LEARNING-BASED FREQUENCY ESTIMATION ALGORITHMS

Chen-Yu Hsu, Piotr Indyk, Dina Katabi & Ali Vakilian
Computer Science and Artificial Intelligence Lab
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
{cyhsu,indyk,dk,vakilian}@mit.edu

ABSTRACT

Estimating the frequencies of elements in a data stream is a fundamental task in data analysis and machine learning. The problem is typically addressed using streaming algorithms which can process very large data using limited storage. Today's streaming algorithms, however, cannot exploit patterns in their input to improve performance. We propose a new class of algorithms that automatically learn relevant patterns in the input data and use them to improve its frequency estimates. The proposed algorithms combine the benefits of machine learning with the formal guarantees available through algorithm theory. We prove that our learning-based algorithms have lower estimation errors than their non-learning counterparts. We also evaluate our algorithms on two real-world datasets and demonstrate empirically their performance gains.

1. INTRODUCTION

Classical algorithms provide formal guarantees over their performance, but often fail to leverage useful patterns in their input data to improve their output. On the other hand, deep learning models are highly successful at capturing and utilizing complex data patterns, but often lack formal error bounds. The last few years have witnessed a growing effort to bridge this gap and introduce algorithms that can adapt to data properties while delivering worst case guarantees. Deep learning modules have been integrated into the design of Bloom filters (Kraska et al., 2018; Mitzenmacher, 2018), caching algorithms (Lykouris & Vassilvitskii, 2018), graph optimization (Dai et al., 2017), similarity search (Salakhutdinov & Hinton, 2009; Weiss et al., 2009) and compressive sensing (Bora et al., 2017). This paper makes a significant step toward this vision by introducing frequency estimation streaming algorithms that automatically learn to leverage the properties of the input data.

Estimating the frequencies of elements in a data stream is one of the most fundamental subroutines in data analysis. It has applications in many areas of machine learning, including feature selection (Aghazadeh et al., 2018), ranking (Dzogang et al., 2015), semi-supervised learning (Talukdar & Cohen, 2014) and natural language processing (Goyal et al., 2012). It has been also used for network measurements (Estan & Varghese, 2003; Yu et al., 2013; Liu et al., 2016) and security (Schechter et al., 2010). Frequency estimation algorithms have been implemented in popular data processing libraries, such as Algebird at Twitter (Boykin et al., 2016). They can answer practical questions like: what are the most searched words on the Internet? or how much traffic is sent between any two machines in a network?

The frequency estimation problem is formalized as follows: given a sequence S of elements from some universe U, for any element $i \in U$, estimate f_i, the number of times i occurs in S. If one could store all arrivals from the stream S, one could sort the elements and compute their frequencies. However, in big data applications, the stream is too large (and may be infinite) and cannot be stored. This challenge has motivated the development of *streaming algorithms*, which read the elements of S in a single pass and compute a good estimate of the frequencies using a limited amount of space.[1] Over the last two decades, many such streaming algorithms have been developed, including

[1] Specifically, the goal of the problem is as follows. Given a sequence S of elements from U, the desired algorithm reads S in a single pass while writing into memory C (whose size can be much smaller than the length of S). Then, given any element $i \in U$, the algorithm reports an estimation of f_i based only on the content of C.

Count-Sketch (Charikar et al., 2002), Count-Min (Cormode & Muthukrishnan, 2005b) and multi-stage filters (Estan & Varghese, 2003). The performance guarantees of these algorithms are well-understood, with upper and lower bounds matching up to $O(\cdot)$ factors (Jowhari et al., 2011).

However, such streaming algorithms typically assume generic data and do not leverage useful patterns or properties of their input. For example, in text data, the word frequency is known to be inversely correlated with the length of the word. Analogously, in network data, certain applications tend to generate more traffic than others. If such properties can be harnessed, one could design frequency estimation algorithms that are much more efficient than the existing ones. Yet, it is important to do so in a general framework that can harness various useful properties, instead of using handcrafted methods specific to a particular pattern or structure (e.g., word length, application type).

In this paper, we introduce learning-based frequency estimation streaming algorithms. Our algorithms are equipped with a learning model that enables them to exploit data properties without being specific to a particular pattern or knowing the useful property a priori. We further provide theoretical analysis of the guarantees associated with such learning-based algorithms.

We focus on the important class of "hashing-based" algorithms, which includes some of the most used algorithms such as Count-Min, Count-Median and Count-Sketch. Informally, these algorithms hash data items into B buckets, count the number of items hashed into each bucket, and use the bucket value as an estimate of item frequency. The process can be repeated using multiple hash functions to improve accuracy. Hashing-based algorithms have several useful properties. In particular, they can handle item deletions, which are implemented by decrementing the respective counters. Furthermore, some of them (notably Count-Min) never underestimate the true frequencies, i.e., $\tilde{f}_i \geq f_i$ holds always. However, hashing algorithms lead to estimation errors due to collisions: when two elements are mapped to the same bucket, they affect each others' estimates. Although collisions are unavoidable given the space constraints, the overall error significantly depends on the pattern of collisions. For example, collisions between high-frequency elements ("heavy hitters") result in a large estimation error, and ideally should be minimized. The existing algorithms, however, use random hash functions, which means that collisions are controlled only probabilistically.

Our idea is to use a small subset of S, call it S', to learn the heavy hitters. We can then assign heavy hitters their own buckets to avoid the more costly collisions. It is important to emphasize that we are learning the *properties* that identify heavy hitters as opposed to the *identities* of the heavy hitters themselves. For example, in the word frequency case, shorter words tend to be more popular. The subset S' itself may miss many of the popular words, but whichever words popular in S' are likely to be short. Our objective is *not* to learn the identity of high frequency words using S'. Rather, we hope that a learning model trained on S' learns that short words are more frequent, so that it can identify popular words even if they did not appear in S'.

Our main contributions are as follows:

- We introduce *learning-based* frequency estimation streaming algorithms, which learn the properties of heavy hitters in their input and exploit this information to reduce errors
- We provide performance guarantees showing that our algorithms can deliver a logarithmic factor improvement in the error bound over their non-learning counterparts. Furthermore, we show that our learning-based instantiation of Count-Min, a widely used algorithm, is asymptotically optimal among *all* instantiations of that algorithm. See Table 4.1 in section 4.1 for the details.
- We evaluate our learning-based algorithms using two real-world datasets: traffic load on an Internet backbone link and search query popularity. In comparison to their non-learning counterparts, our algorithms yield performance gains that range from 18% to 71%.

2. Related Work

Frequency estimation in data streams. Frequency estimation, and the closely related problem of finding frequent elements in a data stream, are some of the most fundamental and well-studied problems in streaming algorithms, see Cormode & Hadjieleftheriou (2008) for an overview. Hashing-based algorithms such as Count-Sketch (Charikar et al., 2002), Count-Min (Cormode & Muthukrishnan, 2005b) and multi-stage filters (Estan & Varghese, 2003) are widely used solutions for these problems. These algorithms also have close connections to sparse recovery and compressed sens-

ing (Candès et al., 2006; Donoho, 2006), where the hashing output can be considered as a compressed representation of the input data (Gilbert & Indyk, 2010).

Several "non-hashing" algorithms for frequency estimation have been also proposed (Misra & Gries, 1982; Demaine et al., 2002; Karp et al., 2003; Metwally et al., 2005). These algorithms do not possess many of the properties of hashing-based methods listed in the introduction (such as the ability to handle deletions), but they often have better accuracy/space tradeoffs. For a fair comparison, our evaluation focuses only on hashing algorithms. However, our approach for learning heavy hitters should be useful for non-hashing algorithms as well.

Some papers have proposed or analyzed frequency estimation algorithms customized to data that follows Zipf Law (Charikar et al., 2002; Cormode & Muthukrishnan, 2005a; Metwally et al., 2005; Minton & Price, 2014; Roy et al., 2016); the last algorithm is somewhat similar to the "lookup table" implementation of the heavy hitter oracle that we use as a baseline in our experiments. Those algorithms need to know the data distribution a priori, and apply only to one distribution. In contrast, our learning-based approach applies to any data property or distribution, and does not need to know that property or distribution a priori.

Learning-based algorithms. Recently, researchers have begun exploring the idea of integrating machine learning models into algorithm design. In particular, researchers have proposed improving compressed sensing algorithms, either by using neural networks to improve sparse recovery algorithms (Mousavi et al., 2017; Bora et al., 2017), or by designing linear measurements that are optimized for a particular class of vectors (Baldassarre et al., 2016; Mousavi et al., 2015), or both. The latter methods can be viewed as solving a problem similar to ours, as our goal is to design "measurements" of the frequency vector $(f_1, f_2 \ldots, f_{|U|})$ tailored to a particular class of vectors. However, the aforementioned methods need to explicitly represent a matrix of size $B \times |U|$, where B is the number of buckets. Hence, they are unsuitable for streaming algorithms which, by definition, have space limitations much smaller than the input size.

Another class of problems that benefited from machine learning is *distance estimation*, i.e., compression of high-dimensional vectors into compact representations from which one can estimate distances between the original vectors. Early solutions to this problem, such as *Locality-Sensitive Hashing*, have been designed for worst case vectors. Over the last decade, numerous methods for learning such representations have been developed (Salakhutdinov & Hinton, 2009; Weiss et al., 2009; Jegou et al., 2011; Wang et al., 2016). Although the objective of those papers is similar to ours, their techniques are not usable in our applications, as they involve a different set of tools and solve different problems.

