Putting the "Learning" into Learning-Augmented Algorithms for Frequency Estimation

Elbert Du [1 2 *] **Franklyn Wang** [1 2 *] **Michael Mitzenmacher** [2]

Abstract

In learning-augmented algorithms, algorithms are enhanced using information from a machine learning algorithm. In turn, this suggests that we should tailor our machine-learning approach for the target algorithm. We here consider this synergy in the context of the learned count-min sketch from (Hsu et al., 2019). Learning here is used to predict heavy hitters from a data stream, which are counted explicitly outside the sketch. We show that an approximately sufficient statistic for the performance of the underlying count-min sketch is given by the *coverage* of the predictor, or the normalized L^1 norm of keys that are filtered by the predictor to be explicitly counted. We show that machine learning models which are trained to optimize for coverage lead to large improvements in performance over prior approaches according to the average absolute frequency error. Our source code can be found at https://github.com/franklynwang/putting-the-learning-in-LAA.

1. Introduction

The nascent research area of algorithms with predictions, also known as learning-augmented algorithms (see (Mitzenmacher & Vassilvitskii, 2020) for a survey), focuses on algorithms that can take advantage of predictions. In this paper, we consider sketch algorithms designed to estimate frequencies of keys in a data stream, where the predictor is a machine-learning algorithm used to predict "heavy-hitters," which are the most frequent keys in the stream.

Abstractly, we view finding a learning-augmented algorithm as an optimization problem over the predictor parameterized by θ and an algorithm A

$$\max_{\theta \in \Theta, A \in \mathcal{A}} \mathcal{M}(\theta, A)$$

where $\mathcal{M}$ is some suitable performance metric. That is, we seek to find the best combination of algorithm and predictor. Most work so far in learning-augmented algorithms has focused on choosing the best parametrization of the algorithm, taking the predictor as a given. Ideally, learning-augmented algorithms should provide feedback to designers of predictors, by providing guidance as to what makes for better predictions in the given context. The goal should be to devise a method of optimizing θ for A, and not just A for θ. To do this, we create a simple function $g(\theta) \approx \mathcal{M}(\theta, A)$ (which we term an *approximately sufficient statistic*) for a fixed A, and directly optimize $g(\theta)$. Here, we apply this framework to the learned frequency estimation problem of (Hsu et al., 2019), but we believe this approach is quite general and can be applied to many learning augmented algorithms.

In (Hsu et al., 2019), the authors perform frequency estimation by using a sketch, which is a data structure designed to give *approximate* frequency counts for keys in a data stream with small amounts of space, using much less than one counter per key. However, to improve traditional sketch performance, they take keys that are predicted to be the most frequent and give them their own counter, and then apply a sketch to the remaining keys. Removing heavy keys greatly improves the sketch performance and thereby overall performance.

We show that in the case of the count-min sketch, the *coverage*, or the L^1 norm of such screened keys, is a particularly good predictor of its evaluation results. By using this metric, we are able to evaluate our learned models more quickly, as well as train against the coverage as a target to obtain better performance. Given this, we then consider better ways of parametrizing the sketch to maximize performance.

Summarized, our contributions are as follows:

- We formalize theoretically and show experimentally that coverage aligns well with performance.
- We show, for suitable Zipfian distributions of keys and

*Equal contribution [1]Department of Mathematics, Harvard University [2]Department of Computer Science, Harvard University. Correspondence to: Franklyn Wang <franklyn_wang@college.havard.edu>, Elbert Du <edu@college.havard.edu>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

predictor accuracy, when optimizing the parameters of the count-min sketch, it is best to use one hash function, in contrast to most typical settings.

- We devise a differentiable batch objective that aligns better with coverage than the objective used in (Hsu et al., 2019), showing that this leads to consistent improvements with *no other changes to architecture or training*.
- Inspired by the pattern of errors of our predictor, we propose a model where more frequent keys are more often filtered by the predictor even above the threshold, in contrast to the uniform model of errors in (Hsu et al., 2019). Error bounds proven with this model more closely approximate real performance. Our bounds capture the error of Zipfian input for all parameters at greater than 0, improving bounds from (Aamand et al., 2019) for the uniform model of errors.

All proofs can be found in Appendix A.

1.1. Related Work

Differentiable Algorithms Another way of combining algorithms and machine learning is through differentiable analogues of algorithms, which allows the systems to be optimized end-to-end (see (Xie & Ermon, 2019) for an example). However, many natural algorithms are not readily expressible this way, which limits this method's utility.

Learned Frequency Estimation The work of learned frequency estimation was begun by (Hsu et al., 2019), which pioneered the idea of using a machine learning model for the classical task of frequency estimation with small space. They show that by using a learned model to remove very frequent keys and store those keys in a hash table, one can greatly improve performance while using the same amount of space.

Learning Augmented Algorithms In addition to frequency estimation, learning has been used to improve algorithms for a large number of problems, including computing frequency moments (Jiang et al., 2020), scheduling (Mitzenmacher, 2020), caching (Rohatgi, 2020; Wei, 2020), support estimation (Eden et al., 2021), queuing (Lykouris & Vassilvitskii, 2018), and membership (Dai & Shrivastava, 2020; Vaidya et al., 2021).

The paper (Wu et al., 2019) targets finding a static matrix A that yields a strong recovery guarantee, so that the vector x (if sparse) can be recovered from Ax. One way to view the present paper through this lens is that, we are learning a matrix $A(x)$ that is a good recovery matrix for $y = f(x)$, where x represents the features and y represents the frequencies.

