suffices to output YES to Query(T, v) if and only if

$$\sum_{z \in [\ell]} f_{d+1}(z) \prod_{i=1}^{h} f_{T_i \mid d+1}(v_i \mid z) \geq \gamma/2 = \lambda.$$

First, we generalize Lemma 3.1 as follows.

[3]This requires sorting W_{j+1} based on the frequency.

(1) 1st pass:
  (a) For each value $z \in [\ell]$, compute $f_{d+1}(z)$ exactly.
  (b) For each coordinate $i \in [d]$, use Count-Min sketch to find H_i.
(2) 2nd pass:
  (a) For each $i \in [d]$ and for all $x \in H_i$, compute $f_i(x)$ exactly to obtain S_i.
  (b) For each value $z \in [\ell]$, dimension $i \in [d]$, and $x \in H_i$, compute $f_{i \mid d+1}(x \mid z)$ exactly.
(3) Output YES to Query(T, v) if and only if all $v_i \in S_{T_i}$ and

$$f_T(v) := \sum_{z \in [\ell]} f_{d+1}(z) \prod_{i=1}^{k} f_{T_i \mid d+1}(v_i \mid z) \geq \lambda.$$

Figure 3: Algorithm under the Naive Bayes assumption

Lemma 4.1. *For all h-dimensional subcubes T, where h is arbitrary, we have*

$$\sum_{z \in [\ell]} f_{d+1}(z) \prod_{i=1}^{h} f_{T_i \mid d+1}(v_i \mid z) \geq \lambda$$
$$\implies \sum_{z \in [\ell]} f_{d+1}(z) \prod_{i \in \mathcal{V}} f_{T_i \mid d+1}(v_i \mid z) \geq \lambda$$

for all $\mathcal{V} \subseteq [h]$ (in other words, $\{T_i : i \in \mathcal{V}\}$ is a subcube of T).

Proof. For a fixed z, observe that $\sum_{y_j \in [n]} f_{T_j \mid d+1}(y_j \mid z) = 1$. We simply rewrite

$$\sum_{z \in [\ell]} f_{d+1}(z) \prod_{i \in \mathcal{V}} f_{T_i \mid d+1}(v_i \mid z)$$
$$= \sum_{z \in [\ell]} f_{d+1}(z) \prod_{i \in \mathcal{V}} f_{T_i \mid d+1}(v_i \mid z) \cdot \prod_{j \notin \mathcal{V}} \left(\sum_{y_j \in [n]} f_{T_j \mid d+1}(y_j \mid z) \right)$$
$$\geq \sum_{z \in [\ell]} f_{d+1}(z) \prod_{i \in \mathcal{V}} f_{T_i \mid d+1}(v_i \mid z) \cdot \prod_{j \notin \mathcal{V}} f_{T_j \mid d+1}(v_j \mid z)$$
$$= \sum_{z \in [\ell]} f_{d+1}(z) \prod_{i=1}^{h} f_{T_i \mid d+1}(v_i \mid z) \geq \lambda .$$

□

Similar to the previous section, for each dimension $i \in [n]$, we find S_i over two passes as described. By appealing to Lemma 4.1, we deduce that if

$$\sum_{z \in [\ell]} f_{d+1}(z) \prod_{i=1}^{k} f_{T_i \mid d+1}(v_i \mid z) \geq \lambda$$

then for all $i = 1, 2, \ldots, k$, we have

$$\sum_{z \in [\ell]} f_{T_i \mid d+1}(v_i \mid z) f_{d+1}(z) = f_{T_i}(v_i) \geq \lambda$$

which in turn implies that $v_i \in S_{T_i}$. Therefore, it is apparent that we output YES to Query(T, v) if and only if all $v_i \in S_{T_i}$ and

$$\sum_{z \in [\ell]} f_{d+1}(z) \prod_{i=1}^{k} f_{T_i \mid d+1}(v_i \mid z) \geq \lambda \ .$$

To check the above, we compute $f_{i \mid d+1}(x \mid z)$ and $f_{d+1}(z)$ for all $x \in S_i$ and for all $z \in [\ell]$. The detailed algorithm is in Figure 3.