More broadly, there have been several recent papers that leverage machine learning to design more efficient algorithms. The authors of (Dai et al., 2017) show how to use reinforcement learning and graph embedding to design algorithms for graph optimization (e.g., TSP). Other learning-augmented combinatorial optimization problems are studied in (He et al., 2014; Balcan et al., 2018; Lykouris & Vassilvitskii, 2018). More recently, (Kraska et al., 2018; Mitzenmacher, 2018) have used machine learning to improve indexing data structures, including Bloom filters that (probabilistically) answer queries of the form "is a given element in the data set?" As in those papers, our algorithms use neural networks to learn certain properties of the input. However, we differ from those papers both in our design and theoretical analysis. Our algorithms are designed to reduce collisions between heavy items, as such collisions greatly increase errors. In contrast, in existence indices, all collisions count equally. This also leads to our theoretical analysis being very different from that in (Mitzenmacher, 2018).

3. Preliminaries

3.1. Estimation Error

We will use $e_i := |\tilde{f}_i - f_i|$ to denote the estimation error for f_i. To measure the overall estimation error between the frequencies $\mathcal{F} = \{f_1, f_2, \cdots, f_{|U|}\}$ and their estimates $\tilde{\mathcal{F}} = \{\tilde{f}_1, \tilde{f}_2, \cdots, \tilde{f}_{|U|}\}$, we will use the *expected* error $E_{i\sim\mathcal{D}}[e_i]$, where $\mathcal{D}$ models the distribution over the *queries* to the data structure. Similar to past work (Roy et al., 2016), we assume the query distribution $\mathcal{D}$ is the same as the distribution of the input stream, i.e., for any j we have $\Pr_{i\sim\mathcal{D}}[i = j] = f_j/N$, where N

is the sum of all frequencies. This leads to the estimation error of $\tilde{\mathcal{F}}$ with respect to $\mathcal{F}$:

$$\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}) := 1/N \sum_{i \in U} |\tilde{f}_i - f_i| \cdot f_i \tag{3.1}$$

We note that the theoretical guarantees of frequency estimation algorithms are typically phrased in the "(ϵ, δ)-form", e.g., $\Pr[|\tilde{f}_i - f_i| > \epsilon N] < \delta$ for *every* i (see e.g., Cormode & Muthukrishnan (2005b)). However, this formulation involves two objectives (ϵ and δ). We believe that the (single objective) expected error in Equation 3.1 is more natural from the machine learning perspective.

3.2. Algorithms for Frequency Estimation

In this section, we recap three variants of hashing-based algorithms for frequency estimation.

Single Hash Function. The basic hashing algorithm uses a single uniformly random hash function $h : U \to [B]$, where we use $[B]$ to denote the set $\{1 \dots B\}$. The algorithm maintains a one-dimensional array $C[1 \dots B]$, in which all entries are initialized to 0. Given an element i, the algorithm increments $C[h(i)]$. It can be seen that at the end of the stream, we have $C[b] = \sum_{j:h(j)=b} f_j$. The estimate $\tilde{f}_i$ of f_i is defined as $\tilde{f}_i = C[h(i)] = \sum_{j:h(i)=h(j)} f_j$. Note that it is always the case that $\tilde{f}_i \geq f_i$.

Count-Min. We have k distinct hash functions $h_i : U \to [B]$ and an array C of size $k \times B$. The algorithm maintains C, such that at the end of the stream we have $C[\ell, b] = \sum_{j:h_\ell(j)=b} f_j$. For each $i \in U$, the frequency estimate $\tilde{f}_i$ is equal to $\min_{\ell \leq k} C[\ell, h_\ell(i)]$, and always satisfies $\tilde{f}_i \geq f_i$.

Count-Sketch. Similarly to Count-Min, we have k distinct hash functions $h_i : U \to [B]$ and an array C of size $k \times B$. Additionally, in Count-Sketch, we have k sign functions $g_i : U \to \{-1, 1\}$, and the algorithm maintains C such that $C[\ell, b] = \sum_{j:h_\ell(j)=b} f_j \cdot g_\ell(j)$. For each $i \in U$, the frequency estimate $\tilde{f}_i$ is equal to the median of $\{g_\ell(i) \cdot C[\ell, h_\ell(i)]\}_{\ell \leq k}$. Note that unlike the previous two methods, here we may have $\tilde{f}_i < f_i$.

3.3. Zipfian Distribution

In our theoretical analysis we assume that the item frequencies follow the Zipf Law. That is, if we re-order the items so that their frequencies appear in a sorted order $f_{i_1} \geq f_{i_2} \geq \dots \geq f_{i_n}$, then $f_{i_j} \propto 1/j$. To simplify the notation we assume that $f_i = 1/i$.

4. Learning-Based Frequency Estimation Algorithms

We aim to develop frequency estimation algorithms that exploit data properties for better performance. To do so, we learn an oracle that identifies heavy hitters, and use the oracle to assign each heavy hitter its unique bucket to avoid collisions. Other items are simply hashed using any classic frequency estimation algorithm (e.g., Count-Min, or Count-Sketch), as shown in the block-diagram in Figure 4.1. This design has two useful properties: First, it allows us to augment a classic frequency estimation algorithm with learning capabilities, producing a learning-based counterpart that inherits the original guarantees of the classic algorithm. For example, if the classic algorithm is Count-Min, the resulting learning-based algorithm never underestimates the frequencies. Second, it provably reduces the estimation errors, and for the case of Count-Min it is (asymptotically) optimal.

Algorithm 1 provides pseudo code for our design. The design assumes an oracle $\mathsf{HH}(i)$ that attempts to determine whether an item i is a "heavy hitter" or not. All items classified as heavy hitters are assigned to one of the B_r *unique* buckets reserved for heavy items. All other items are fed to the remaining $B - B_r$ buckets using a conventional frequency estimation algorithm $SketchAlg$ (e.g., Count-Min or Count-Sketch).

The estimation procedure is analogous. To compute $\tilde{f}_i$, the algorithm first checks whether i is stored in a unique bucket, and if so, reports its count. Otherwise, it queries the $SketchAlg$ procedure. Note that if the element is stored in a unique bucket, its reported count is exact, i.e., $\tilde{f}_i = f_i$.

The oracle is constructed using machine learning and trained with a small subset of S, call it S'. Note that the oracle learns the *properties* that identify heavy hitters as opposed to the *identities* of the heavy hitters themselves. For example, in the case of word frequency, the oracle would learn that

shorter words are more frequent, so that it can identify popular words even if they did not appear in the training set S'.

4.1. Analysis

Our algorithms combine simplicity with strong error bounds. Below, we summarize our theoretical results, and leave all theorems, lemmas, and proofs to the appendix. In particular, Table 4.1 lists the results proven in this paper, where each row refers to a specific streaming algorithm, its corresponding error bound, and the theorem/lemma that proves the bound.

First, we show (Theorem 9.11 and Theorem 9.14) that if the heavy hitter oracle is accurate, then the error of the learned variant of Count-Min is up to a logarithmic factor smaller than that of its non-learning counterpart. The improvement is maximized when B is of the same order as n (a common scenario[2]). Furthermore, we prove that this result continues to hold even if the learned oracle makes prediction errors with probability δ, as long as $\delta = O(1/\ln n)$ (Lemma 9.15).

Second, we show that, asymptotically, our learned Count-Min algorithm cannot be improved any further by designing a better hashing scheme. Specifically, for the case of Learned Count-Min with a perfect oracle, our design achieves the same asymptotic error as the "Ideal Count-Min", which optimizes its hash function for the given input (Theorem 10.4).

Finally, we note that the learning-augmented algorithm inherits any (ϵ, δ)-guarantees of the original version. Specifically, its error is not larger than that of $SketchAlg$ with space $B - B_r$, for any input.

Algorithms	**Expected Error**	**Analysis**
Single Hash Function (B)	$\Theta(\ln^2 n/B)$	Lemma 9.2
Count-Min Sketch (k, B)	$\Omega(\frac{k \ln n}{B \ln k})$ and $O(\frac{k \ln n \ln^{\frac{k+2}{k-1}}(\frac{kn}{B})}{B})$	Theorem 9.11
Learned Count-Min(B)	$\Theta(\ln^2(n/B)/B)$	Theorem 9.14
Ideal Count-Min(B)	$\Theta(\ln^2(n/B)/B)$	Theorem 10.4

Table 4.1: Our performance bounds for different algorithms on streams with frequencies obeying Zipf Law. k is a constant (≥ 2) that refers to the number of hash functions, B is the number of buckets, and n is the number of distinct elements. The space complexity of all algorithms is the same, $\Theta(B)$. See section 9.4 for non-asymptotic versions of the some of the above bounds

5. Experiments

Baselines. We compare our learning-based algorithms with their non-learning counterparts. Specifically, we augment Count-Min with a learned oracle using Algorithm 1, and call the learning-augmented algorithm "Learned Count-Min". We then compare Learned Count-Min with traditional Count-Min. We also compare it with "Learned Count-Min with Ideal Oracle" where the neural-network oracle is replaced with an ideal oracle that knows the identities of the heavy hitters in the test data, and "Table Lookup with Count-Min" where the heavy hitter oracle is replaced with a lookup table that memorizes heavy hitters in the training set. The comparison with the latter baseline allows

[2]For example, Goyal et al. (2012) uses $B = 20M, n = 33.5M$ and $B = 50M, n = 94M$.

Algorithm 1 Learning-Based Frequency Estimation

procedure LEARNEDSKETCH(B, B_r, HH, $SketchAlg$)
  for each stream element i **do**
    if HH$(i) = 1$ **then**
      if i is already stored in a unique bucket **then**
        increment the count of i
      else create a new unique bucket for i and
        initialize the count to 1
      end if
    else
      feed i to $SketchAlg$ with $B - B_r$ buckets
    end if
  end for
end procedure

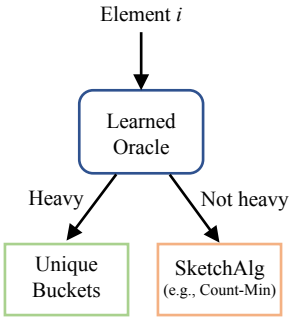


Figure 4.1: Pseudo-code and block-diagram representation of our algorithms

us to show the ability of Learned Count-Min to generalize and detect heavy items unseen in the training set. We repeat the evaluation where we replace Count-Min (CM) with Count-Sketch (CS) and the corresponding variants. We use validation data to select the best k for all algorithms.

