CS166 Final Paper: The L_p-norm Sketch

Daniel Tan, Ryan Tolsma, Wanqi Zhu

June 2018

1 Introduction

We consider here the problem of L_p-norm estimation, which can be described as follows. Imagine that we have a sequence of elements $A = a_1, a_2, \ldots, a_m$, where $a_i \in 1 \ldots n$. Crucially, we impose the requirement that the elements a_i are not available through random access; they may arrive one at a time in a "time stream", or they may be database entries that take a prohibitively long time to load from memory. Thus, if we need to access this data, we are restricted to traversing the sequence in a single pass from start to end.

For any given $t \leq m$, we define $x^{(t)}$ to be the frequency vector of elements $1, \cdots, n$ up to that point, i.e. $x_i^{(t)} = |\{j | a_j = i, j \leq t\}|$. The fundamental problem to solve is that, for any t, we need to be able to estimate the L_p-norm of the data, defined as follows.

Definition 1.1. *The L_p-norm of a vector $x = (x_0, \ldots, x_n)$ is given by $||x^{(t)}||_p = (\sum_{i=0}^{n} x_i^p)^{\frac{1}{p}}$*

For applications in which n is not too large, the solution is trivial. We can simply store $x^{(t)}$ directly in memory with n counters each counting up to m, updating it as we pass over the sequence, and directly compute the L_p norm whenever queried. Thus updates to the vector are $O(1)$, memory usage is $O(n \log m)$, and queries are $O(n)$. Thus, we can answer any k queries in one pass over the sequence with $O(m + kn)$ time, using a naive implementation.

However, there are many applications for which both m and n are too large to store directly in memory. As an example, a sensor deployed to monitor heavy metal concentrations in a river might only have a single microprocessor with only a few megabytes of RAM. In such "streaming applications", we need to be able to effectively estimate $||x^{(t)}||_p$ using sublinear space. In exchange, we sacrifice the exact guarantees of the naive solution for probabilistically correct estimates. In this paper, we explore two complementary approaches to estimating the L_p norm - first, a sampling-based method, and then a hashing-based method.

In the following analysis, we will fix t and define $m_i = |\{j | a_j = i, j \leq t\}|$, the number of elements equal to i in the stream.

2 Frequency moment estimation by sampling

In this section, we introduce a sampling-based method for estimating the frequency moment. First, we introduce the concept of a frequency moment.

Definition 2.1. *The k^{th} frequency moment F_k of a stream is given by $F_k(A) = \sum_{i=1}^{n} m_i^k$, where $k \in Z$.*

We see that the definitions of L_p-norm and k^{th} frequency moment are quite similar; indeed, they are identical up to exponentiation. Thus, an estimate of one provides an estimate of the other, since exponentiation is a one-to-one monotonic function. In this section, we will describe a data structure that, given parameters $\epsilon > 0, \lambda > 0$, provides an estimate of the L_p norm to within a factor of $1 \pm \lambda$ with probability $1 - \epsilon$ and uses space:

$$O\left(\frac{p \log(1/\epsilon)}{\lambda^2} n^{1-1/p} (\log n + \log m)\right)$$

2.2 Background and Intuition

In the following section, we will go through a rigorous description and proof of the algorithm. Before that, however, we'd like to give an intuition as to why we define our random variables X as so, we make the following observations. Let $f_j = m_{a_j}$ be the frequency of a_j.

$$F_k = \sum_{i=1}^{n} m_i^k \tag{1}$$

$$= \sum_{i=1}^{n} \sum_{a_j=i} f_j^{k-1} \tag{2}$$

$$= \sum_{j=1}^{m} f_j^{k-1} \tag{3}$$

where line 2 comes from the fact that $|\{a_j \mid a_j = i\}| = m_i$, i.e. there are exactly m_i elements in the stream of value i by definition. Intuitively, we can assign m_i^{k-1} to each of the m_i elements of the stream that have value equal to i, and so summing over $a_1, \cdots, a_m$ gives us the desired sum over m_i^k's.