Theorem 4.2. *There exists a 2-pass algorithm that uses $\tilde{O}(\ell d \gamma^{-1})$ space and solves subcube heavy hitters under the Naive Bayes assumption. The time to answer* Query(T, v) *and* AllQuery(T) *are $O(\ell k)$ and $O(\ell k \gamma^{-2})$ respectively.*

Proof. The total space to obtain H_i and S_i over the two passes is $\tilde{O}(d\lambda^{-1})$. Additionally, computing $f_{i \mid d+1}(x \mid z)$ for all $i \in [d]$, $z \in [\ell]$, and $x \in H_i$ requires $\tilde{O}(\ell d \lambda^{-1})$ bits of space. The overall space is $\tilde{O}(\ell d \lambda^{-1})$ as a result.

We already established the correctness of answering Query(T, v) for all subcubes T and the corresponding joint values v above. We now exhibit a fast algorithm to answer AllQuery(T). Define

$$W_j := \{v \in [n]^j : \sum_{z \in [\ell]} \prod_{i=1}^{j} f_{T_i \mid d+1}(v_i \mid z) f_{d+1}(z) \geq \lambda\} \ .$$

The goal is to find W_k and note that $W_1 = S_1$ is obtained directly by the algorithm. Next, we show how to derive W_{j+1} in $O(\lambda^{-2})$ time from W_j. Observe that $|W_j| \leq 5/4 \cdot \lambda^{-1}$ because if $v \in W_j$, then

$$\sum_{z \in [\ell]} f_{T_1 \mid d+1}(v_1 \mid z) \cdots f_{T_j \mid d+1}(v_j \mid z) f_{d+1}(z) \geq \lambda$$
$$\implies f_{T_{1:j}}(v_{1:j}) \geq \lambda - \alpha = 4/5 \cdot \lambda^{-1}$$

according to the Naive Bayes assumption. For each $(v_1, \cdots, v_j)$ in W_j, we collect all $v_{j+1} \in S_{j+1}$ such that

$$\sum_{z \in [\ell]} f_{T_1 \mid d+1}(v_1 \mid z) \cdots f_{T_j \mid d+1}(v_{j+1} \mid z) f_{d+1}(z) \geq \lambda$$

and put $(v_1, \cdots, v_{j+1})$ to W_{j+1}. Since $|W_j| \leq 5/4 \cdot \lambda^{-1}$ and $|S_{j+1}| \leq \lambda^{-1}$, this step obviously takes $O(\ell \lambda^{-2})$ time. We construct W_{j+1} given W_j in $O(\ell \lambda^{-2})$ time. As a result, we attain W_k in $O(k \ell \lambda^{-2}) = O(k \ell \gamma^{-2})$ time. The correctness of this procedure follows directly from Lemma 4.1 since $(v_1, \ldots, v_{j+1}) \in W_{j+1}$ implies that $(v_1, \ldots, v_j) \in W_j$ and $v_{j+1} \in S_{j+1}$. Since we check all possible combinations of $(v_1, \ldots, v_j) \in W_j$ and $v_{j+1} \in S_{j+1}$, the algorithm guarantees to construct W_{j+1} completely.

□

5 EXPERIMENTAL STUDY

Overview. We experimentally study our algorithms on a synthetic dataset generated from a Naive Bayes model, and two real-world datasets from Adobe Marketing Cloud[4] and Yandex. We thoroughly compare the following approaches in our experiments:

(1) The reservoir sampling method (RSampling) described in Section 2.
(2) Two-pass algorithms (TwoPassAlg) described in Section 3 or 4 depending on the context of the experiment.
(3) Count-Min sketch heuristic (CMHeuristic): this heuristic uses Count-Min sketch's point query estimation (see [3]) to estimate the frequencies given by the near-independence or Naive Bayes formula instead of making a second pass through the stream to compute their exact values. We note that this approach has no theoretical guarantee. However, it is a practical heuristic.

We highlight the main differences between theory and the actual implementation:

- Instead of running our algorithms with the memory bound given by our theoretical analyses, we run and compare them for different memory limits. This is more practical and natural from the implementation perspective.
- In theory, RSampling and TwoPassAlg use a fixed threshold $\gamma^* = \gamma/2$ to decide between outputting YES or NO. We however experiment with different values of γ^* which is helpful when the memory is more restricted or when the modeling assumptions are not a perfect fit in real data.