Training a Heavy Hitter Oracle. We construct the heavy hitter oracle by training a neural network to predict the heaviness of an item. Note that the prediction of the network is not the final estimation. It is used in Algorithm 1 to decide whether to assign an item to a unique bucket. We train the network to predict the item counts (or the log of the counts) and minimize the squared loss of the prediction. Empirically, we found that when the counts of heavy items are few orders of magnitude larger than the average counts (as is the case for the Internet traffic data set), predicting the log of the counts leads to more stable training and better results. Once the model is trained, we select the optimal cutoff threshold using validation data, and use the model as the oracle described in Algorithm 1.

5.1. Internet Traffic Estimation

For our first experiment, the goal is to estimate the number of packets for each network flow. A flow is a sequence of packets between two machines on the Internet. It is identified by the IP addresses of its source and destination and the application ports. Estimating the size of each flow i – i.e., the number of its packets f_i – is a basic task in network management (Sivaraman et al., 2017).

Dataset: The traffic data is collected at a backbone link of a Tier1 ISP between Chicago and Seattle in 2016 (CAIDA). Each recording session is around one hour. Within each minute, there are around 30 million packets and 1 million unique flows. For a recording session, we use the first 7 minutes for training, the following minute for validation, and estimate the packet counts in subsequent minutes. The distribution of packet counts over Internet flows is heavy tailed, as shown in Figure 5.1.

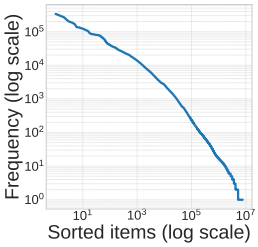


Figure 5.1: Frequency of Internet Flows

Model: The patterns of the Internet traffic are very dynamic, i.e., the flows with heavy traffic change frequently from one minute to the next. However, we hypothesize that the space of IP addresses should be smooth in terms of traffic load. For example, data centers at large companies and university campuses with many students tend to generate heavy traffic. Thus, though the individual flows from these sites change frequently, we could still discover regions of IP addresses with heavy traffic through a learning approach.

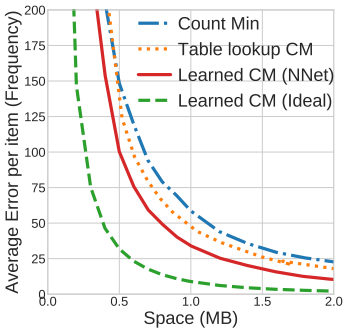
We trained a neural network to predict the log of the packet counts for each flow. The model takes as input the IP addresses and ports in each packet. We use two RNNs to encode the source and destination IP addresses separately. The RNN takes one bit of the IP address at each step, starting from the most significant bit. We use the final states of the RNN as the feature vector for an IP address. The reason to use RNN is that the patterns in the bits are hierarchical, i.e., the more significant bits govern larger regions in the IP space. Additionally, we use two-layer fully-connected networks to encode the source and destination ports. We then concatenate the encoded IP vectors, encoded port vectors, and the protocol type as the final features to predict the packet counts [3]. The inference time takes 2.8 microseconds per item on a single GPU without any optimizations[4].

Results: We plot the results of two representative test minutes (the 20th and 50th) in Figure 5.2. All plots in the figure refer to the estimation error (Equation 3.1) as a function of the used space. The space includes space for storing the buckets and the model. Since we use the same model for all test minutes, the model space is amortized over the 50-minute testing period.

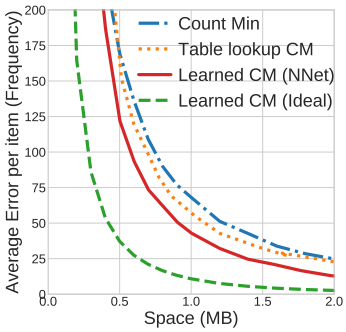
Figure 5.2 reveals multiple findings. First, the figure shows that our learning-based algorithms exhibit a better performance than their non-learning counterparts. Specifically, Learned Count-Min, compared to Count-Min, reduces the the error by 32% with space of 0.5 MB and 42% with space of 1.0 MB (Figure 5.2a). Learned Count-Sketch, compared to Count-Sketch, reduces the error by 52%

[3] We use RNNs with 64 hidden units. The two-layer fully-connected networks for the ports have 16 and 8 hidden units. The final layer before the prediction has 32 hidden units.

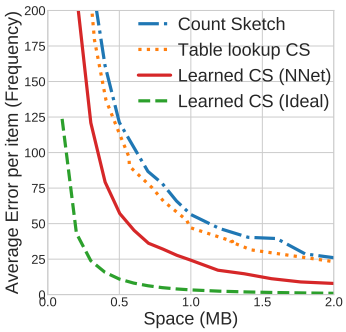
[4] Note that new specialized hardware such as Google TPU, hardware accelerators and network compression (Han et al., 2017; Sze et al., 2017; Chen et al., 2017; Han et al., 2016; 2015) can drastically improve the inference time. Further, Nvidia has predicted that GPU will get 1000x faster by 2025. Because of these trends, the overhead of neural network inference is expected to be less significant in the future (Kraska et al., 2018).



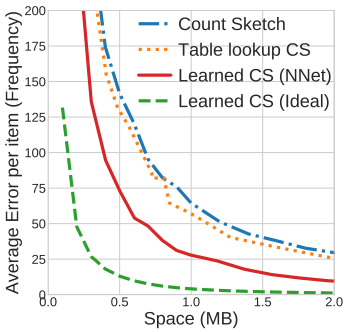
(a) Learned Count-Min - 20th test minute



(b) Learned Count-Min - 50th test minute



(c) Learned Count-Sketch - 20th test minute



(d) Learned Count-Sketch - 50th test minute

Figure 5.2: Comparison of our algorithms with Count-Min and Count-Sketch on Internet traffic data.

at 0.5 MB and 57% at 1.0 MB (Figure 5.2c). In our experiments, each regular bucket takes 4 bytes. For the learned versions, we account for the extra space needed for the unique buckets to store the item IDs and the counts. One unique bucket takes 8 bytes, twice the space of a normal bucket.[5]

Second, the figure also shows that our neural-network oracle performs better than memorizing the heavy hitters in a lookup table. This is likely due to the dynamic nature of Internet traffic –i.e., the heavy flows in the training set are significantly different from those in the test data. Hence, memorization does not work well. On the other hand, our model is able to extract structures in the input that generalize to unseen test data.

Third, the figure shows that our model's performance stays roughly the same from the 20th to the 50th minute (Figure 5.2b and Figure 5.2d), showing that it learns properties of the heavy items that generalize over time.

Lastly, although we achieve significant improvement over Count-Min and Count-Sketch, our scheme can potentially achieve even better results with an ideal oracle, as shown by the dashed green line in Figure 5.2. This indicates potential gains from further optimizing the neural network model.

5.2. Search Query Estimation

For our second experiment, the goal is to estimate the number of times a search query appears.

Dataset: We use the AOL query log dataset, which consists of 21 million search queries collected from 650 thousand users over 90 days. The users are anonymized in the dataset. There are 3.8 million unique queries. Each query is a search phrase with multiple words (e.g., "periodic table element poster"). We use the first 5 days for training, the following day for validation, and estimate the number of times different search queries appear in subsequent days. The distribution of search query frequency follows the Zipfian law, as shown in Figure 5.3

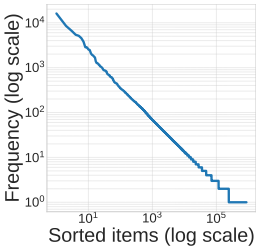
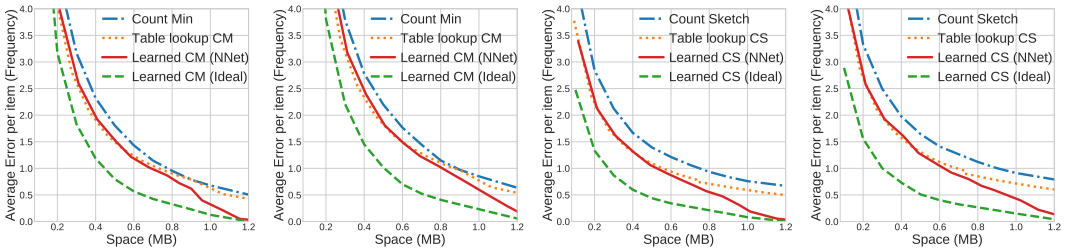


Figure 5.3: Frequency of search queries

[5] By using hashing with open addressing, it suffices to store IDs hashed into $\log B_r + t$ bits (instead of whole IDs) to ensure there is no collision with probability $1 - 2^{-t}$. $\log B_r + t$ is comparable to the number of bits per counter, so the space for a unique bucket is twice the space of a normal bucket.



(a) Learned Count-Min 50th test day (b) Learned Count-Min 80th test day (c) Learned Count-Sketch 50th test day (d) Learned Count-Sketch 80th test day

Figure 5.4: Comparison of our algorithms with Count-Min and Count-Sketch on search query data.

