Improving the Bloom Filter with Support Size Estimation

Team SketchEst: Jennifer Brennan, Qiuyu Chen, Fatemeh Ghezloo

August 8, 2021

1 Problem Statement

A Bloom filter [Blo70] is a memory-efficient data structure designed to support set membership queries. A Bloom filter consists of h hash functions that hash into b buckets each, and an $h \times b$ bit array. When an item is inserted into the filter, it is hashed h times, and the corresponding h buckets in the bit array are set to 1. When an item is queried, the filter asserts that the item is in the set if all h of its associated bits are set to 1. Every item in the set is correctly identified by the filter, but the filter also declares some false positives. The false positive rate of a Bloom filter is approximately

$$\delta \approx \left(1 - e^{hk/b}\right)^h$$

where k is the number of distinct items to be inserted. If we want to design a Bloom filter with a given false positive rate δ, we need to know k. In many applications, the number of distinct items is unknown a priori. Our goal in this project is to estimate k from a sample of the items in the set, and use this estimate to design an appropriate Bloom filter.

2 Our Approach

We propose optimizing the hyperparmeters of the Bloom filter using an estimate of the support size of the data generating distribution, obtained from a small sample of the data stream. We first sample a small set of n elements from the distribution, then apply support size estimation on this sample to estimate the number of distinct items in the full dataset. We then use this estimate to choose filter parameters h and b that achieve a desired false positive rate using the minimum table size. This is more efficient than a classical Bloom filter because we estimate the size of distinct items based on samples rather than guessing the expected number of distinct items.

3 Related Work

In recent years, several variants of the basic Bloom filter have been proposed in the literature and they are being used in a large number of systems. However, they all suffer from the limitation of knowing the size of the filters. Considering that it's not always possible to know in advance how many elements will need to be stored, this leads to over-dimensioning the filters. To drop this limitation, [Alm+07] proposes the idea of the scalable Bloom filters. A Scalable Bloom Filter addresses the problem of having to choose an a priori maximum size for the set, and allows an

arbitrary growth of the set being represented, while assuring a maximum false positive probability. The standard filter starts with one filter with k_0 slices and error probability P_0. When this first filter is full, another one is added with k_1 slices and $P_1 = P_0 r$ error probability, where r is the tightening ratio with $0 < r < 1$. This mechanism adapts to set growth by using a series of standard Bloom Filters of increasing sizes and tighter error probabilities, added as needed.

The standard Bloom filters don't take dynamic sets into account which are a great part of real world applications. Thus, dynamic Bloom filters [Guo+09] were presented to deal with this problem. Dynamic Bloom filters not only have the advantage of Bloom filters, but also have better features than Bloom filters when dealing with dynamic datasets. The false match probability of Bloom filters increases exponentially with the increase of the cardinality of a dynamic set, while that of dynamic Bloom filters increases slowly because it expands capacity in an incremental manner according to the set cardinality. Through mathematical analysis, they show that dynamic Bloom filters use less expected memory than Bloom filters when dealing with dynamic sets with upper bounds on set cardinality, and that dynamic Bloom filters are more stable than Bloom filters due to infrequent reconstruction when addressing dynamic sets without upper bounds on set cardinality.

Also, there are split Bloom filters [XDL04] which in order to represent a set, they allocate a fixed $s \times m$ bit matrix instead of an m-bit vector to scale the size of the filter. It employs a certain number of s filters, each having m bits and then uniformly selects them when inserting an item of the set. The false positive rate increases as the set cardinality grows. When an existing split Bloom filter exceeds an upper bound threshold, it must be reconstructed using a new bit matrix. Although dynamic Bloom filters and split Bloom filters share a similar structure, they are different in the amount of memory they consume. Split Bloom filters always consume $s \times m$ bits and waste too much memory before the set cardinality reaches $(m \times ln2)/k$, whereas dynamic Bloom filters allocate memory in an incremental manner.

All these papers, propose variants of bloom filters with variable size with respect to the size of the dataset which could be dynamic. On the contrary, our approach proposes a fixed-sized bloom filter with estimating the number of distinct items in the dataset.