The heavy hitters threshold γ is carefully chosen. In particular, we want the proportion between the number of heavy hitters and the total number joint values to be reasonably small, i.e., approximately at most 1% in this paper. Therefore, we use different values of γ for each dataset (see Table 2 for the actual parameter values).

5.1 Synthetic Dataset

The synthetic dataset is sampled from a pre-trained Naive Bayes model that is used to estimate the probability of a page view (the dataset was provided by [8]). The coordinates (see Table 1) consist of one class variable Z and five feature variables $(X_1, \ldots, X_5)$ with high cardinalities. The dataset strongly follows the property that $X_1, \ldots, X_5$ are conditional independent given Z.

Variable	Meaning	Cardinality
Z	COUNTRY	7
X_1	CITY	17,167
X_2	PAGE NAME	4,616
X_3	STARTING PAGE NAME	8,960
X_4	CAMPAIGN	3,969
X_5	BROWSER	287

Table 1: Synthetic dataset overview

Warm-up experiment. We first evaluate RSampling and CMHeuristic on this synthetic dataset. As mentioned earlier, we compare the performance of the two approaches for each fixed memory size.[5] We take a subset of $\approx$ 850,000 records of synthetic data conditioned on $Z = 7$ so that $X_1, \ldots, X_5$ are independent in this subset. We then run experiments on three subcubes $T^{(1)} = \{X_1, X_2, X_3\}$, $T^{(2)} = \{X_2, X_3, X_4\}$ and $T^{(3)} = \{X_3, X_4, X_5\}$. In this warm-up experiment, the main goal is not to find the heavy hitters but to compare the accuracy of the heavy hitters frequency estimations given by CMHeuristic and RSampling. We measure the performance via the

[4] http://www.adobe.com/marketing-cloud.html

[5] We compute the memory use by the RSampling as the product of dimension and the sample size. The memory used by CMHeuristic is computed as the product of dimension and the Count-Min sketch's size.

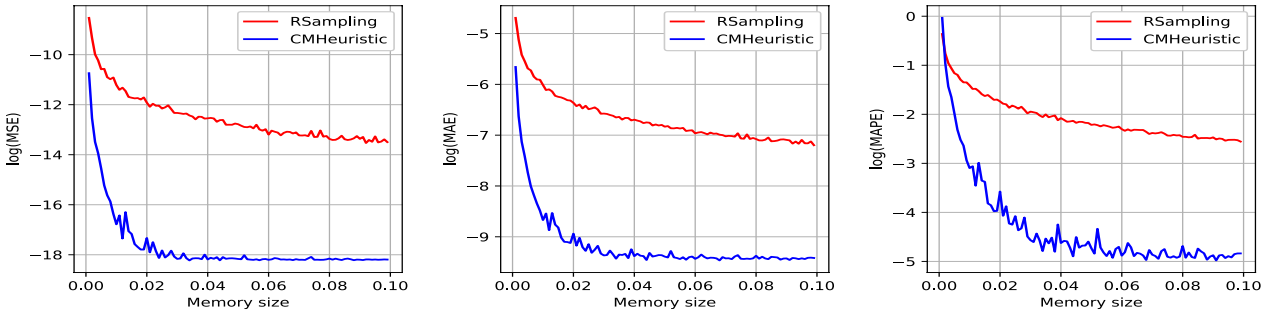

Figure 4: Warm-up experiment on synthetic data. Memory size ranges from 0.1% to 10% of data size

mean square error (MSE), the mean absolute error (MAE), and the mean absolute percentage error (MAPE). To do this, the true frequencies were pre-computed. We use the frequencies of the top 10 heavy hitters in each of the above subcubes. The results in Figure 4 indicate that CMHeuristic outperforms RSampling when restricted to small memory. This experiment shows that knowing the underlying distribution structure helps improving small space heuristic's performance in estimating the heavy hitters frequencies.