Loss modelling In other applications where total differentiability is not possible and a machine learning model lies at the beginning of the pipeline, other works have also considered the idea of using loss functions to better capture downstream tasks. Particularly relevant is the work of (Guo et al., 2020) which uses an anisotropic quantization loss instead of a standard isotropic loss function for similarity search, where this quantization is made to be more accurate precisely when the key is particularly likely to be similar to the query.

2. Preliminaries

For notational convenience, we assume our key set is $[n] = \{1, 2, \dots n\}$. Throughout the paper, we let $\boldsymbol{f} = (f_1, f_2, \dots f_n)$ refer to the true frequency vector of all the keys. Generally $\boldsymbol{f}'$ refers to the frequency vector after removing keys screened by a learned predictor (which are given separate counters), so $\boldsymbol{f}'$ agrees with $\boldsymbol{f}$ at all nonzero entries, but may be zero at indices where $\boldsymbol{f}$ is nonzero. We let F^α be a random variable supported on $[n]$ so that $P(F^\alpha = i) \propto (f_i)^\alpha$. This represents a distribution that samples each key according to some power of its frequency.

Count-Min Sketch The Count-Min Sketch is a data structure used to estimate the frequencies of keys in a data stream (Cormode & Muthukrishnan, 2005a). The sketch is an $\ell \times w$ array CM, where we refer to ℓ as the number of *layers*, or the height of the array, and w as the *width* of the array. We have ℓ independent hash functions

$$h_1, h_2, \dots h_\ell : [n] \to [w]$$

where n is the number of keys, and we let the keys be integers in $[n]$. We initially have this array set to all 0s.

Then, each time we get as input a key-value pair (i, v), with $v > 0$, we increment the counter of $CM[j, h_j[i]]$ by v for each $j \in [\ell]$. Finally, to estimate the frequency of a key i (meaning the sum of the values associated with key i), we take the minimum of $CM[j, h_j[i]]$ over all $j \in [\ell]$. (One can also allow negative increments under natural conditions; we consider positive values for convenience.) This method has low approximation error, and formalizations of its error bound can be found in, e.g. (Cormode & Muthukrishnan, 2005a;b).

Count Sketch The Count Sketch (Cormode & Hadjieleftheriou, 2008) is a data structure very similar to the Count-Min Sketch. We have an $\ell \times w$ array CS and 2ℓ independent hash functions

$$h_1, h_2, \dots h_\ell : [n] \to [w]$$

and

$$s_1, s_2, \dots s_\ell : [n] \to \pm 1$$

where w is the "width" of the sketch, n is the number of keys, and we let the keys be integers in $[n]$. We initially have this array set to all 0s, and each time we get as input a key-value pair (i, v), we add $s_j[i] \cdot v$ to $CS[j, h_j[i]]$ for each $j \in [\ell]$. To estimate the frequency of a key i, we take the median of $CS[j, h_j[i]]$ over all $j \in [\ell]$, where if ℓ is even we take the average of the $(\ell)/2$th and $(\ell/2)+1$th largest numbers to find the median.

Learning-Based Frequency Estimation Notation We reproduce the algorithm of (Hsu et al., 2019) below in Algorithm 1. Recall their algorithm gives keys that the predictor thinks are heavy hitters an explicit counter (if one is available), and places the other keys into the count-min sketch. We also borrow their notation, so B is the total space used, B_r is the number of explicit counters (which exactly store the counts of the heavy hitters), and HH refers to a heavy hitter predictor, which predicts whether or not a given key is a heavy hitter.

Algorithm 1 Learned Sketch

```
input: $B$, $B_r$, Predictor HH
Initialize $B_r$ explicit counters
Initialize an empty Sketch S with space $B - B_r$
for each key-value pair $(i, v)$ do
  if HH$(i) = 1$ and explicit counter is available. then
    if i is already in a explicit counter then
      increment the count of the explicit counter containing $i$ by $v$
    else
      Initialize the count of an empty explicit counter to $v$ and indicate that it contains $i$
    end if
  else
    give $i$ as input to S
  end if
end for
```

When the heavy hitter detector identifies a stream element i as a heavy hitter, we say that the element was *screened*. Otherwise, we say that the element was *accepted*. We use *explicit counter* to refer to counters for screened keys, and *cell* to refer to a counter in a count-min sketch.

We define the *coverage* of f'_i with respect to f_i as

$$1 - \frac{\sum_i f'_i}{\sum_i f_i},$$

representing the relative mass of the keys that are screened by the predictor from $\boldsymbol{f}$ to $\boldsymbol{f'}$.

Following (Hsu et al., 2019), for a Zipf distribution with parameter p, the i^{th} most frequent element has frequency proportional to i^{-p}.

3. A Nearly Sufficient Statistic for Count-Min sketch performance

Intuitively, it is clear that given a set of keys, if we can remove some keys from the stream (those we classify as heavy hitters), we would get better performance from the count-min sketch if we removed the keys with the highest frequency, and further if we had a choice of two keys to remove it would be better to remove the one with higher frequency. (This is easily proven, for example by a simple coupling argument). However, the idea that we need to "choose the largest frequencies" is insufficient for training a predictor. Instead, we need to have an objective function we can optimize, and thus we aim to show that coverage is the correct objective function for the Count-Min Sketch.