Model: Unlike traffic data, popular search queries tend to appear more consistently across multiple days. For example, "google" is the most popular search phrase in most of the days in the dataset. Simply storing the most popular words can easily construct a reasonable heavy hitter predictor. However, beyond remembering the popular words, other factors also contribute to the popularity of a search phrase that we can learn. For example, popular search phrases appearing in slightly different forms may be related to similar topics. Though not included in the AOL dataset, in general, metadata of a search query (e.g., the location of the search) can provide useful context of its popularity.

To construct the heavy hitter oracle, we trained a neural network to predict the number of times a search phrase appears. To process the search phrase, we train an RNN with LSTM cells that takes characters of a search phrase as input. The final states encoded by the RNN are fed to a fully-connected layer to predict the query frequency. Our character vocabulary includes lower-case English alphabets, numbers, punctuation marks, and a token for unknown characters. We map the character IDs to embedding vectors before feeding them to the RNN[6]. We choose RNN due to its effectiveness in processing sequence data (Sutskever et al., 2014; Graves, 2013; Kraska et al., 2018).

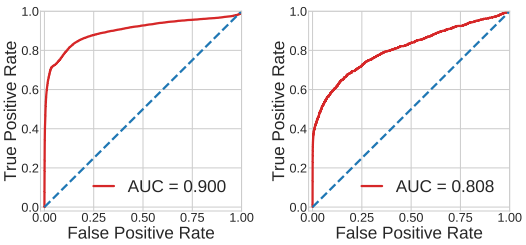
Results: We plot the estimation error vs. space for two representative test days (the 50th and 80th day) in Figure 5.4. As before, the space includes both the bucket space and the space used by the model. The model space is amortized over the test days since the same model is used for all days.

Similarly, our learned sketches outperforms their conventional counterparts. For Learned Count-Min, compared to Count-Min, it reduces the loss by 18% at 0.5 MB and 52% at 1.0 MB (Figure 5.4a). For Learned Count-Sketch, compared to Count-Sketch, it reduces the loss by 24% at 0.5 MB and 71% at 1.0 MB (Figure 5.4c). Further, our algorithm performs similarly for the 50th and the 80th day (Figure 5.4b and Figure 5.4d), showing that the properties it learns generalize over a long period.

The figures also show an interesting difference from the Internet traffic data: memorizing the heavy hitters in a lookup table is quite effective in the low space region. This is likely because the search queries are less dynamic compared to Internet traffic (i.e., top queries in the training set are also popular on later days). However, as the algorithm is allowed more space, memorization becomes ineffective.

5.3. Analyzing Heavy Hitter Models

We analyze the accuracy of the neural network heavy hitter models to better understand the results on the two datasets. Specifically, we use the models to predict whether an item is a heavy hitter (top 1% in counts) or not, and plot the ROC curves in Figure 5.5. The figures show that the model for the Internet traffic data has learned to predict heavy items more effectively, with an AUC score of 0.9. As for the model for search query data, the AUC score is 0.8. This also explains why we see larger improvements over non-learning algorithms in Figure 5.2.



(a) Internet traffic 50th test minute (b) Search query 50th test day

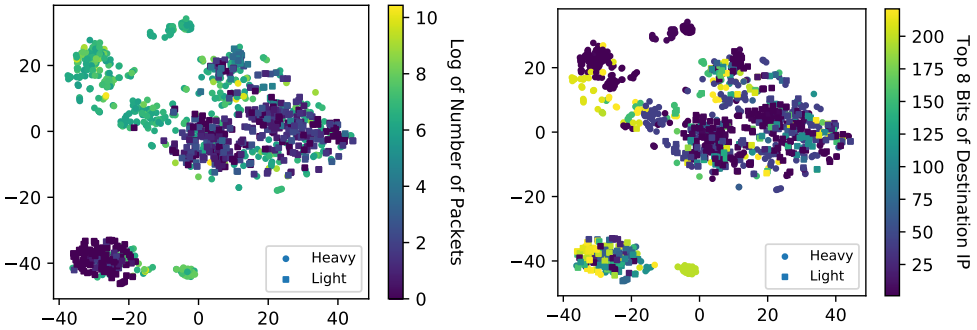
Figure 5.5: ROC curves of the learned models.

[6]We use an embedding size of 64 dimensions, an RNN with 256 hidden units, and a fully-connected layer with 32 hidden units.

6. EMBEDDING SPACE VISUALIZATION

In this section, we visualize the embedding spaces learned by our heavy hitter models to shed light on the properties or structures the models learned. Specifically, we take the neural network activations before the final fully-connected layer, and visualize them in a 2-dimensional space using t-SNE (Maaten & Hinton, 2008). To illustrate the differences between heavy hitters (top 1% in counts) and the rest ("light" items), we randomly sample an equal amount of examples from both classes. We visualize the embedding space for both the Internet traffic and search query datasets.

We show the embedding space learned by the model on the Internet traffic data in Figure 6.1. Each point in the scatter plot represents one Internet traffic flow. By coloring each flow with its number of packets in Figure 6.1a, we see that the model separate flows with more packets (green and yellow clusters) from flows with fewer packets (blue clusters). To understand what structure the model learns to separate these flows, we color each flow with its destination IP address in Figure 6.1b. We found that clusters with more packets are often formed by flows sharing similar destination address prefixes. Interestingly, the model learns to group flows with similar IP prefixes closer in the embedding space. For example, the dark blue cluster at the upper left of Figure 6.1b shares a destination IP address prefix "1.96.*.*". Learning this "structure" from the Internet traffic data allows the model to generalize to packets unseen in the training set.

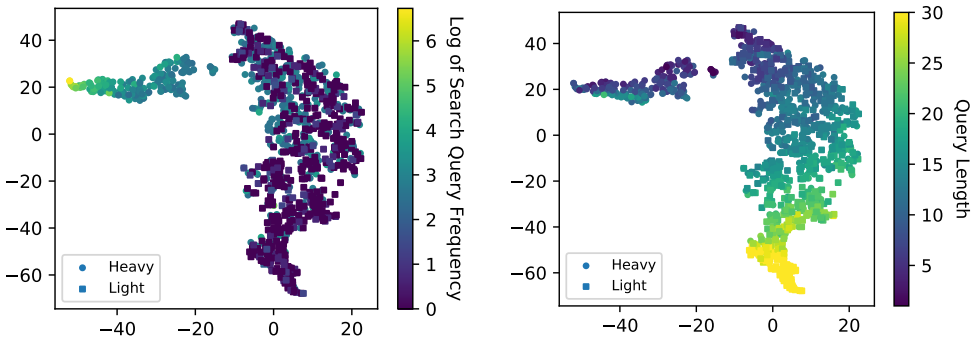


(a) Colored by log of number of packets

(b) Colored by the top 8 bits of destination IP

Figure 6.1: Visualization of the embedding space learned by our model on the Internet traffic data. Each point in the scatter plot represents one network flow.

We show the embedding space learned by the model on the search query data in Figure 6.2. Each point in the scatter plot represents one search query. Similarly, the model learns to separate frequent search queries from the rest in Figure 6.2a. By coloring the queries with the number of characters in Figure 6.2b, we have multiple interesting findings. First, queries with similar length are closer in the embedding space, and the y-axis forms the dimension representing query length. Second, if we simply use the query length to predict heavy hitters, many light queries will be misclassified. The model must have learned other structures to separate heavy hitters in Figure 6.2a.



(a) Colored by log of search query frequency

(b) Colored by query length

Figure 6.2: Visualization of the embedding space learned by our model on the search query data. Each point in the scatter plot represents one search query.

7. CONCLUSION

We have presented a new approach for designing frequency estimation streaming algorithms by augmenting them with a learning model that exploits data properties. We have demonstrated the benefits of our design both analytically and empirically. We envision that our work will motivate a deeper integration of learning in algorithm design, leading to more efficient algorithms.

8. ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for helpful comments. The work was partially supported by NSF TRIPODS award #1740751, NSF AITF award #1535851 and Simons Investigator Award.

REFERENCES

Amirali Aghazadeh, Ryan Spring, Daniel LeJeune, Gautam Dasarathy, Anshumali Shrivastava, and Richard G Baraniuk. Mission: Ultra large-scale feature selection using count-sketches. In *International Conference on Machine Learning*, 2018.

Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International Conference on Machine Learning*, 2018.

Luca Baldassarre, Yen-Huan Li, Jonathan Scarlett, Baran Gözcü, Ilija Bogunovic, and Volkan Cevher. Learning-based compressive subsampling. *IEEE Journal of Selected Topics in Signal Processing*, 10(4):809–822, 2016.

Ashish Bora, Ajil Jalal, Eric Price, and Alexandros G Dimakis. Compressed sensing using generative models. In *International Conference on Machine Learning*, pp. 537–546, 2017.

Oscar Boykin, Avi Bryant, Edwin Chen, ellchow, Mike Gagnon, Moses Nakamura, Steven Noble, Sam Ritchie, Ashutosh Singhal, and Argyris Zymnis. Algebird. `https://twitter.github.io/algebird/`, 2016.

CAIDA. Caida internet traces 2016 chicago. http://www.caida.org/data/monitors/passive-equinix-chicago.xml.

Emmanuel J Candès, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 52(2):489–509, 2006.

Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pp. 693–703. Springer, 2002.

Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, 2008.

Graham Cormode and S Muthukrishnan. Summarizing and mining skewed data streams. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pp. 44–55. SIAM, 2005a.

Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005b.

Hanjun Dai, Elias Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pp. 6351–6361, 2017.

Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. Frequency estimation of internet packet streams with limited space. In *European Symposium on Algorithms*, pp. 348–360. Springer, 2002.

David L Donoho. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

Fabon Dzogang, Thomas Lansdall-Welfare, Saatviga Sudhahar, and Nello Cristianini. Scalable preference learning from data streams. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 885–890. ACM, 2015.

Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems (TOCS)*, 21(3): 270–313, 2003.