There is also a rich literature on support size estimation, or estimating the number of distinct items in a distribution. An early example is the Good-Turing frequency estimator [Goo53], which estimates the probability of seeing an as-yet unobserved domain element in a given window of time. Recently, Valiant and Valiant [VV13] proposed an estimator for symmetric properties of discrete distributions - a category that includes support size - that requires only $O(n/\log n)$ iid observations from a distribution with support size n. This is remarkable because it is sublinear in the support size - a reasonable estimate of the support size can be acquired before we could possibly see every item.

4 Theoretical Results

We have explored the use of two possible support size estimators, and examined the kinds of guarantees we could get on the suboptimality of our returned Bloom filter.

4.1 The "Estimating the Unseen" estimator

Suppose we used Valiant & Valiant's "Estimating the Unseen" estimator for support size [VV13], and then used this estimate to design a Bloom filter with a given false positive rate. Then, ignoring

integrality considerations, the optimal Bloom filter parameters h (number of hash functions) and b (bits per hash function) are as follows:

$$h = \log_2\left(\frac{1}{\delta}\right)$$
$$b = 1.44k\log_2\left(\frac{1}{\delta}\right)$$

where δ is the nominal false positive rate, and k is the support size of the item distribution (i.e., k is the number of distinct items we will insert). Note that only the number of bits b is dependent on the support size, and this dependence is linear. Since the total memory footprint of the filter is proportional to $h \cdot b$, we see that the memory is also linear in k.

If we use Valiant & Valiant's estimator, we can apply their theoretical results to bound the sub-optimality of our filter parameters. Specifically, their Corollary 12 describes the sample complexity necessary to estimate the support size k to multiplicative error ε. This, in turn, lets us bound the sample complexity necessary to choose our Bloom filter parameters, and have memory within a multiplicative error ε of the optimal choice. We have the following result.

Lemma 4.1. *There exist absolute positive constants α, γ such that for any $\varepsilon \in (0,1)$, there exists k_ε such that for any $k > k_\varepsilon$, if we take at least*

$$n \geq \frac{\gamma}{\varepsilon^2}\frac{k}{\log k}$$

iid samples from some distribution p having at most k distinct elements, and use these n elements to estimate the support size of the distribution, then with probability at least $1 - e^{-k^\alpha}$ our resulting Bloom filter will take memory within $1.44k\varepsilon\log_2\left(\frac{1}{\delta}\right)^2$ of the Bloom filter we would have chosen had we known k exactly, provided that no probability in p lies in $(0, \frac{1}{k})$.

This lemma suggests that we could find a Bloom filter with memory requirement within a factor of $(1+\varepsilon)$ of the memory we would use if we had knowledge of the true support size k, after only taking $n \propto \frac{1}{\varepsilon^2}\frac{k}{\log k}$ samples to generate this initial estimate.

This sort of lemma is useful for theoretical analysis, but unfortunately the sample complexity guarantee depends on unknown quantities like k, the true support size. This prevents us from reporting the corresponding confidence intervals on the support size, which would be necessary if we wanted to guarantee that our Bloom filter's false positive rate was controlled at the nominal level.

4.2 The Good-Turing Estimator

Good & Turing [Goo53] developed an estimator for the "missing mass" in a probability distribution. Specifically, they answered the question, "If I were to draw one additional sample from this distribution, what is the probability that it was previously unseen?" Their estimate is given by

$$\hat{p} = \frac{\text{number of items seen exactly once}}{\text{total number of items drawn}}$$

The idea behind this estimator is that our probability of seeing a novel item in the next draw is roughly the same as our probability of seeing a novel item in all previous draws. This motivates

the use of the empirical probability of seeing a novel item in a previous draw, $\hat{p}$, to estimate the missing mass. Note that a missing mass estimator is very different from a support size estimator. For example, suppose the true missing mass is 10%; i.e., there is a 10% chance that the next item is not in your current list of observed items. However, that mass could be distributed among elements in very different ways. Perhaps there is one unseen element with probability 0.1. At the other extreme, maybe there are a thousand unseen elements, each with probability 0.1/1000. In order to use the missing mass to bound the support size, we need to know something about the probabilities of elements in the distribution. In particular, if we could lower bound the probability of every element, then we could get an approximate upper bound on the support size.