The error metric that is typically used for frequency estimation (and which we use here) is the *average absolute frequency error* (Hsu et al., 2019),

$$\sum_i f_i \cdot |\tilde{f}_i - f_i| \propto \mathbb{E}_{i \sim F^1}[|\tilde{f}_i - f_i|] \qquad (1)$$

where $\tilde{f}_i$ is the predicted value by the algorithm, and $\boldsymbol{f}$ can be any vector.

The error of one layer of a count-min sketch on key i can be written as

$$R_{\boldsymbol{f},i} = \sum_{j \neq i} f_j X_j.$$

where $\boldsymbol{f}$ is a vector and $X_1, X_2, \ldots X_j$ are independent Bernoulli random variables that are each 1 with probability $1/w$. We also define for convenience

$$R_{\boldsymbol{f}} = \sum_j f_j X_j.$$

The overall error of the sketch on key i is the minimum of ℓ independent copies of $R_{\boldsymbol{f},i}$, which we denote by $R^{\ell}_{\boldsymbol{f},i}$, and similarly we let $R^{\ell}_{\boldsymbol{f}}$ be the minimum of ℓ independent copies of $R_{\boldsymbol{f}}$.

In the particular case of $\boldsymbol{f'}$, we can now write the expected error of the learned sketch as

$$\mathbb{E}[\text{err}(\boldsymbol{f'})] = \sum_i f'_i \mathbb{E}\left[R^{\ell}_{\boldsymbol{f'},i}\right].$$

The efficacy of the count-min sketch can be understood from this perspective. Note that $R_{\boldsymbol{f'},\cdot}$ is a weighted sum of Bernoulli variables that are usually zero, so when the input is skewed, the expected value is largely dominated by rare cases where a highly-weighted Bernoulli is equal to 1 (a phenomenon we call *Bernoulli skew*). Using the count-min sketch with $\ell > 1$ alleviates this skew, because Bernoulli skew must happen multiple times for the same key to give a high error for that key. As a result, many times the optimal value for ℓ, given a fixed amount of space, is greater than 1.

However, when we introduce screening using our predictor, we discover that the Bernoulli skew mostly vanishes (since the more frequent keys are screened out), and often $\ell = 1$ is ideal. Further, it may also be the case that the distribution of keys is unskewed to begin with (say, as a Gaussian) in which case $\ell = 1$ is likely to be ideal with or without the presence of screening.

3.1. Performance with one and many layers

In the case where there is one layer in the sketch, analyzing the error turns out to be straightforward; we show that the expected error has expected value proportional to the squared L^1 norm, provided that the squared L^1 norm is much larger than the squared L^2 norm.

Theorem 3.1. *If $\|\boldsymbol{f'}\|_2 < \alpha \|\boldsymbol{f'}\|_1$, the expected error of the count-min sketch (with one row) lies between $(1-\alpha^2)\|\boldsymbol{f'}\|_1^2/m$ and $\|\boldsymbol{f'}\|_1^2/m$.*

In practical settings, α is generally very small, so this is essentially equal to $(\|\boldsymbol{f'}\|_1^2/m)$, which is proportional to $(1 - \text{coverage})^2$. Thus, for a single row, optimizing the coverage yields an excellent approximation to optimizing the estimation error.

For multiple rows, we use the following approximation:

$$\mathbb{E}[\text{err}(\boldsymbol{f'})] = \sum_i f'_i \mathbb{E}[R^{\ell}_{\boldsymbol{f'},i}] \approx \sum_i f'_i \mathbb{E}[R^{\ell}_{\boldsymbol{f'}}]$$
$$= \|\boldsymbol{f'}\|_1 \mathbb{E}[R^{\ell}_{\boldsymbol{f'}}] = \underbrace{\left(\frac{\sum_i f'_i}{\sum_i f_i}\right)}_{1-\text{coverage}} \underbrace{\left(\sum_i f_i\right)}_{\text{constant}} \underbrace{\mathbb{E}[R^{\ell}_{\boldsymbol{f'}}]}_{\text{expected residual}} . \tag{2}$$

Note the approximation involves replacing $R^{\ell}_{\boldsymbol{f'},i}$ by $R^{\ell}_{\boldsymbol{f'}}$, potentially (but with fairly low probability) including an item's frequency in its error.

Equation 2 shows that the performance is a combination of two factors, namely the coverage and the expected residual. This again argues for optimizing coverage in the learning process. First, optimizing coverage does also aim to minimize the residual (albeit indirectly). Second, we show that the expected residuals between “similar” frequency sequences, for a suitable definition of similar, are themselves nearly equal when there is more than one hash function. In particular, learning procedures that are aiming generally to find heavy hitters will yield similar sets of keys that were not screened, and thus close expected residuals. Both points suggest optimizing for coverage is a suitable proxy for overall error.

In order to precisely state our result, we define an appropriate notion of similarity. For two decreasing sequences $A = (a_1 \geq a_2 \geq \ldots \geq a_n)$ and $B = (b_1 \geq b_2 \geq \ldots \geq b_m)$, we say the *domination number* d of A with respect to B is the smallest number so that $a_i \geq b_{i+d}$ for all $i \leq m - d$. That is, if we remove the largest d frequencies from B, then the a sequence pointwise dominates the b sequence. We consider sequences A and B to be similar when they have low domination numbers with respect to each other.