Anna Gilbert and Piotr Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98 (6):937–947, 2010.

Amit Goyal, Hal Daumé III, and Graham Cormode. Sketch algorithms for estimating point queries in nlp. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 1093–1103. Association for Computational Linguistics, 2012.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pp. 243–254. IEEE, 2016.

Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 75–84. ACM, 2017.

He He, Hal Daume III, and Jason M Eisner. Learning to search in branch and bound algorithms. In *Advances in neural information processing systems*, pp. 3293–3301, 2014.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2011.

Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 49–58. ACM, 2011.

Richard M Karp, Scott Shenker, and Christos H Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems (TODS)*, 28(1): 51–55, 2003.

Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*, pp. 489–504. ACM, 2018.

Abhishek Kumar, Minho Sung, Jun Jim Xu, and Jia Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *ACM SIGMETRICS Performance Evaluation Review*, volume 32, pp. 177–188. ACM, 2004.

Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 101–114. ACM, 2016.

Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, 2018.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *International Conference on Database Theory*, pp. 398–412. Springer, 2005.

Gregory T Minton and Eric Price. Improved concentration bounds for count-sketch. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pp. 669–686. SIAM, 2014.

Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2 (2):143–152, 1982.

Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, 2018.

Ali Mousavi, Ankit B Patel, and Richard G Baraniuk. A deep learning approach to structured signal recovery. In *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*, pp. 1336–1343. IEEE, 2015.

Ali Mousavi, Gautam Dasarathy, and Richard G Baraniuk. Deepcodec: Adaptive sensing and recovery via deep convolutional neural networks. *arXiv preprint arXiv:1707.03386*, 2017.

Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the 2016 International Conference on Management of Data*, pp. 1449–1463. ACM, 2016.

Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

Stuart Schechter, Cormac Herley, and Michael Mitzenmacher. Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In *Proceedings of the 5th USENIX conference on Hot topics in security*, pp. 1–8. USENIX Association, 2010.

Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S Muthukrishnan, and Jennifer Rexford. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*, pp. 164–176. ACM, 2017.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

Partha Talukdar and William Cohen. Scaling graph-based semi supervised learning to large number of labels using count-min sketch. In *Artificial Intelligence and Statistics*, pp. 940–947, 2014.

Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - a survey. *Proceedings of the IEEE*, 104(1):34–57, 2016.

Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Advances in neural information processing systems*, pp. 1753–1760, 2009.

Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *NSDI*, volume 13, pp. 29–42, 2013.

9. ANALYSIS - FULL PROOFS

In this section, we analyze the performance of three different approaches, single (uniformly random) hash function, Count-Min sketch, and Learned Count-Min sketch when the frequency of items is from Zipfian distribution. For simplicity, we assume that the number of distinct elements n is equal to the size of the universe $|U|$, and $f_i = 1/i$. We use $[n]$ to denote the set $\{1 \dots n\}$. We also drop the normalization factor $1/N$ in the definition of estimation error.

The following observation is useful throughout this section (in particular, in the section on *non-asymptotic analysis*).

Observation 9.1. *For sufficiently large values of n (i.e., $n > 250$),*

$$\ln(n+1) < \sum_{i=1}^{n} \frac{1}{i} < \ln n + 0.58 \tag{9.1}$$

9.1. SINGLE HASH FUNCTION

Lemma 9.2. *The expected error of a single uniformly random hash function $h : [n] \to [B]$ (with B buckets) for estimating the frequency of items whose distribution is Zipfian is $\Theta(\frac{\ln^2 n}{B})$.*

Proof:

$$\mathbf{E}[\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_h)] = \mathbf{E}[\sum_{j\in[n]} (\tilde{f}_j - f_j) \cdot f_j] = \sum_{j\in[n]} \mathbf{E}[\sum_{\{q\neq j \mid h[q]=h[j]\}} f_q] \cdot f_j$$
$$= \sum_{j\in[n]} (\Theta(\frac{\ln n}{B}) - \frac{f_j}{B}) \cdot f_j = \Theta(\frac{\ln^2 n}{B}). \qquad \square$$

Moreover, since each bucket maintains the frequency of items that are mapped to it under h, the space complexity of this approach is proportional to the number of buckets which is $\Theta(B)$.

9.2. COUNT-MIN SKETCH

Here, we provide an upper bound and lower bound for the expected estimation error of Count-Min sketch with k hash functions and B buckets per row. In the rest of this section, for each $j \in [n], \ell \leq k$, we use $e_{j,\ell}$ and e_j respectively to denote the estimation error of f_j by h_ℓ and Count-Min sketch. Recall that the expected error of Count-Min sketch is defined as follows:

$$\mathbf{E}[\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{CM})] = \mathbf{E}[\sum_{j\in[n]} (\tilde{f}_j - f_j) \cdot f_j] = \sum_{j\in[n]} \mathbf{E}[e_j] \cdot f_j, \tag{9.2}$$

Our high-level approach is to partition the interval $[0, B\ln n]$ into $m+1$ smaller intervals by a sequence of thresholds $\Theta(\ln^{1+\gamma}(\frac{n}{B})) = r_0 \leq \cdots \leq r_m = B\ln n$ where γ is a parameter to be determined later. Formally, we define the sequence of r_is to satisfy the following property:

$$\forall \ell > 0, \quad r_\ell := (\ln(\frac{n}{B}) + \ln r_{\ell+1}) \ln^\gamma r_{\ell+1}. \tag{9.3}$$

Claim 9.3. *For each $i \geq 0$, $\ln r_{i+1} \geq 2 \ln r_i$.*

Proof: By (9.3) and assuming $\ln r_{i+1} \geq \ln(\frac{n}{B})$, $r_i < 2\ln^{1+\gamma} r_{i+1}$. Hence, $\ln r_{i+1} > (\frac{r_i}{2})^{\frac{1}{1+\gamma}} > 2\ln r_i$ for sufficiently large values of r_i[7] assuming $\gamma \leq 3$.

Note that as long as $\ln r_{i+1} \geq \ln(\frac{n}{B})$, $r_i \leq r_{i+1}$. Otherwise, $r_i = o(\ln^{1+\gamma}(\frac{n}{B})) < r_0$. $\square$

Corollary 9.4. *For each $i \geq 1$ and $c \geq 1$, $\sum_{j\geq i} \ln^{-c} r_j = O(\ln^{-c} r_i)$.*

[7] More precisely, $r_i \geq 1.3 \times 10^6$.

Then, to compute (9.2), we rewrite $\mathbf{E}[e_j]$ using the thresholds $r_0, \cdots, r_m$ as follows:

$$\begin{aligned}\mathbf{E}[e_j] &= \int_0^{\infty} \Pr(e_j \geq x)dx \qquad (9.4)\\ &= \int_0^{\frac{r_0}{B}} \Pr(e_j \geq x)dx + \sum_{i=0}^{m-1} \int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} \Pr(e_j \geq x)dx \\ &\leq O(\frac{r_0}{B}) + \sum_{i=0}^{m-1} \int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} \Big(\Pr(e_j \geq x | e_j < \frac{r_{i+1}}{B}) + \sum_{q=i+1}^{m-1} \Pr(e_j \geq \frac{r_q}{B} | e_j < \frac{r_{q+1}}{B})\Big)dx\end{aligned}$$

Next, we compute $\Pr(e_j \geq \frac{t}{B} | e_j < \frac{r_{i+1}}{B})$ for each $t \in [r_i, r_{i+1})$.

Lemma 9.5. *For each $t \in [r_i, r_{i+1})$, $\Pr(e_j \geq \frac{t}{B} | e_j < \frac{r_{i+1}}{B}) = O\Big(\frac{k(\ln(\frac{n}{B}) + \ln r_{i+1})}{t \ln^{\gamma} r_{i+1}}\Big)$.*

Proof: First we prove the following useful observation.

Claim 9.6. *For each item j and hash function h_ℓ, $\mathbf{E}[e_{j,\ell} \mid e_{j,\ell} < \frac{r}{B}] \leq \frac{\ln(\frac{n}{B}) + \ln r}{B}$.*

Proof: Note that the condition $e_{j,\ell} < \frac{r}{B}$ implies that for all items $q < \frac{B}{r}$ (and $q \neq j$), $h_\ell(j) \neq h_\ell(q)$. Hence,

$$\mathbf{E}[e_{j,\ell} \mid e_{j,\ell} < \frac{r}{B}] \leq \sum_{q > \frac{B}{r}} f_q \cdot \frac{1}{B} \leq \frac{\ln n - \ln(\frac{B}{r})}{B} = \frac{\ln(\frac{n}{B}) + \ln r}{B}. \qquad \square$$

Thus, by Markov's inequality, for each item j and hash function h_ℓ,

$$\Pr(e_{j,\ell} > \frac{t}{B} \mid e_{j,\ell} \leq \frac{r}{B}) \leq \frac{\ln(\frac{n}{B}) + \ln r}{t}. \qquad (9.5)$$

Now, for each h_ℓ in Count-Min sketch, we bound the value of $\Pr(e_{j,\ell} \geq \frac{t}{B})$ where $t \in [r_i, r_{i+1})$:

$$\begin{aligned}\Pr(e_{j,\ell} \geq \frac{t}{B}) &\leq \sum_{q=i}^{m-1} \Pr(e_{j,\ell} \geq \frac{r_q}{B} | e_{j,\ell} < \frac{r_{q+1}}{B}) \\ &\leq \sum_{q=i}^{m-1} \frac{\ln(\frac{n}{B}) + \ln r_{q+1}}{r_q} = O(\frac{1}{\ln^{\gamma} r_{i+1}}) \triangleright \text{ by (9.5) and Corollary 9.4} \qquad (9.6)\end{aligned}$$

Hence, for $k \geq 2$,

$$\begin{aligned}\Pr(e_j \geq \frac{t}{B} | e_j < \frac{r_{i+1}}{B}) &\leq k \Pr(e_{j,1} \geq \frac{t}{B} | e_{j,1} < \frac{r_{i+1}}{B}) \Pi_{\ell=2}^{k} \Pr(e_{j,\ell} \geq \frac{t}{B}) \\ &= O\Big(\frac{k(\ln(\frac{n}{B}) + \ln r_{i+1})}{t \ln^{\gamma(k-1)} r_{i+1}}\Big) \triangleright \text{ by (9.5) and (9.6)} \qquad \square\end{aligned}$$

Next, for each item j, we bound the contribution of each interval $(\frac{r_i}{B}, \frac{r_{i+1}}{B})$ to $\mathbf{E}[e_j]$, namely $\int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} \Pr(e_j \geq x)dx$.