Independence experiment. We compare performance of the three aforementioned methods on finding heavy hitters under the near-independence assumption. In this experiment, we use the same subset of data and subcubes as in the previous experiment. We fix the memory as 1% of data size.

We measure the performance, for different values of γ^*, based on the number of true positives and false positives. As shown in Figure 5, for small memory, both CMHeuristic and TwoPassAlg manage to find more heavy hitters than RSampling. However, RSampling makes fewer mistakes in terms of false positives than CMHeuristic when γ^* is small. One possible explanation is that when γ^* is small (close to γ), TwoPassAlg, with the advantage of the second pass, accurately estimates frequencies of potential heavy hitters whereas the other two methods, especially CMHeuristic, overestimate the frequencies and therefore report more false positives. For larger γ^*, false positives become less likely and all three approaches achieve similar performances in terms of false positives. In general, TwoPassAlg obtains the best performance, as theoretically predicted, in the ROC curve.

Naive Bayes experiment. We use the whole dataset of one million records without the previous restriction $Z = 7$ and keep other settings unchanged. We only compared the performance of TwoPassAlg and RSampling because the conditional probability cannot be directly derived from CMHeuristic. When restricted to small memory, TwoPassAlg attains a better performance in finding heavy hitters compared to RSampling. See Figure 6.

5.2 Near-Independence Assumption on Clickstream Dataset

To evaluate TwoPassAlg on real data, we use an advertising dataset called *Clickstream Data Feeds* from Adobe Marketing Cloud. The dataset size is $\approx 67,000$ and all values have been anonymized in advance. There are 19 high cardinality variables grouped by categories as follows.

- Geography info: city, region, country, domain, carrier
- Page info: page url, start page url, first hit page url
- Search info: visit number, referrer, campaign, keywords, search engine,
- External info: browser, browser width/height, plugins, language, OS.

We avoid obvious correlated features in the query subcubes, e.g., "search engine" and "keywords" are highly correlated. See Table 3 for highly correlated variables.

Variable 1	Variable 2	Correlation
keywords	search engine	0.6752
campaign	referrer	0.3579
browser height	OS	0.3225
page url	first hit page url	0.1748
country	carrier	0.0717

Table 3: Highly-correlated variables in *Clickstream*

We note that TwoPassAlg maintains theoretical guarantee as long as the coordinates in the query subcubes satisfy the near-independence or Naive Bayes assumption. We carefully select a subset of coordinates that may follow the near independence assumption to query on. For instance, we show our experiment results for the subcubes, along with the number of heavy hitters recorded:

$$T^{(1)} = \{\text{region}, \text{page url}, \text{language}\}\ (42)$$

$$T^{(2)} = \{\text{region}, \text{campaign}, \text{plugins}\}\ (79)$$

$$T^{(3)} = \{\text{domain}, \text{first hit page url}, \text{plugins}\}\ (25)$$

$$T^{(4)} = \{\text{carrier}, \text{referrer}, \text{OS}\}\ (24)$$

We increase memory size to 30% of the data size in consideration of smaller dataset and higher dimensionality compared to the synthetic dataset. Recall that the memory used by RSampling and TwoPassAlg is partially determined by the number of dimensions and therefore it is reasonable to use a relatively larger memory size.

All three algorithms are able to find most true heavy hitters (see Figure 7), but TwoPassAlg returns far fewer false positives than the other two methods when γ^* is small. Furthermore, TwoPassAlg

Dataset	Memory size	γ	γ^*	#Subcubes	Avg #HH	HH proportion
Synthetic ($Z = 7$)	1%	2×10^{-3}	$[15 \times 10^{-3}, 9 \times 10^{-3}]$	3	28.7	0.017%
Synthetic (whole)	1%	2×10^{-3}	$[15 \times 10^{-3}, 9 \times 10^{-3}]$	3	26.0	0.015%
Clickstream	30%	2×10^{-3}	$[10^{-3}, 10^{-2}]$	4	42.5	0.24%
Yandex (10 subsets)	0.2%	0.1	$[0.05, 0.5]$	8	2.2	1.65%

Table 2: Parameter values for each experiment, HH proportion is the average proportion of the number of heavy hitters and the number of joint values (in the subcube)