Theorem 3.2. *Let A and B have domination number at most d with respect to each other, and let $p = (1 - 1/w)^d$.*

$$\mathbb{E}[R^{\ell}_A] - \mathbb{E}[R^{\ell}_B] \leq \left(\sum_{i=1}^{\ell-1}\left(\binom{\ell}{i}p^i (1-p)^{\ell-i}\big(\mathbb{E}[R^i_B] - \mathbb{E}[R^{\ell}_B]\big)\right) + (1-p)^{\ell} \left(\mathbb{E}[R^1_B] + \frac{1/w}{1-(1-1/w)^{|Q|}} S(|Q|) - \mathbb{E}[R^{\ell}_B]\right)\right) \tag{3}$$

where Q is the multiset of keys that should be removed from A in order for B to pointwise dominate A and $S(|Q|)$ is the sum of the elements of Q.

Since when d is small, $(1 - p)$ is small as well, we see that $\mathbb{E}[R^{\ell}_A]$ and $\mathbb{E}[R^{\ell}_B]$ are similar for small d. Furthermore, for even slightly large ℓ, $(1 - p)^{\ell}$ will tend to be small, making the second term small as well.

As the expected residuals are very close for similar sequences, most of the performance difference in the case of multiple hash functions is determined by the coverage.

3.2. Optimizing the number of sketches

Now, we explore the optimal number of sketches, to see if the case of 1 hash function or many hash functions has better performance. Prevailing wisdom states that the optimal number of hash functions to use in a count-min sketch is usually some constant greater than 1. Yet we find that when the most frequent keys are all screened from Zipfian distributed input, having one layer is often optimal.[1] This happens because removing the most frequent keys heavily reduces the Bernoulli skew, so multiple layers are largely unnecessary, as the variable $R_{\boldsymbol{f},i}$ is concentrated around its mean. Intuitively, we are better off using the space by putting more cells in a single row (reducing collisions overall) rather than splitting cells into multiple rows (giving multiple chances to avoid collisions with high-frequency items).

Theorem 3.3. *For each $p, c > 0$, given Zipfian input with parameter p, and the largest $n' = cn$ frequencies removed (say, by an oracle), there exists a constant $c' > 0$ such that a Count-Min Sketch with total space at most $c'n$ has larger expected error with k rows than 1 row for all $k \geq 2$.*

[1]This is a strengthening of a special case of the result found in (Aamand et al., 2019), where they proved that the optimal number of rows was $O(1)$.

3.3. Synthetic Experiments

We verify the theoretical results with synthetic experiments. In what follows, there are $n = 10^6$ keys, and the frequency of the kth key is $\lceil n/k \rceil$. We consider three different scenarios:

- No Screening: No keys are screened.
- Perfect Screening: The top 1% most common keys are screened.
- Imperfect Screening: a uniformly randomly chosen 80% of the top 1.25% most common keys are screened.

We use 20,000 total cells in the count-min sketch in each experiment, so the memory is the same even if we vary the number of rows. Tables 1, 2, 3 give the average residual, average absolute frequency error, and root mean squared error respectively, where we define root mean squared error to be

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\tilde{f}_i - f_i)^2}.$$

The results shown are averaged over 1000 trials. We see that, as suggested in Theorem 3.2, the average residuals with two or more rows are very close with imperfect and perfect screening. Also, as Theorem 3.3 suggests, one row yields better performance when using screening, but not in the absence of screening. Interestingly the root mean squared error is not aligned with the average absolute frequency error, as we can see one row is ideal in the imperfect screening case for average absolute frequency error but far from ideal in the case of root mean squared error. However, this is not unexpected as the root mean squared error is dominated by collisions between any key and a key with high frequency.

# Rows	None	Imperfect	Perfect
1	592.16	294.48	209.36
2	529.23	373.20	364.22
3	731.38	538.45	530.13
4	952.03	711.55	701.01

Table 1. Average Residual.

# Rows	None	Imperfect	Perfect
1	592.16	129.55	62.75
2	529.23	164.18	109.17
3	731.38	236.88	158.90
4	952.03	313.03	210.12

Table 2. Absolute Average Frequency Error.

# Rows	None	Imperfect	Perfect
1	9098.19	3936.37	219.29
2	686.32	391.19	369.31
3	769.70	543.00	533.03
4	983.48	713.76	702.36

Table 3. Average Root Mean Squared Error.

4. Choosing the Loss Function

Recall that in our notation, F^0 represents a distribution that gives equal weight for each key, whereas F^1 represents a distribution that weights keys proportional to their frequency.

There are many loss functions we could use to optimize the predictor. We denote the predictor by $g_\theta(i)$, where θ represents the parameters of a neural network. Note that we have not specified what $g_\theta(i)$ is approximating. The reason for this is that choosing what $g_\theta(i)$ should approximate is a crucial part of our design space.

(Hsu et al., 2019) chooses to minimize the loss function given by

$$\mathbb{E}_{i\sim F^0}[(g_\theta(i) - \ln f_i)^2],$$

so the neural network predicts the log frequency of the keys.[2] We call this loss function the *unweighted log loss*. Since the coverage can be expressed as

$$\mathbb{E}_{i\sim F^1}[1_{i \text{ in } H}],$$

where H is the set of heavy-hitters, a potential alternative loss function is

$$\mathbb{E}_{i\sim F^1}[(g_\theta(i) - \ln f_i)^2].$$

We call this the *weighted log loss*, because it accounts for the weighting.

Another approach might be creating a predictor that directly optimizes the absolute average frequency error (Equation 1)

$$\mathbb{E}_{i\sim F^1}[|e^{g_\theta(i)} - f_i|].$$

For completeness we also consider its unweighted variant

$$\mathbb{E}_{i\sim F^0}[|e^{g_\theta(i)} - f_i|].$$

We call the first of these the *weighted L^1 loss* and the second the *unweighted L^1 loss*.