Claim 9.7. *For each $i \geq 0$, $\int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} \Pr(e_j \geq x)dx = O(\frac{k(\ln(\frac{n}{B}) + \ln r_{i+1})}{B \ln^{\gamma(k-1)-1} r_{i+1}})$.*

Proof:

$$\int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} \Pr(e_j \geq x)dx = \int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} \Big(\underbrace{\Pr(e_j \geq x | e_j < \frac{r_{i+1}}{B})}_{A} + \sum_{q=i+1}^{m-1} \underbrace{\Pr(e_j \geq \frac{r_q}{B} | e_j < \frac{r_{q+1}}{B})}_{B_q}\Big)dx$$

First, we compute $\int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} A dx$.

$$\begin{aligned}\int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} \Pr(e_j \geq x | e_j < \frac{r_{i+1}}{B})dx &\leq \int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} O(\frac{k(\ln(\frac{n}{B}) + \ln r_{i+1})}{(B \ln^{\gamma(k-1)} r_{i+1})x})dx \\ &= O(\frac{k(\ln(\frac{n}{B}) + \ln r_{i+1})}{B \ln^{\gamma(k-1)-1} r_{i+1}}) \qquad (9.7)\end{aligned}$$

Similarly, $\int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} \sum_{q=i+1}^{m-1} B_q dx$ is at most:

$$\begin{aligned}\int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} \sum_{q=i+1}^{m-1} \Pr(e_j \geq \frac{r_q}{B} | e_j < \frac{r_{q+1}}{B}) dx &= \int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} \sum_{q=i+1}^{m-1} O(\frac{k(\ln(\frac{n}{B}) + \ln r_{q+1})}{r_q \ln^{\gamma(k-1)} r_{q+1}}) dx \\ &= \int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} \sum_{q=i+1}^{m-1} O(\frac{k}{\ln^{\gamma k} r_{q+1}}) dx \quad \triangleright \text{ by (9.3)} \\ &= \int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} O(\frac{k}{\ln^{\gamma k} r_{i+2}}) dx \quad \triangleright \text{ by Corollary 9.4} \\ &= O(\frac{k r_{i+1}}{B \ln^{\gamma k} r_{i+2}}) = O(\frac{k(\ln(\frac{n}{B}) + \ln r_{i+2})}{B \ln^{\gamma(k-1)} r_{i+2}}) \end{aligned} \tag{9.8}$$

Hence,

$$\begin{aligned}\int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} \Pr(e_j \geq x) dx &= \int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} \Big(\Pr(e_j \geq x | e_j < \frac{r_{i+1}}{B}) + \sum_{q=i+1}^{m-1} \Pr(e_j \geq \frac{r_q}{B} | e_j < \frac{r_{q+1}}{B}) \Big) dx \\ &= O(\frac{k(\ln(\frac{n}{B}) + \ln r_{i+1})}{B \ln^{\gamma(k-1)-1} r_{i+1}}) + O(\frac{k(\ln(\frac{n}{B}) + \ln r_{i+2})}{B \ln^{\gamma(k-1)} r_{i+2}}) \quad \triangleright \text{ by (9.7)-(9.8)} \\ &= O(\frac{k(\ln(\frac{n}{B}) + \ln r_{i+1})}{B \ln^{\gamma(k-1)-1} r_{i+1}}) \quad \triangleright \text{ by (9.3)} \end{aligned} \tag{9.9}$$

□

Now, we complete the error analysis of (9.4):

$$\begin{aligned}\mathbf{E}[e_j] &\leq O(\frac{r_0}{B}) + \sum_{i=0}^{m-1} \int_{\frac{r_i}{B}}^{\frac{r_{i+1}}{B}} \Big(\Pr(e_j \geq x | e_j < \frac{r_{i+1}}{B}) + \sum_{q=i+1}^{m-1} \Pr(e_j \geq \frac{r_q}{B} | e_j < \frac{r_{q+1}}{B}) \Big) dx \\ &= O(\frac{r_0}{B}) + \sum_{i=0}^{m-1} O(\frac{k(\ln(\frac{n}{B}) + \ln r_{i+1})}{B \ln^{\gamma(k-1)-1} r_{i+1}}) \quad \triangleright \text{ by (9.9)} \\ &= O(\frac{r_0}{B}) + O(k \frac{\ln(\frac{n}{B})}{B \ln^{\gamma(k-1)-1} r_1}) + O(\frac{k}{B \ln^{\gamma(k-1)-2} r_1}) \end{aligned} \tag{9.10}$$

Note that (9.10) requires $\gamma(k-1) - 2 \geq 1$ which is satisfied by setting $\gamma = 3/(k-1)$ and $k \geq 2$. Thus, for each item j,

$$\mathbf{E}[e_j] = O(r_0/B) = O(\frac{\ln^{\frac{k+2}{k-1}}(\frac{n}{B})}{B}) \tag{9.11}$$

Lemma 9.8. *The expected error of Count-Min sketch of size $k \times B$ (with $k \geq 2$) for estimating items whose frequency distribution is Zipfian is $O(\frac{\ln n \ln^{\frac{k+2}{k-1}}(n/B)}{B})$. In particular, if $B = \Theta(n)$, then $\mathbf{E}[\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{CM})] = O(\frac{\ln n}{n})$.*

Proof: By plugging in our upper bound on the estimation error of each item computed in (9.11) in the definition of expected estimation error of Count-Min (9.2), we have the following.

$$\mathbf{E}[\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{CM})] = \sum_{j=1}^{n} \mathbf{E}[e_j] \cdot f_j = O(\frac{\ln^{\frac{k+2}{k-1}}(\frac{n}{B})}{B} \cdot \ln n).$$

□

Next, we show a lower bound on the expected error of Count-Min sketch with B buckets (more precisely, of size $(k \times B/k)$) for estimating the frequency of items that follow Zipf Law.

Observation 9.9. *For each item j, $\Pr[e_j \geq 1/(2(\frac{B}{k}) \ln k + 1)] \geq 1 - \frac{1}{k}$.*

Proof: For each item j, the probability that none of the first $2(\frac{B}{k}) \ln k + 1$ items (excluding itself) collide with it in at least one of the hash functions $h_1, \cdots, h_k$ is at most $k(1 - \frac{1}{(B/k)})^{2(\frac{B}{k}) \ln k} \leq \frac{1}{k}$.

Hence, with probability at least $1 - \frac{1}{k}$, the estimation $\tilde{f}_j$ includes the frequency of one of the top $2(\frac{B}{k}) \ln k + 1$ frequent items. □

Lemma 9.10. *The expected error of Count-Min sketch of size $k \times (\frac{B}{k})$ for estimating items whose frequency distribution is Zipfian is at least $\frac{(k-1)\ln n}{2B \ln k + k}$.*

Proof: Observation 9.9 implies that $\mathbf{E}[e_j] \geq (1 - \frac{1}{k}) \frac{1}{2(\frac{B}{k}) \ln k + 1} = \frac{k-1}{2B \ln k + k}$. Hence,

$$\begin{aligned}\mathbf{E}[\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{CM})] &= \sum_{j \in [n]} \mathbf{E}[e_j] \cdot f_j \\ &\geq (\frac{k-1}{2B \ln k + k}) \ln n \quad \triangleright \text{ By Observation 9.1}\end{aligned}$$

□

Theorem 9.11. *The expected error of Count-Min sketch of size $k \times (\frac{B}{k})$ on estimating the frequency of n items from Zipfian distribution is at least $\Omega(\frac{k \ln n}{B \ln k})$ and at most $O(\frac{k \ln n \ln^{\frac{k+2}{k-1}}(\frac{kn}{B})}{B})$.*

In particular, for the case $B = \Theta(n)$ and $k = O(1)$, the expected error of Count-Min sketch is $\Theta(\frac{\ln n}{B})$.

Proof: The proof follows from Lemma 9.8 and 9.10. We remark that the bound in Lemma 9.8 is for the expected estimation error of Count-Min sketch of size $k \times B$. Hence, to get the bound on the expected error of Count-Min of size $k \times (\frac{B}{k})$, we must replace B with B/k. □

9.3. Learned Count-Min Sketch

Definition 9.12 (ϕ-HeavyHitter). *Given a set of items $\mathcal{I} = \{i_1, \cdots, i_n\}$ with frequencies $\vec{f} = \langle f_1, \cdots, f_n \rangle$, an item j is a ϕ-HeavyHitter of $\mathcal{I}$ if $f_j \geq \phi ||\vec{f}||_1$.*

Remark 9.13. *If the frequency distribution of items $\mathcal{I}$ is Zipfian, then the number of ϕ-HeavyHitters is at most $1/(\phi \ln n)$. In other words, $B_r \leq (\phi \ln n)^{-1}$.*

To recall, in our Learned Count-Min sketch with parameters (B_r, B), B_r buckets are reserved for the frequent items returned by HH and the rest of items are fed to a Count-Min sketch of size $k \times (\frac{B - B_r}{k})$ where k is a parameter to be determined. We emphasize that the space complexity of Learned Count-Min sketch with parameter (B_r, B) is $B_r + B = O(B)$.