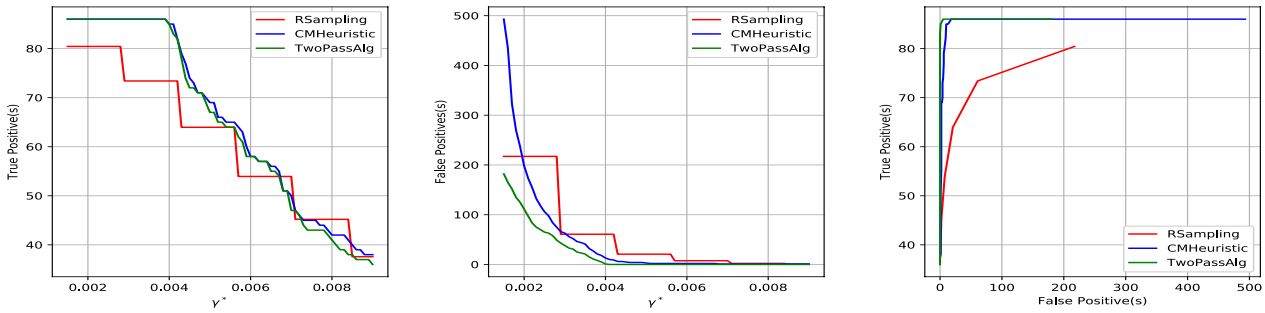

Figure 5: Near-independence experiment on synthetic data

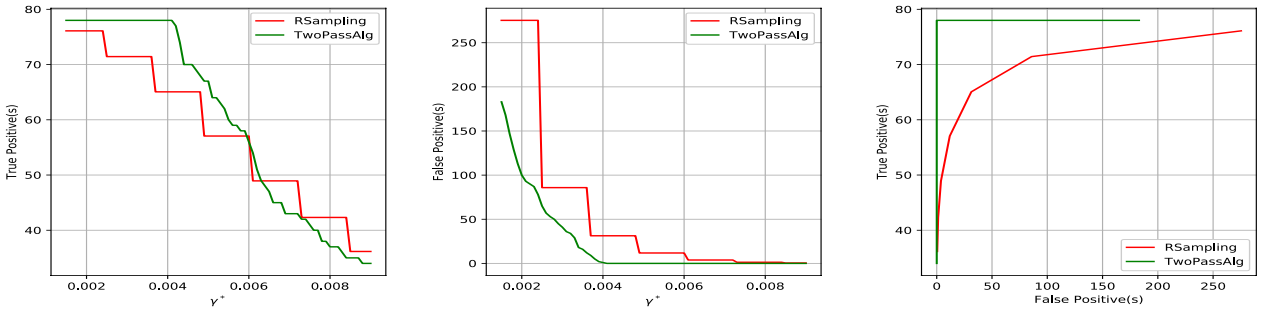

Figure 6: Naive Bayes experiment on synthetic data

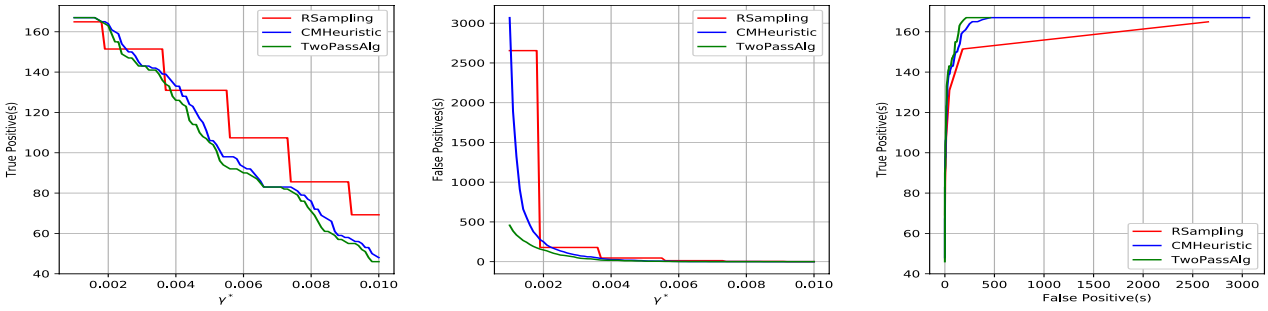

Figure 7: Near-independence experiment with Clickstream dataset

reaches zero false positive for reasonably large γ^*. Thus, in the ROC curve, TwoPassAlg performs slightly better than CMHeuristic. It is unclear if some variables of this dataset constitute a Naive Bayes model. Therefore, we do not experiment with conditional independence on this dataset.