Finally, we select a loss function based on our observation that the coverage is *almost sufficient* for the performance. Note that we can write the coverage as

$$\mathbb{E}_{i\sim F^0}[1_{i \text{ in } H} f_i].$$

[2]The logarithm is used because neural networks tend to have poor performance when approximating large numbers.

A seemingly compelling method is to have the function $g_\theta(i)$ predict $1_{i \text{ in } H}$ *directly*, instead of predicting the log frequency as it does with the above loss functions. Intuitively, this approach is helpful because our end goal is select the heaviest hitters to remove rather than accurately predict their frequency. This leads us to consider the loss function $\mathbb{E}_{i \sim f^0}[g_\theta(i) f_i]$.

However, optimizing using this loss function without any global restrictions on $g_\theta(i)$ will simply encourage $g_\theta(i)$ to take on the maximum possible value. We are not aware of any existing approach to enforce our desired constraint that $g_\theta(i)$ selects a given size subset. We thus suggest the following approach, which we call BatchRank, to approximate this goal. To our knowledge, this approach has not appeared previously.

We divide the dataset into randomly selected batches of size B for each epoch, and the model updates on each batch. For each batch of B keys, we split it into sub-batches of size K randomly (with each key in exactly one sub-batch). Then within the sub-batches we calculate a normalized value

$$g'_\theta(i) = \frac{g_\theta(i) - \mu}{\sigma}$$

where μ and σ are the mean and standard deviation of g_θ across the sub-batch. This idea is partially inspired by BatchNorm (Ioffe & Szegedy, 2015), although in our case it serves to normalize outputs on batches as opposed to normalizing the hidden layers.

Then our final loss function is

$$\mathbb{E}_{i \sim F^0}[g'_\theta(i) f_i].$$

Intuitively, creating sub-batches performs better than simply normalizing within each batch, because when we normalize over larger sets of keys, our function g_θ degenerates into choosing the maximum within each batch and thus loses valuable signal in differentiating between less frequent keys.

5. Experiments

5.1. Experimental Details

Datasets and Architectures Following (Hsu et al., 2019), we use the CAIDA dataset. We use the first 7 minutes of the link for training, the 8th minute for validation, and the 9th, 30th, and 60th minutes for testing. The 9th minute is meant to test how well the algorithms have learned the temporally local distribution; the assumption is the 9th minute is relatively similar to the first six. The 30th and 60th minutes are used to see whether the algorithms have learned general patterns of heavy flows. We use the 9th and 60th minutes to demonstrate our coverage results and the 30th and 60th minutes to show the full results of the count-min sketch.

We only include the table and graph for minute 60 in this paper. The results for minutes 9 and 30 can be found in Appendix C, but the results are fairly similar to those of minute 60.

We use the same architecture as in (Hsu et al., 2019), an LSTM (Gers et al., 2000), and run our algorithm on the CAIDA dataset. We use 1 NVIDIA V100 for each run. Each training run takes around 12 to 18 hours, and trains for 200 epochs with batch size equal to 1024. For the optimizer, we use Adam with learning rate 10^{-3}. When optimizing over the distribution $i \sim F^0$, we sample batches uniformly from F_0, and when optimizing over the distribution $i \sim F^1$, we sample the keys proportionally when creating batches.

Sketches We explain how we perform our evaluation given the predictions. In practice, when a key arrives we are given its prediction, and we need to decide whether it should have its own explicit counter. Therefore, we must select a threshold and give the key an explicit counter if its predicted value is above the threshold and there is a counter available; we assume a fixed-size set of explicit counters. We select the threshold by looking at validation data and choosing a corresponding percentile threshold in the validation data. For example, if we decide that we would like to have explicit counters equal to 10% of the number of keys, we can set a cutoff equal to the 90th percentile of the predictor on the validation data. However, if we overestimate the threshold, we run the risk of having explicit counters go unused, wasting valuable space. On the other hand, if we underestimate the threshold, while some frequent keys may not obtain an explicit counter, the most common keys are still likely to obtain an explicit counter as they are more likely to appear earlier in the stream. (We assume that the distribution of key arrival order is uniform.) Thus, we deliberately underestimate the cutoff to be the number which would include $1.1k$ of the validation keys if we have k explicit counters.

5.2. Results

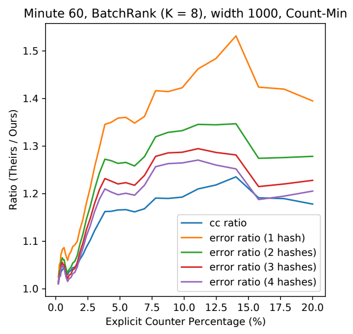
We first show the coverage of each of our loss functions (with one run each) in Table 4. The coverage size at $i\%$ is the total frequency, or coverage, of $i\%$ of the keys with the highest value of f_θ, the estimated frequency. The ideal row represents the coverage of the $i\%$ of keys with the highest true frequency. Unweighted Log Loss is the original loss function used in (Hsu et al., 2019), and all of the others are the new methods we proposed. BatchRank and Weighted Log Loss consistently outperform the unweighted log loss and both L^1 losses.

Next, we examine whether our superior coverage results translate into improvements in frequency estimation error when using the count-min sketch.