Theorem 9.14. *The optimal expected error of Learned Count-Min sketches with parameters (B_r, B) is at most $\frac{(\ln(n/B_r) + 0.58)^2}{B - B_r}$. In particular, for $B_r = \Theta(B) = \Theta(n)$, $\mathbf{E}[\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{L-CM})] = O(\frac{1}{n})$.*

Proof: Since, the count of top B_r frequent items are stored in their own buckets, for each $j \leq B_r$, $e_j = 0$. Hence,

$$\begin{aligned}\mathbf{E}[\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{L-CM})] &= \sum_{j \in [n]} \mathbf{E}[e_j] \cdot f_j \\ &= \sum_{j > B_r} \mathbf{E}[e_j] \cdot f_j \quad \triangleright \forall j \leq B_r, e_j = 0 \\ &< \frac{(\ln(n/B_r) + 0.58)^2}{B - B_r} \quad \triangleright \text{ By Observation 9.1}\end{aligned}$$

Note that the last inequality follows from the guarantee of single hash functions; in other words, setting $k = 1$ in the Count-Min sketch. □

9.3.1. Learned Count-Min Sketch Using Noisy HeavyHitters Oracle

Unlike the previous part, here we assume that we are given a *noisy* HeavyHitters oracle HH_δ such that for each item j, $\Pr\left(\mathsf{HH}_\delta(j, \frac{1}{B_r \ln n}) \neq \mathsf{HH}_0(j, \frac{1}{B_r \ln n})\right) \leq \delta$ where HH_0 is an ideal HeavyHitter oracle that detects heavy items with no error.

Lemma 9.15. *In an optimal Learned Count-Min sketch with parameters (B_r, B) and a noisy HeavyHitters oracle HH_δ, $\mathbf{E}[\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{(L-CM,\delta)})] = O(\frac{\delta^2 \ln^2 B_r + \ln^2(n/B_r)}{B - B_r})$.*

Proof: The key observation is that each heavy item, any of B_r most frequent items, may only misclassify with probability δ. Hence, for each item j classified as "*not heavy*",

$$\mathbf{E}[e_j] \leq \delta \cdot (\frac{\ln B_r + 1}{B - B_r}) + (\frac{\ln(n/B_r) + 1}{B - B_r}) = O(\frac{\delta \ln B_r + \ln(n/B_r)}{B - B_r}), \tag{9.12}$$

where the first term denotes the expected contribution of the misclassified heavy items and the second term denotes the expected contribution of non-heavy items.

The rest of analysis is similar to the proof of Theorem 9.14.

$$\begin{aligned}
\mathbf{E}[\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{(L-CM,\delta)})] &= \sum_{j \in [n]} \mathbf{E}[e_j] \cdot f_j \\
&< \delta \sum_{j \leq B_r} \mathbf{E}[e_j] \cdot f_j + \sum_{j > B_r} \mathbf{E}[e_j] \cdot f_j \\
&\leq O(\delta \cdot \ln B_r + \ln(n/B_r)) \cdot O(\frac{\delta \ln B_r + \ln(n/B_r)}{B - B_r}) \quad \triangleright \text{ By (9.12)} \\
&\leq O(\frac{\delta^2 \ln^2 B_r + \ln^2(n/B_r)}{B - B_r}).
\end{aligned}$$

□

Corollary 9.16. *Assuming $B_r = \Theta(B) = \Theta(n)$ and $\delta = O(1/\ln n)$, $\mathbf{E}[\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{(L-CM,\delta)})] = O(1/n)$.*

Space analysis. Here, we compute the amount of space that is required by this approach.

Lemma 9.17. *The amount of space used by Learned Count-Min sketch of size $k \cdot (\frac{B-B_r}{k})$ with cutoff (B_r reserved buckets) for estimating the frequency of items whose distribution is Zipfian is $O(B)$.*

Proof: The amount of space required to store the counters corresponding to functions $h_1, \cdots, h_k$ is $O(k \cdot (\frac{B-B_r}{k}))$. Here, we also need to keep a mapping from the heavy items (top B_r frequent items according to HH_δ) to the reserved buckets B_r which requires extra $O(B_r)$ space; each reserved buckets stores both the hashmap of its corresponding item and its count. □

9.4. Non-asymptotic Analysis of Count-Min and Learned Count-Min

In this section, we compare the non-asymptotic expected error of Count-Min sketch and out Learned Count-Min sketch with ideal HeavyHitters oracle. Throughout this section, we assume that the amount of available space to the frequency estimation algorithms is $(1+\alpha)B$ words. More precisely, we compare the expected error of Count-Min sketch with k hash functions and our Learned Count-Min sketch with $B_r = \alpha B$ reserved buckets. Recall that we computed the following bounds on the expected error of these approaches (Lemma 9.10 and Theorem 9.14):

$$\mathbf{E}[\mathrm{Err}_{CM}] \geq \frac{(k-1)\ln n}{2(1+\alpha)B\ln(k) + k}, \quad \mathbf{E}[\mathrm{Err}_{L-CM}] \leq \frac{(\ln(\frac{n}{\alpha B}) + 0.58)^2}{(1-\alpha)B}.$$

In the rest of this section, we assume that $B \geq \gamma n$ and then compute the minimum value of γ that guarantees $\mathbf{E}[\mathrm{Err}_{L-CM}] \leq \frac{1}{1+\varepsilon} \cdot \mathbf{E}[\mathrm{Err}_{CM}]$. In other words, we compute the minimum amount of space that is required so that our Learned Count-Min sketch performs better than Count-Min sketch by a factor of at least $(1+\varepsilon)$.

$$\mathbf{E}[\mathrm{Err}_{L-CM}] \leq \frac{(\ln(\frac{1}{\alpha\gamma}) + 0.58)^2}{(1-\alpha)B} \leq \frac{1}{1+\varepsilon} \cdot \frac{(k-1)\ln n}{2(1+\alpha)B\ln k + k} \leq \frac{1}{1+\varepsilon} \cdot \mathbf{E}[\mathrm{Err}_{CM}],$$

Hence, we must have $(0.58 - \ln\alpha - \ln\gamma)^2 \leq \frac{(k-1)(1-\alpha)}{(1+\varepsilon)(2(1+\alpha)\ln k + (\frac{k}{B}))} \cdot \ln n$. By solving the corresponding quadratic equation,

$$\ln^2\gamma + 2(\ln\alpha - 0.58)\ln\gamma + ((\ln\alpha - 0.58)^2 - \frac{(k-1)(1-\alpha)}{(1+\varepsilon)(2(1+\alpha)\ln k + (\frac{k}{B}))} \cdot \ln n) \leq 0.$$

This implies that $\ln\gamma = -\ln\alpha + 0.58 - \sqrt{\frac{(k-1)(1-\alpha)}{(1+\varepsilon)(2(1+\alpha)\ln k + (\frac{k}{B}))} \cdot \ln n}$. By setting $\alpha = \frac{1}{2}$, $\gamma \leq \frac{3.58}{e^{\sqrt{\frac{(k-1)\ln n}{2(1+\varepsilon)(3\ln k + (k/B))}}}}$.

Next, we consider different values of k and show that in each case what is the minimum amount of space in which Learned CM outperforms CM by a factor of 1.06 (setting $\varepsilon = 0.06$).

- $\boldsymbol{k = 1}$. In this case, we are basically comparing the expected error of a single hash function and Learned Count-Min. In particular, in order to get a gap of at least $(1+\varepsilon)$, by a more careful analysis of Lemma 9.2, γ must satisfy the following condition:
$$\mathbf{E}[\mathrm{Err}_{L-CM}] \leq \frac{(\ln(\frac{1}{\alpha\gamma}) + 0.58)^2}{(1-\alpha)B} \leq \frac{1}{1+\varepsilon} \cdot \frac{\ln^2 n - 1.65}{(1+\alpha)B} \leq \frac{1}{1+\varepsilon} \cdot \mathbf{E}[\mathrm{Err}_h],$$
To simplify it further, we require that $(\ln(\frac{2}{\gamma}) + 0.58)^2 \leq 3.18 \cdot (\ln^2 n - 1.65)$ which implies that $\gamma = \Theta(1/\ln n)$.

- $\boldsymbol{k = 2}$. In this case, $\gamma \leq \frac{3.58}{e^{\sqrt{(\ln n)/4.5}}}$ for sufficiently large values of B. Hence, we require that the total amount of available space is at least $\frac{5.37n}{e^{\sqrt{(\ln n)/4.5}}}$.

- $\boldsymbol{k \in \{3, 4\}}$: In this case, $\gamma \leq \frac{2}{e^{\sqrt{(\ln n)/3.5}}}$ for sufficiently large values of B. Hence, we require that the total amount of available space is at least $\frac{5.37n}{e^{\sqrt{(\ln n)/3.5}}}$.

- $\boldsymbol{k \geq 5}$. In this case, $\gamma \leq \frac{2}{e^{\sqrt{(\ln n)/2.6}}}$ for sufficiently large values of B. Hence, we require that the total amount of available space is at least $\frac{5.37n}{e^{\sqrt{(\ln n)/2.6}}}$.

We also note that settings where the number of buckets is close to n are quite common in practice. For example, Goyal et al. (2012) uses $((1+\alpha)B = 20M, n = 33.5M)$ and $((1+\alpha)B = 50M, n = 94M)$ with $k = 3$, while Kumar et al. (2004) uses different pairs of $((1+\alpha)B, n)$ including $(B = 288K, n = 563K)$. Plugging these values into the exact formulas computed above shows that Learned CM has a significantly lower loss than CM.