Conveniently for us, we often have access to the total number of (potentially non-distinct) elements we will insert into the Bloom filter; e.g. by using the file size of the file we are scanning to determine the number of data points it contains. If the file contains K items to insert, then we know each item appears with probability at least $1/K$, and the number of missing items is upper bounded by

$$\text{number of missing items} \leq (\text{probability of seeing a missing item}) \cdot K$$

Using the Good-Turing missing mass estimate, we have the estimated upper bound

$$\text{number of missing items} \lessapprox \frac{\text{number of items seen exactly once}}{\text{total number of items drawn}} \cdot K$$

This will be the second estimator we implement for this project.

Finally, we note that the simplicity of the Good-Turing estimator makes it an ideal candidate for finite-sample bounds on the support size. Consider that the expected value of the Good-Turing estimator in fact upper bounds our quantity of interest,

$$\begin{aligned}\mathbb{P}(\text{item } i \text{ was previously unseen}) &\leq \mathbb{P}(\text{item } i-1 \text{ was previously unseen}) \\ &= \mathbb{E}[\hat{p}]\end{aligned}$$

Furthermore, $\hat{p}$ is a binomial random variable, so by Hoeffding's inequality it is straightforward to write an upper bound that holds with probability at least $1-\alpha$,

$$\mathbb{P}(\text{item } i \text{ was previously unseen}) \leq \hat{p} + \sqrt{\frac{\log(1/\alpha)}{2n}}$$

where n is the number of observations we have taken to estimate the support size. We have already shown how the missing mass can be used to upper bound the support size, so we have the following bound on the number of unseen elements:

$$\text{number of missing items} \leq K\left(\hat{p} + \sqrt{\frac{\log(1/\delta)}{2n}}\right)$$

We end with a few observations on the bound above. First, if we use this bound as the support size k in the Bloom filter, then with high probability our designed filter will not exceed the nominal false positive rate. This comes at the cost of routinely overestimating the space requirement, but this trade may be acceptable in some applications. Second, we discuss the form of this bound for an extreme example of the data distribution. When the true support size k is very small compared

to the number of items K, we expect $\widehat{p}$ to behave like $1/n$. In this case, the upper bound will be dominated by the second term, which behaves like $1/\sqrt{n}$. Then the bound behaves like $K/\sqrt{n}$ is always at least $\sqrt{K}$. This tells us that our upper bound will never return fewer than order $\sqrt{K}$ items, even if k is in fact a small constant. It would be interesting to apply a Bernstein bound to the estimate $\widehat{p}$, which uses information about the variance of the random variable, to see if that improved this behavior.

5 Experiments

5.1 Dataset

We evaluate our method on two datasets. For hamlet dataset, we extracted 30,780 words from Hamlet script, which contains 5,065 unique words. We also extracted names of institutions from DBLP dataset, which contains 959292 institutions and 4711 items are unique.

5.2 Results on unshuffled data

In our first experiment, we investigate the correlation between number of samples and the false positive rate. We use the Good Turing estimator and the unseen estimator to estimate their corresponding support size, and compare the false positive rates from these two estimators. Figure 1 Left shows that as N is increasing, the false positive rate is not necessarily converging to the nominal false positive rate, and unseen estimator is very unstable when the number of samples is smaller than 5000. Both estimators try to converge when n is between 5000 and 10000, but then fail to find good solutions when n is larger than 10000.

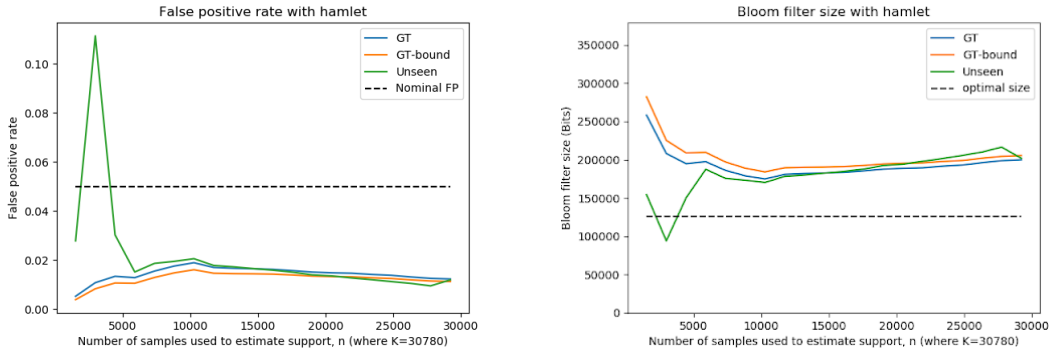

Figure 1: Left: False positive rate on hamlet dataset. Right: Memory (bloom filter size) on hamlet dataset

We also analyze the correlation between bloom filter size and number of samples n. We observe the performance is quite consistent the first experiment: when n is around 10000, the bloom filter achieves optimal size. Unseen estimator seems better than Good Turing estimator when n is small, but fails to converge to find the optimal solution when n gets larger.