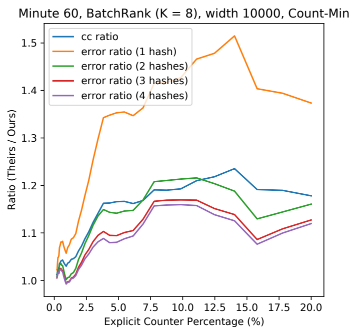
In Figure 1, we plot the complementary coverage (i.e.

Table 4. Coverage (%) on CAIDA dataset, 60th minute. The largest entry in each column is in **bold**.

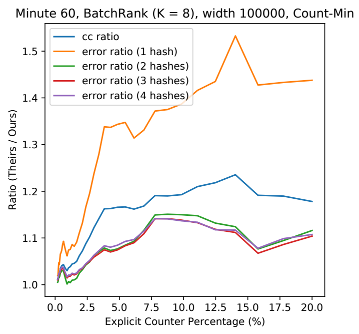
Method	Coverage Size							
	1%	2%	5%	10%	20%	30%	50%	75%
Unweighted Log Loss	31.1%	37.1%	45.2%	55.7%	66.8%	78.6%	88.4%	96.7%
Weighted Log Loss	28.3%	38.2%	51.7%	61.6%	**75.1%**	**82.2%**	**91.1%**	**97.8%**
Unweighted L^1 Loss	18.1%	21.1%	27.6%	35.5%	46.4%	59.1%	75.7%	88.0%
Weighted L^1 Loss	16.2%	23.3%	35.1%	48.0%	62.3%	70.7%	82.7%	92.6%
BatchRank, κ = 64	**34.6%**	**41.1%**	50.6%	61.5%	71.5%	77.9%	87.5%	96.1%
BatchRank, κ = 8	33.1%	40.7%	**52.4%**	**62.8%**	74.2%	80.9%	89.7%	96.6%
Ideal	62.3%	69.6%	78.5%	84.9%	90.5%	93.8%	97.3%	99.1%



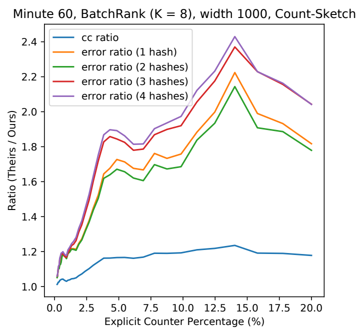
(a)



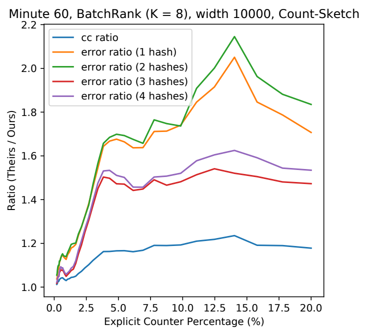
(b)



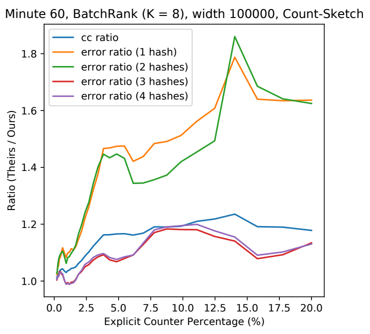
(c)



(d)



(e)



(f)

Figure 1. Plots of Error and Complementary Coverage (CC) ratios between learned count-min sketches and learned count-sketches using the loss function from (Hsu et al., 2019) and BatchRank ($K = 8$) where the sketch's width is fixed. We only consider explicit counter percentages of 20% or less, because in practical scenarios where sketching is useful we are unlikely to be able to store 20% of the counters explicitly.

$1 -$ coverage) and the absolute average frequency errors of a model against the baseline (unweighted log loss) for cases with small (width 1,000), medium (width 10,000), and large (width 100,000) sketches with varying number of hash functions for both count-min sketches and count-sketches. We ran 100 trials for the case of one hash function, and 20 trials for the cases of two or more hash functions, as the case of one hash function has a particularly high variance, and took the average. To visualize the results, we simply plot the ratios between the average absolute frequency errors and the ratios between the complementary coverages. We show only BatchRank with $K = 8$, but weighted log loss and BatchRank with $K = 64$ are similar.

When we have only 1 hash function, our results for count-min sketches follow our predicted behavior (Theorem 3.1), and the error ratio is almost exactly the square of the complementary coverage ratio.

Otherwise, the error ratios for the count-min sketch are still above 1, but below the complementary coverage ratios. Recall that error is complementary coverage times expected residual, so the expected residual is higher for BatchRank than unweighted log loss. In each of these cases, however, our model's higher coverage enables it to obtain lower errors since the differences in expected residual is not large, as predicted by Theorem 3.2. Overall, our methods outperform unweighted log loss in all cases, as none of the error ratios ever drop below 1. Furthermore, our improvements are significant, with the method of (Hsu et al., 2019) resulting in an average error of 121% of our method over all experiments with a count-min sketch, with the ratio reaching over 160% for the count-sketch.

Figure 1 shows similar results for the count-sketch, in that optimizing learning for coverage improves performance, although the issue of the correct number of hash functions is more complicated.

5.3. Discussion

As we can see from Table 4, BatchRank tends to do quite well, especially when predicting the top $i\%$ for small i, despite not formally estimating any frequencies. We see that K should be tuned so that with relatively small amounts of space it should be larger and with relatively large amounts of space it should be smaller.