10. Optimal Hash Function

In this section we compute an asymptotic bound error bound of optimal hash functions that map items $[n]$ to a set of buckets $C[1 \cdots B]$.

Claim 10.1. *For a set of items I, let $f(I)$ denote the total frequency of all items in I; $f(I) := \sum_{i \in I} f_i$. In any optimal hash function of form $\{h : I \to [B_I]\}$ with minimum estimation error, any item with frequency at least $\frac{f(I)}{B_I}$ does not collide with any other items in I.*

Proof: For each bucket $b \in [B_I]$, lets $f_h(b)$ denotes the frequency of items mapped to b under h; $f(b) := \sum_{i \in I: h(i)=b} f_i$. Recall that the estimation error of a hash function h is defined as $\mathrm{Err}(\mathcal{F}(I), \tilde{\mathcal{F}}_h(I)) := \sum_{i \in I} f_i \cdot (f(h(i)) - i)$. Note that we can rewrite $\mathrm{Err}(\mathcal{F}(I), \tilde{\mathcal{F}}_h(I))$ as
$$\begin{aligned}\mathrm{Err}(\mathcal{F}(I), \tilde{\mathcal{F}}_h(I)) = \sum_{i \in I} f_i \cdot (f(h(i)) - f_i) &= \sum_{i \in I} f_i \cdot f(h(i)) - \sum_{i \in I} f_i^2 \\ &= \sum_{b \in [B_I]} f(b)^2 - \sum_{i \in I} f_i^2. \qquad (10.1)\end{aligned}$$
Note that in (10.1) the second term is independent of h and is a constant. Hence, an optimal hash function minimizes the first term, $\sum_{b \in B_I} f(b)^2$.

Suppose that an item i^* with frequency at least $\frac{f(I)}{B_I}$ collides with a (non-empty) set of items $I^* \subseteq I \setminus \{i^*\}$ under an optimal hash function h^*. Since the total frequency of the items mapped to the bucket b^* containing i^* is greater than $\frac{f(I)}{B_I}$ (i.e., $f(h(i^*)) > \frac{f(I)}{B_I}$), there exists a bucket $\overline{b}$ such that $f(\overline{b}) < \frac{f(I)}{B_I}$. Next, we define a new hash function $\overline{h}$ with smaller estimation error compared to h^* which contradicts the optimality of h^*:
$$\overline{h}(i) = \begin{cases} h^*(i) & \text{if } i \in I \setminus I^* \\ \overline{b} & \text{otherwise.} \end{cases}$$

Formally,

$$\begin{aligned}
\mathrm{Err}(\mathcal{F}(I), \tilde{\mathcal{F}}_{h^*}(I)) - \mathrm{Err}(\mathcal{F}(I), \tilde{\mathcal{F}}_{\overline{h}}(I)) &= f_{h^*}(b^*)^2 + f_{h^*}(\bar{b})^2 - f_{\overline{h}}(b^*)^2 - f_{\overline{h}}(\bar{b})^2 \\
&= (f_{i^*} + f(I^*))^2 + f_{h^*}(\bar{b})^2 - f_{i^*}^2 - (f_{h^*}(\bar{b}) + f(I^*))^2 \\
&= 2f_{i^*} \cdot f(I^*) - 2f_{h^*}(\bar{b}) \cdot f(I^*) \\
&= 2f(I^*) \cdot (f_{i^*} - f_{h^*}(\bar{b})) \\
&> 0 \quad \triangleright \text{ Since } f_{i^*} > \frac{f(I)}{B} > f_{h^*}(\bar{b}).
\end{aligned}$$

□

Next, we show that in any optimal hash function $h^* : [n] \to [B]$ and assuming Zipfian input distribution, $\Theta(B)$ most frequent items do not collide with any other items under h^*.

Lemma 10.2. *Suppose that $B = n/\gamma$ where $\gamma \geq e^{4.2}$ is a constant and lets assume that items follow Zipfian distribution. In any hash function $h^* : [n] \to [B]$ with minimum estimation error, none of the $\frac{B}{2\ln\gamma}$ most frequent items collide with any other items (i.e., they are mapped to a singleton bucket).*

Proof: Let i_{j^*} be the most frequent item that is not mapped to a singleton bucket under h^*. If $j^* > \frac{B}{2\ln\gamma}$ then the statement holds. Suppose it is not the case and $j^* \leq \frac{B}{2\ln\gamma}$. Let I denote the set of items with frequency at most $f_{j^*} = 1/j^*$ (i.e., $I = \{i_j \mid j \geq j^*\}$) and let B_I denote the number of buckets that the items with index at least j^* mapped to; $B_I = B - j^* + 1$. Also note that by Observation 9.1, $f(I) < \ln(\frac{n}{j^*}) + 1$. Next, by Claim 10.1, we show that h^* does not hash the items $\{j^*, \cdots, n\}$ to B_I optimally. In particular, we show that the frequency of item j^* is more than $\frac{f(I)}{B_I}$. To prove this, first we observe that the function $g(j) := j \cdot (\ln(n/j) + 1)$ is strictly increasing in $[1, n]$. Hence, for any $j^* \leq \frac{B}{2\ln\gamma}$,

$$\begin{aligned}
j^* \cdot (\ln(\frac{n}{j^*}) + 1) &\leq \frac{B}{2\ln\gamma} \cdot (\ln(2\gamma\ln\gamma) + 1) \\
&\leq B \cdot (1 - \frac{1}{2\ln\gamma}) \quad \triangleright \text{ Since } \ln(2\ln\gamma) + 2 < \ln\gamma \text{ for } \ln\gamma \geq 4.2 \\
&< B_I
\end{aligned}$$

Thus, $f_{j^*} = \frac{1}{j^*} > \frac{\ln(n/j^*)+1}{B_I} > \frac{f(I)}{B_I}$ which implies that an optimal hash function must map j^* to a singleton bucket. □

Theorem 10.3. *If $n/B \geq e^{4.2}$, then the estimation error of any hash function that maps a set of n items following Zipfian distribution to B buckets is $\Omega(\frac{\ln^2(\frac{n}{B})}{B})$.*

Proof: By Lemma 10.2, in any hash function with minimum estimation error, the $(\frac{B}{2\ln\gamma})$ most frequent items do not collide with any other items (i.e., they are mapped into a singleton bucket) where $\gamma = n/B > e^{4.2}$.

Hence, the goal is to minimize (10.1) for the set of items I which consist of all items other than the $(\frac{B}{2\ln\gamma})$ most frequent items. Since the sum of squares of m items that summed to S is at least S^2/m, the multi-set loss of any optimal hash function is at least:

$$\begin{aligned}
\mathrm{Err}(\mathcal{F}(I), \tilde{\mathcal{F}}_{h^*}(I)) &= \sum_{b \in [B]} f(b)^2 - \sum_{i \in [n]} f_i^2 && \triangleright \text{ By (10.1)} \\
&\geq \frac{(\sum_{i \in I} f_i)^2}{B(1 - \frac{1}{2\ln\gamma})} - \sum_{i \in I} f_i^2 \\
&\geq \frac{(\ln(2\gamma\ln\gamma) - 1)^2}{B} - \frac{2\ln\gamma}{B} + \frac{1}{n} && \triangleright \text{ By Observation 9.1} \\
&= \Omega(\frac{\ln^2\gamma}{B}) && \triangleright \text{ Since } \gamma > e^4 \\
&= \Omega(\frac{\ln^2(n/B)}{B}).
\end{aligned}$$

□

Next, we show a more general statement which basically shows that the estimation error of any Count-Min sketch with B buckets is $\Omega(\frac{\ln^2(n/B)}{B})$ no matter how many rows (i.e., the value of k) it has.

Theorem 10.4. *If $n/B \geq e^{4.2}$, then the estimation error of any Count-Min sketch that maps a set of n items following Zipfian distribution to B buckets is $\Omega(\frac{\ln^2(\frac{n}{B})}{B})$.*

Proof: We prove the statement by showing a reduction that given a Count-Min sketch $CM(k)$ with hash functions $h_1, \cdots, h_k \in \{h : [n] \to [B/k]\}$ constructs a single hash function $h^* : [n] \to [B]$ whose estimation error is less than or equal to the estimation error of $CM(k)$.

For each item i, we define $C'[i]$ to be the bucket whose value is returned by $CM(k)$ as the estimate of f_i; $C'[i] := \arg\min_{j \in [k]} C[j, h_j(i)]$. Since the total number of buckets in $CM(k)$ is B, $|\{C'[i] \mid i \in [n]\}| \leq B$; in other words, we only consider the subset of buckets that $CM(k)$ uses to report the estimates of $\{f_i \mid i \in [n]\}$ which trivially has size at most B. We define h^* as follows:

$$h^*(i) = (j^*, h_{j^*}(i)) \quad \triangleright \text{ for each } i \in [n], \text{ where } j^* = \arg\min_{j \in [k]} C[j, h_j(i)]$$

Unlike $CM(k)$, h^* maps each item to exactly one bucket in $\{C[\ell, j] \mid \ell \in [k], j \in [B/k]\}$; hence, for each item i, $C'[h^*(i)] \leq C[h^*(i)] = \tilde{f}_i$ where $\tilde{f}_i$ is the estimate of f_i returned by $CM(k)$. Moreover, since for each i, $C'[h^*(i)] \geq f_i$,

$$\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{h^*}) \leq \mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_{CM(k)}).$$

Finally, by Theorem 10.3, the estimation error of h^* is $\Omega(\frac{\ln^2(n/B)}{B})$ which implies that the estimation error of $CM(k)$ is $\Omega(\frac{\ln^2(n/B)}{B})$ as well. □