5.3 Naive Bayes Assumption on Yandex dataset

Finally, we consider the *Yandex dataset* [16]. This dataset describes the returned results of search engine for user queries. In the pre-processing step, we split the dataset by user query into 10 subsets

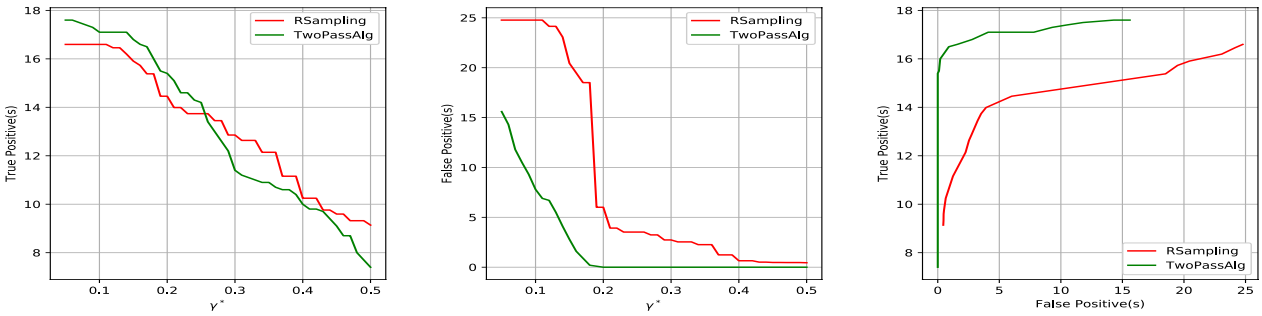

Figure 8: Naive Bayes experiment with Yandex dataset

of sizes ranging from $\approx$ 58,000 to $\approx$ 99,000. In each subset, we treat the first 10 search results as variables $X_1,..,X_{10}$ and "day of query" as the latent variable Z. We conjecture that the search results $X_1,..,X_{10}$ are approximately independent conditioned on a given day Z. We observe that web patterns typically experience heavy weekly seasonality and these search results largely depend on user query time for some fixed query. We proceed to evaluate TwoPassAlg under the Naive Bayes assumption on this dataset.

We consider 8 subcubes in the form $T^{(i)} = \{X_i, X_{i+1}, X_{i+2}\}$. We deliberately set a smaller memory size for this experiment because the cardinality of this dataset is relatively low and RSampling works well with larger memory as the number of joint values is relatively small in this dataset. Different subsets of data may have different number of heavy hitters, so we take the average over 10 subsets as the final result. We report the results in Figure 8. We observe that both RSampling and TwoPassAlg are able to find most true heavy hitters. However, for larger γ^*, TwoPassAlg reaches zero false positives more quickly. This final experiment demonstrates the effectiveness of our main contribution TwoPassAlg on a real dataset.

6 CONCLUDING REMARKS

Data streams in analytics applications have very large dimensions. So, we study the fundamental problem of heavy hitters of size γ with d, the number of dimensions of the items in the stream, as a parameter. We defined subspace heavy hitter problems with subspace of size k, $k \leq d$. A special case of this is frequent itemset mining for which a lower bound of $\Omega(kd/\gamma)$ is known [10] on space needed, ignoring polylog factors. We showed a simple reservoir sampling based algorithm that meets this bound, upto polylog factors. But the focus of our work is to avoid this quadratic dependence on the number of dimensions in the worst case.

We presented a model-based direction, adopting the classical Naive Bayes approach to assume that the dimensions are nearly independent, with or without conditioning on a latent variable. We present two pass streaming algorithms using $O(d/\gamma)$ space, and further present an efficient algorithm to list all the subcube heavy hitters.