Furthermore, prediction L^1 coverages are consistently much worse than BatchRank, showing that often making a predictor that does well by itself (as the L^1 loss is exactly aligned to the prediction task) does not suffice for the predictor to perform well within a learning-augmented algorithm. This provides some demonstration of our high-level idea that optimizing for coverage can lead to better results; that is, optimizing with the whole algorithm in mind rather than optimizing for the full downstream task (in this case frequency estimation) can yield substantial improvements.

6. A Refined Model of Errors

We now consider modelling the predictor's relationship to the correct answer. Models for prediction errors in learning-augmented algorithms are often quite simple. For example, (Hsu et al., 2019) considers a model where a key is given an explicit counter with probability $1-\delta$ if the key is a heavy hitter, and is placed in the count-min sketch otherwise.

However, a more natural model would take into account that the higher the frequency of a heavy-hitter key, the more likely it is to be screened and placed into an explicit counter. (Similar concepts appear in the literature for learned Bloom filters; see e.g. (Vaidya et al., 2021).) This model is empirically justified; Figure 2 shows the screening rate of the model that uses BatchRank with $K = 8$ and a 10% screening threshold. We see that, for the most frequent keys, the probability of being screened increases with frequency. For completeness, similar results for other predictors are given in Appendix D.

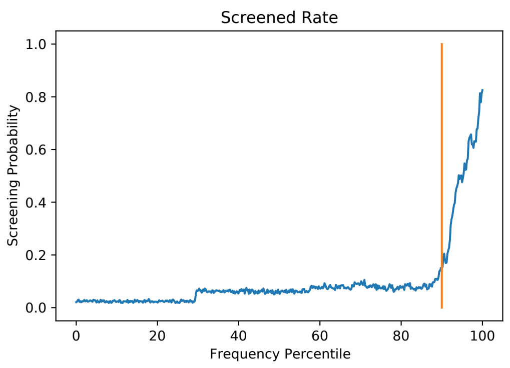


Figure 2. Screened Rate of BatchRank with $K = 8$ on Minute 60. The x-axis denotes the percentile of frequency, and the y-axis represents the probability that it was screened (with a 10% threshold). Note that the screening rate is essentially monotonically increasing, and sharply so above the threshold.

This heterogeneity, whereby more frequent keys are more likely to be screened, potentially allows for tighter theoretical analyses. Here we show we can obtain a slightly sharper result than (Hsu et al., 2019) by using this improved model.

To obtain our theoretical results, we assume that the probability of a key being screened is polynomially related to its frequency. That is, there is some constant $c > 0$ such that the probability that a key i with frequency f_i is not screened is proportional to $\frac{1}{f_i^c}$ until it reaches the maximum value of 1. This model is sufficiently general that one can intuitively think of it as including all distributions where the probability of being screened increases monotonically with f_i as further increasing the skew only improves the performance of the sketch, and it matches the behavior in Figure 2.

For this section, as is done in (Hsu et al., 2019), we additionally assume for the sake of simplicity that if the input is Zipfian with parameter p, the frequency of the i^{th} most frequent key is i^{-p}. In the following theorem, one can think of the exponent c as controlling the error rate of the oracle, which approaches uniformly screening random keys as $c \to 0$ and approaches perfection as $c \to \infty$. The seemingly strange choice of normalizing constant is justified in Appendix A.5. Essentially, its purpose is to ensure that B_r slots are filled by predicted heavy hitters.

Theorem 6.1. *For any constant $c > 0$, if the input is Zipfian with parameter 1 and the heavy hitter oracle screens key i with probability $p(f_i)$ where for $1 \le i \le (1 + 1/c)B_r$,*

$$p(f_i) = 1 - \left(\frac{B_r/c}{\sum_{j=1}^{B_r(1+1/c)} j^c} \right) i^c$$

and $p(f_i) = 0$ otherwise, then the error for the Learned

Count-Min Sketch is $O\left(\frac{1/c^2+\ln^2(n/B_r)}{B-B_r}\right)$.

Remark. If we keep this same probability distribution and have an arbitrary parameter $p > 0$ so that $p \neq c+1$ for the Zipf distribution, we can get the more general bound

$$O\left(\frac{B_r^{2-2p}/(c-p+1)^2}{B-B_r} + \frac{n^{2-2p}/(1-p)^2}{B-B_r}\right) \quad (4)$$

which we prove in Appendix A.6. When $p = c+1$, we instead get the bound

$$O\left(\frac{\ln^2(B_r)+1/c^2}{B_r^{2c}(B-B_r)}\right).$$

This result may appear to be strange, but this is because as p varies, so does $\|\mathbf{f}\|_1$ or the total frequency of the input. When we take this into account and normalize the error bounds, we get bounds which line up more nicely with what one would expect. Detailed analysis of the normalized versions of these bounds are given in Appendix A.7.

To compare our results with the bounds in (Hsu et al., 2019) we state the following corollary.

Corollary 6.1.1. *The error of the Learned Count-Min Sketch under the oracle model from 6.1, with* $B_r = c'B$ *for some* $0 < c' < 1$*, and with Zipfian input with parameter* 1*, is* $\Theta\left(\frac{\ln^2(n/B)}{B}\right)$.

We note that the error bound of Corollary 6.1.1 asymptotically matches the error lower bound proved in Theorem 10.4 of (Hsu et al., 2019) with a perfect heavy hitters oracle for the Zipfian input distribution with parameter 1.