However, we observe a much better performance on DBLP dataset with both estimators. In Figure2 Left, we can see both estimators converge to nominal false positive rate when n is larger

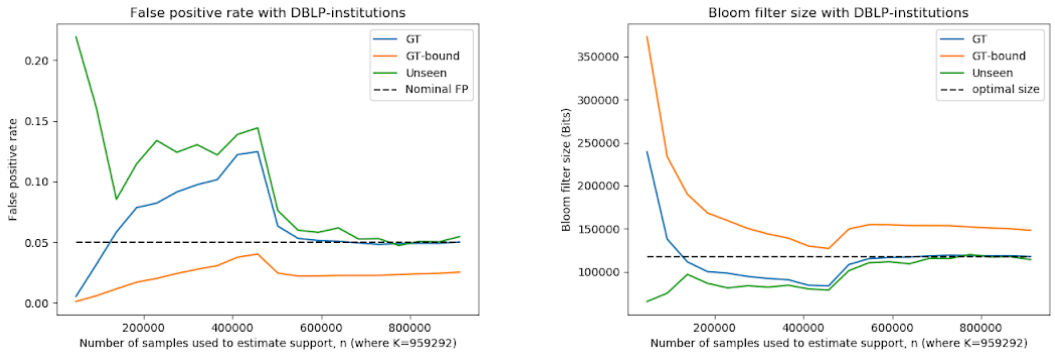

Figure 2: Left: False positive rate on DBLP dataset. Right: Memory (bloom filter size) on DBLP dataset

than 500000. In particular, good turning estimator seems to perform better than unseen estimator. However, both estimators require fairly large amount of data.

In addition, we want to see if we have a fixed amount of memory, how would these two estimators perform if we increase K number of items (file size) allocated in our memory. Figure3 shows that the good turning estimator performs much better than the unseen estimator as K increases: it could converge to the optimal solution when K increases.

The unseen estimator performs much better on DBLP dataset compared to Hamlet dataset: As K is increasing, the unseen estimator could converge to a good solution much quicker than the good turning estimator, but it has much more variance when K is less than 100000.

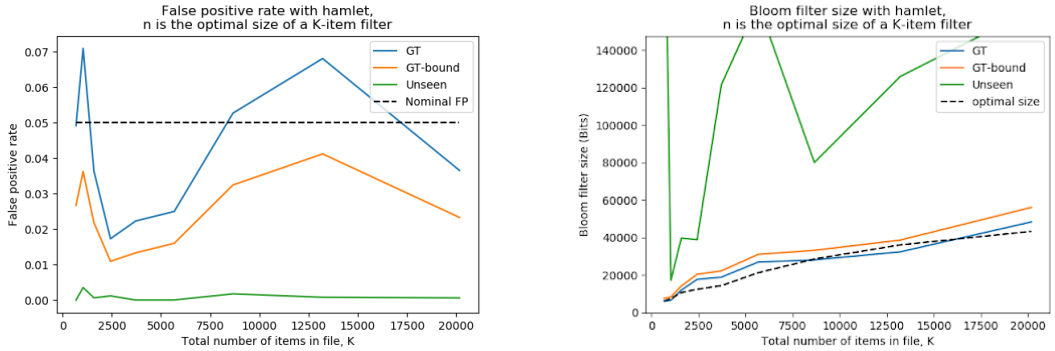

Figure 3: Left: False positive rate on Hamlet dataset as the total number of items K increases. Right: Memory (bloom filter size) on Hamlet dataset as total number of items K increases