Our work demonstrates the power of model-based approach for analyzing high dimensional data that abounds in digital analytics applications. Our approach to subspace heavy hitters opens several directions for further study. For example,

- Can heavy hitters be detected efficiently under more general models among dimensions, such as graphical models with hidden variables?
- Can these models be learned or fitted over data streams with polylogarithmic space? We believe this is an algorithmic problem of great interest and will have applications in machine learning beyond the context here.
- We assumed that we know the latent dimension. Can this be learned from the data stream?
- Can the model-based approach be extended to other problems besides heavy hitters, including clustering, anomaly detection, geometric problems and others which have been studied in the streaming literature.

REFERENCES

[1] Ion Androutsopoulos, Georgios Paliouras, Vangelis Karkaletsis, Georgios Sakkis, Constantine D. Spyropoulos, and Panagiotis Stamatopoulos. 2000. Learning to Filter Spam E-Mail: A Comparison of a Naive Bayesian and a Memory-Based Approach. *CoRR* cs.CL/0009009 (2000).

[2] Graham Cormode. 2008. Finding Frequent Items in Data Streams. http://dmac.rutgers.edu/Workshops/WGUnifyingTheory/Slides/cormode.pdf. (2008). DIMACS Workshop.

[3] Graham Cormode and S. Muthukrishnan. 2004. An Improved Data Stream Summary: The Count-Min Sketch and Its Applications. In *LATIN (Lecture Notes in Computer Science)*, Vol. 2976. Springer, 29–38.

[4] Constantinos Daskalakis, Nishanth Dikkala, and Gautam Kamath. 2016. Testing Ising Models. *CoRR* abs/1612.03147 (2016). arXiv:1612.03147 http://arxiv.org/abs/1612.03147

[5] Marco F. Duarte, Volkan Cevher, and Richard G. Baraniuk. 2009. Model-based compressive sensing for signal ensembles. In *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on*. IEEE, 244–250.

[6] Amit Goyal, Hal Daumé III, and Suresh Venkatasubramanian. 2009. Streaming for large scale NLP: Language Modeling. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, May 31 - June 5, 2009, Boulder, Colorado, USA*. The Association for Computational Linguistics, 512–520. http://www.aclweb.org/anthology/N09-1058

[7] Matthew Hamilton, Rhonda Chaytor, and Todd Wareham. 2006. The Parameterized Complexity of Enumerating Frequent Itemsets. In *IWPEC (Lecture Notes in Computer Science)*, Vol. 4169. Springer, 227–238.

[8] Branislav Kveton, Hung Hai Bui, Mohammad Ghavamzadeh, Georgios Theocharous, S. Muthukrishnan, and Siqi Sun. 2016. Graphical Model Sketch. In *ECML/PKDD (1) (Lecture Notes in Computer Science)*, Vol. 9851. Springer, 81–97.

[9] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. 2004. RCV1: A New Benchmark Collection for Text Categorization Research. *Journal of Machine Learning Research* 5 (2004), 361–397.

[10] Edo Liberty, Michael Mitzenmacher, Justin Thaler, and Jonathan Ullman. 2016. Space Lower Bounds for Itemset Frequency Sketches. In *PODS*. ACM, 441–454.

[11] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

[12] Andrew McGregor and Hoa T. Vu. 2015. Evaluating Bayesian Networks via Data Streams. In *COCOON (Lecture Notes in Computer Science)*, Vol. 9198. Springer, 731–743.

[13] Stuart J. Russell and Peter Norvig. 2010. *Artificial Intelligence - A Modern Approach*. Pearson Education.
[14] Jeffrey Scott Vitter. 1985. Random Sampling with a Reservoir. *ACM Trans. Math. Softw.* 11, 1 (1985), 37–57.
[15] David P. Woodruff. 2016. New Algorithms for Heavy Hitters in Data Streams (Invited Talk). In *ICDT (LIPIcs)*, Vol. 48. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 4:1–4:12.
[16] Yandex 2013. Yandex Personalized Web Search Challenge. https://www.kaggle.com/c/yandex-personalized-web-search-challenge. (2013).
[17] Guizhen Yang. 2004. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *KDD*. ACM, 344–353.