Also, the bound of Corollary 6.1.1, when $B_r = c'B$ for some $0 < c' < 1$, improves on the bound

$$O\left(\frac{\delta^2\ln^2(B_r)+\ln^2(n/B_r)}{B-B_r}\right)$$

from Theorem 9.15 of (Hsu et al., 2019). Here in their model, the oracle misclassifies heavy hitter keys with a fixed probability δ.

This agrees with what we would expect as when the predictor is incorrect about keys far from the cutoff, the situation is much worse than when the predictor is incorrect about keys close to the cutoff both intuitively and when looking at the coverage and error.

7. Conclusion and Future Work

We have examined the frequency estimation problem, one of the first problems studied under the paradigm of learning-augmented algorithms, and have shown that by tailoring the learning task to the algorithm we achieve reasonable and consistent improvements in performance. We view this as an important example that motivates co-design between the algorithm and predictor for learning-augmenting algorithms. Our consideration also led us to a new theoretical analysis based on a more realistic model of predictor performance, and we again think that improved results may similarly be available for other problems when the predictor is more accurately modelled.

For future work, we note the BatchRank approach used is somewhat ad-hoc. BatchRank resembles BatchNorm, which has proven successful in many deep learning application. While even BatchNorm is still not well understood, we believe that batch normalization here helps force the learning to determine whether items are included or not. We believe that more theoretical analysis of batch normalization would be useful.

8. Acknowledgements

We would like to thank Alexander Wei for helpful discussions. We would like to thank the MIT IBM-Watson AI lab for graciously providing us computational resources on the Satori cluster, and John Cohn for technical support, without which this project would not have been possible. We would like to thank Chen-Yu Hsu for providing us with preprocessed versions of the datasets, which saved us large amounts of time. We would also like to thank anonymous referees for helpful feedback. Michael Mitzenmacher was supported in part by NSF grants NSF CCF-1563710, CCF-1535795 and DMS-2023528, and by a gift to the Center for Research on Computation and Society at Harvard University.

References

Aamand, A., Indyk, P., and Vakilian, A. (learned) frequency estimation algorithms under zipfian distribution. *CoRR*, abs/1908.05198, 2019.

Cormode, G. and Hadjieleftheriou, M. Finding frequent items in data streams. *Proc. VLDB Endow.*, 1(2): 1530–1541, 2008. ISSN 2150-8097. doi: 10.14778/1454159.1454225.

Cormode, G. and Muthukrishnan, S. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005a. ISSN 0196-6774.

Cormode, G. and Muthukrishnan, S. Summarizing and mining skewed data streams. In *SIAM International Conference on Data Mining*, pp. 44–55. SIAM, 2005b.

Dai, Z. and Shrivastava, A. Adaptive learned bloom filter (ada-bf): Efficient utilization of the classifier with application to real-time information filtering on the web. In *NeurIPS*, 2020.

Eden, T., Indyk, P., Narayanan, S., Rubinfeld, R., Silwal, S., and Wagner, T. Learning-based support estimation in sublinear time. In *ICLR*, 2021. URL `https://openreview.net/forum?id=tilovEHA3YS`.

Gers, F. A., Schmidhuber, J., and Cummins, F. A. Learning to forget: Continual prediction with LSTM. *Neural Comput.*, 12(10):2451–2471, 2000.

Guo, R., Sun, P., Lindgren, E., Geng, Q., Simcha, D., Chern, F., and Kumar, S. Accelerating large-scale inference with anisotropic vector quantization. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3887–3896. PMLR, 2020.

Hsu, C., Indyk, P., Katabi, D., and Vakilian, A. Learning-based frequency estimation algorithms. In *ICLR*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=r1lohoCqY7`.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 448–456. JMLR.org, 2015.

Jiang, T., Li, Y., Lin, H., Ruan, Y., and Woodruff, D. P. Learning-augmented data stream algorithms. In *ICLR*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=HyxJ1xBYDH`.

Lykouris, T. and Vassilvitskii, S. Competitive caching with machine learned advice. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3302–3311. PMLR, 2018.

Mitzenmacher, M. Scheduling with Predictions and the Price of Misprediction. In *ITCS*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 14:1–14:18, 2020. ISBN 978-3-95977-134-4. doi: 10.4230/LIPIcs.ITCS.2020.14.

Mitzenmacher, M. and Vassilvitskii, S. Algorithms with predictions. *arXiv preprint arXiv:2006.09123*, 2020.

Rohatgi, D. Near-optimal bounds for online caching with machine learned advice. In *SODA*, pp. 1834–1845. SIAM, 2020. doi: 10.1137/1.9781611975994.112.

Vaidya, K., Knorr, E., Mitzenmacher, M., and Kraska, T. Partitioned learned bloom filters. In *ICLR*, 2021. URL `https://openreview.net/forum?id=6BRLOfrMhW`.

Wei, A. Better and simpler learning-augmented online caching. In *APPROX/RANDOM*, volume 176, pp. 60:1–60:17, 2020.

Wu, S., Dimakis, A., Sanghavi, S., Yu, F. X., Holtmann-Rice, D. N., Storcheus, D., Rostamizadeh, A., and Kumar, S. Learning a compressed sensing measurement matrix via gradient unrolling. In *ICML 2019*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6828–6839. PMLR, 2019.

Xie, S. M. and Ermon, S. Reparameterizable subset sampling via continuous relaxations. In *IJCAI*, pp. 3919–3925, 2019. doi: 10.24963/ijcai.2019/544.