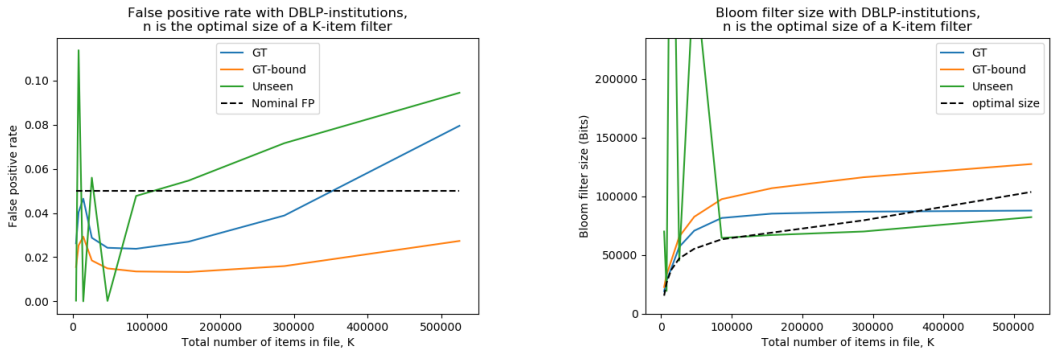

Figure 4: Left: False positive rate on DBLP dataset as total number of items increasing. Right: Memory (bloom filter size) on DBLP dataset as total number of items increasing

5.3 Results on shuffled data

Previous results were based on our estimators assuming that the first n samples drawn from data distribution are iid. To check whether this assumption is reasonable, we plotted the cumulative fraction of number of observed distinct items vs total number of items seen so far. Also, we plotted the 90 percent confidence interval. If the data was iid, the graphs in Figure5 would fall within the shaded region but they clearly don't.

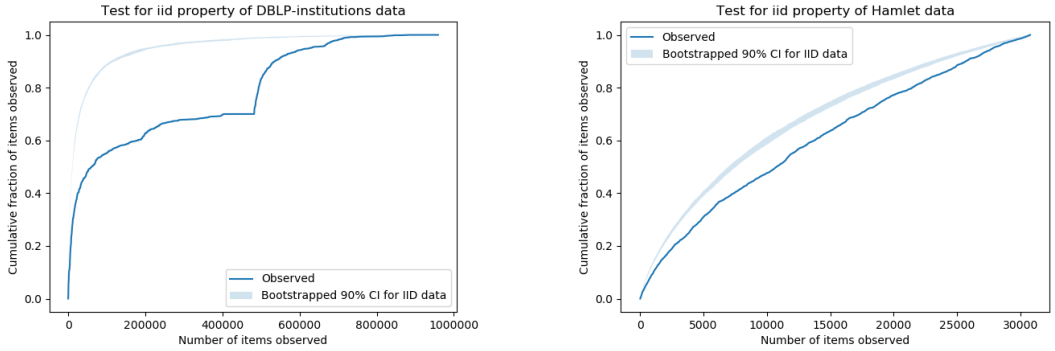

Figure 5: Left: Test for iid property of DBLP dataset . Right: Test for iid property of Hamlet dataset

We asked how the experimental results of would change if we simulated drawing data from these real distributions, but in an iid manner. We shuffled the data in a random order and repeated previous experiments for forty times to measure the average performance of the estimators. Figure6, shows the performance of our estimators on the shuffled Hamlet dataset, as we increase the fraction of the file read. On left, we have the results for the bloom filter false positive rate and we can see that the Unseen estimator is more variable and has slightly less bias. Although, they all appear to converge to a biased solution.

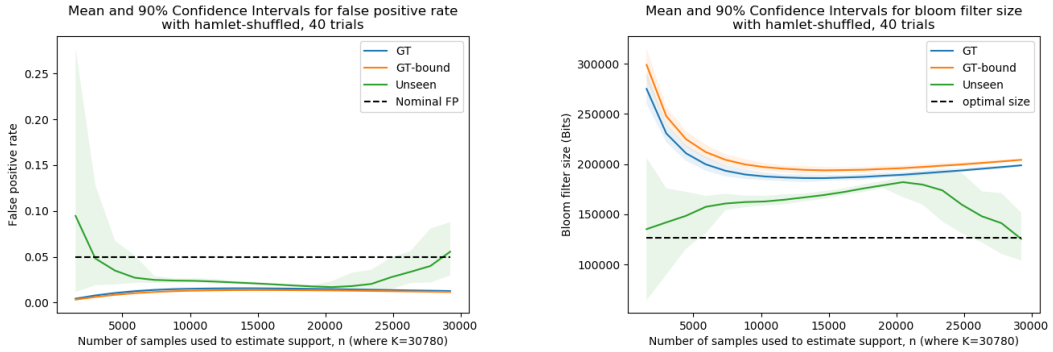

Figure 6: Left: False positive rate on shuffled Hamlet dataset. Right: Memory (bloom filter size) on shuffled Hamlet dataset

Figure7 shows the results of same experiment for the shuffled DBLP data. The Unseen estimator has high variance but it clearly is better than the rest as it converges quickly.

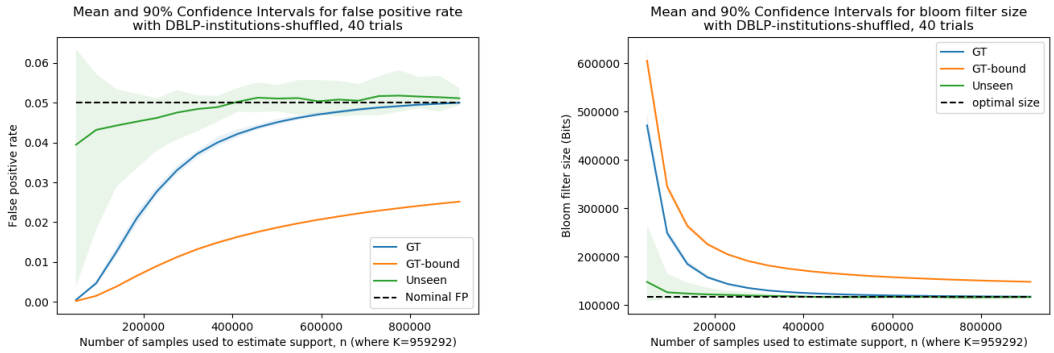

Figure 7: Left: False positive rate on shuffled DBLP dataset. Right: Memory (bloom filter size) on shuffled DBLP dataset

Following results are for the second experiment where we allocate the memory for a K element filter and use that as our initial sample pool. Figure8 shows the results for shuffled Hamlet data where all the estimators fail to converge to a good solution. But, Unseen estimator is worse that the other two. Figure9 represents the results for the same experiment but the shuffled DBLP data. The unseen estimator is more variable with better average performance and the best convergence.

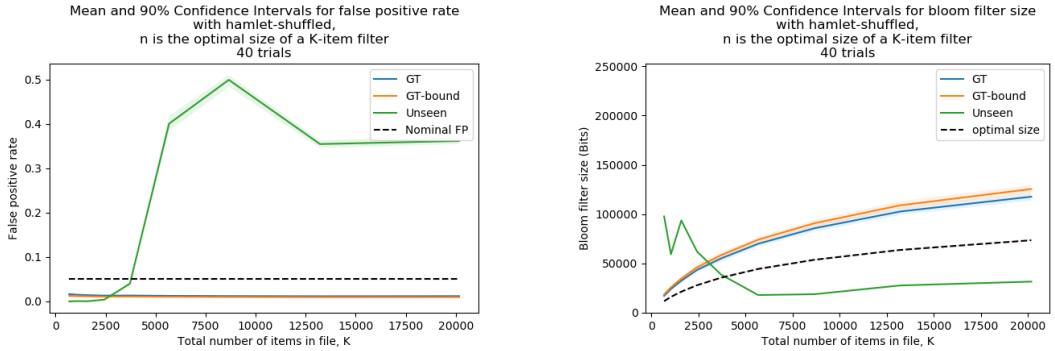

Figure 8: Left: False positive rate on shuffled Hamlet dataset. Right: Memory (bloom filter size) on shuffled Hamlet dataset

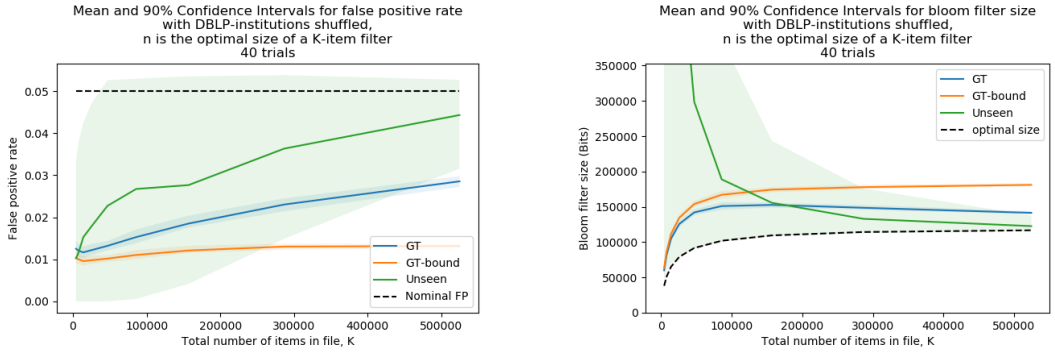

Figure 9: Left: False positive rate on shuffled DBLP institutions dataset. Right: Memory (bloom filter size) on shuffled DBLP institutions dataset

6 Conclusion

We have described a way to improve the design of Bloom filters by estimating the support size of a streaming data set. We have described three estimators, one of which we can apply off-the-shelf (the Unseen estimator), and two of which are modifications of existing estimators (the Good Turing and Good Turing Bound estimators are modifications of the Good-Turing missing-mass estimator). The main takeaway from this work is that **these estimators rely heavily on the iid assumption, and do not perform well when this assumption is violated.**

When the iid assumption is valid (which, for us, are represented in the "shuffled" plots), the two files we tested give insight into the behavior of the two estimators.

The Hamlet data comes from a long-tailed distribution, where any sample contains a large fraction of novel elements. In hindsight, it is not surprising that our estimators consistently overestimate the support size of Hamlet, because the true support size (which we can think of as Shakespeare's vocabulary), was undoubtedly larger than just the words appearing in the play. If the stream length

is known ahead of time, this suggests that none of the estimators will provide good estimates on such a long-tailed distribution. However, if the stream length is potentially unbounded, then the unseen estimator (which does not need to know the total length K) could be a good choice.

The DBLP data, by contrast, comes from a much shorter-tailed distribution; few institutions make up a majority of the dataset. We see on the shuffled data that the Unseen estimator converges most quickly to the true support size. In addition, in the experiments where we chose n based on the optimal size of a K-element filter, only the Unseen estimator is able to accurately estimate the support size using that value of n. This suggests that **if the dataset is suspected to be light-tailed, and the data stream is very large, then the Unseen estimator can be used to estimate the support size, using no more memory than would be required for the worst-case filter.**

Future work for this project could proceed in many directions. It would be interesting to couple this technique with a change point detector, which could alert us to distribution shifts like the one in the DBLP data set, and might allow us to modify our support size estimator. This would require detecting change points using very little memory. Toward a similar end, it would be interesting to understand whether we could estimate support size with a non-iid sample; certainly we could not estimate with a completely adversarial sample, but we might ask if there are relaxed assumptions on the data generating process that admit nontrivial estimators. Finally, it would be of interest to compare to additional baselines, such as the other techniques in the related work.

7 Distribution of tasks

Jennifer reviewed papers about the theory behind the unseen estimator and the Good Turing estimator, and coded the Good Turing estimator. Fatemeh and Jennifer started working and design some experiments. Zoey helped with the experiments in the Unseen estimator, including the attempt to import Matlab to python. Fatemeh helped to review the related work, and designed the poster and slides. She also helped to generate data from hamlet script. Jennifer extracted institution names from DBLP dataset. We had weekly meetings to discuss about the project. All of us participated in writing the reports of this project.

References

[Goo53] Irving J Good. "The population frequencies of species and the estimation of population parameters". In: *Biometrika* 40.3-4 (1953), pp. 237–264.

[Blo70] Burton H Bloom. "Space/time trade-offs in hash coding with allowable errors". In: *Communications of the ACM* 13.7 (1970), pp. 422–426.

[XDL04] M.-Z Xiao, Y.-F Dai, and X.-M Li. "Split bloom filter". In: *Tien Tzu Hsueh Pao/Acta Electronica Sinica* 32 (Feb. 2004), pp. 241–245.

[Alm+07] Paulo Sérgio Almeida et al. "Scalable bloom filters". In: *Information Processing Letters* 101.6 (2007), pp. 255–261.

[Guo+09] Deke Guo et al. "The dynamic bloom filters". In: *IEEE Transactions on Knowledge and Data Engineering* 22.1 (2009), pp. 120–133.

[VV13] Paul Valiant and Gregory Valiant. "Estimating the unseen: improved estimators for entropy and other properties". In: *Advances in Neural Information Processing Systems*. 2013, pp. 2157–2165.