The above representation of F_k is nice, because the individual elements in the stream are independent. As such, if we let X be a uniformly chosen index from $1, 2, \ldots, m$, corresponding to value $v = a_X$, we have

$$\mathrm{E}[F_k] = m \cdot m_v^{k-1} = m \cdot f_X^{k-1}.$$

So if we could keep count of the frequency of **one** element a_X, we'd be able to estimate F_k on average. However, since we only pass through the stream once, we can't keep track of how many times v appeared before index X as we won't know our element's value until we see index X. So the natural thing to do is to keep track how many times v appear after index X in the stream, which takes logarithmic space and is precisely what the paper tracks in r.

So we know that we want to estimate based on some expression on r for $m \cdot f_i^{k-1}$. Note that among all f_i appearances of value v in the data stream $\{x_{i_1} = x_{i_2} = \cdots = x_{i_{f_i}}\}$, we're equally likely to have chosen index X to be any one of $i_1, \cdots, i_{f_i}$, and so r is uniformly distributed among $1, 2, \cdots, f_i$.

Fixing the value v, we want, on expectation, each of those possibilities of r to contribute to $m \cdot f_i^{k-1}$. So it's natural to use a telescoping function which sums to $f_i \cdot (m \cdot f_i^{k-1}) = m(f_i^k)$ over the f_i possibilities of $r = 1, 2, \cdots, f_i$, and so the choice of $X = m(r^k - (r-1)^k)$ follows naturally.

3 Proving the Upper Bound Space Usage in Estimating F_k

The primary advantage associated with sampling and sketching based algorithms and data structures is from their low space usage. While we would like to still achieve error bounds with probability at most ϵ with confidence $1 - \lambda$, these impose restraints on the minimum amount of memory storage required. With some careful analysis, we can see that the frequency moment sampling methods we described can be upper bounded in terms of required memory storage. Like many probabilistic data structures, the process and algorithm are often quite simple, but proving their correctness can be challenging. This section will be devoted to the mathematical analysis involved in proving these upper bounds.

3.1 Bounding Particular Sums

The following lemma will be useful in proving the error bounds on F_k estimation.

Lemma 3.1.1. *For every n positive reals $m_1, \ldots, m_n$*

$$\left(\sum_{i=1}^{n} m_i\right)\left(\sum_{i=1}^{n} m_i^{2k-1}\right) \leq n^{1-1/k}\left(\sum_{i=1}^{k} m_i^k\right)^2.$$

Proof. Set $M = \max_{i \leq i \leq n} m_i$, then we have that $M^k \leq \sum_{i=1}^{n} m_i^k$ and hence we see that

$$\begin{aligned}
\left(\sum_{i=1}^{n} m_i\right)\left(\sum_{i=1}^{n} m_i^{2k-1}\right) &\leq \left(\sum_{i=1}^{n} m_i\right)\left(M^{k-1}\sum_{i=1}^{n} m_i^k\right) \\
&\leq \left(\sum_{i=1}^{n} m_i\right)\left(\sum_{i=1}^{n} m_i^k\right)^{(k-1)/k}\left(\sum_{i=1}^{n} m_i^k\right) \\
&= \left(\sum_{i=1}^{n} m_i\right)\left(\sum_{i=1}^{n} m_i^k\right)^{(2k-1)/k} \\
&\leq n^{1-1/k}\left(\sum_{i=1}^{n} m_i^k\right)^{1/k}\left(\sum_{i=1}^{n} m_i^k\right)^{(2k-1)/k} = n^{1-1/k}\left(\sum_{i=1}^{n} m_i^k\right)^2
\end{aligned}$$

where the last inequality uses the common fact that $(\sum_{i=1}^{n} m_i)/n \leq (\sum_{i=1}^{n} m_i^k/n)^{1/k}$.

□

3.2 Analysis of the Space Complexity of Estimating F_k

Now we can proceed to prove our upper bounds on space usage.

Theorem 1. *For every $k \geq 1$, every $\lambda > 0$ and every $\epsilon > 0$ there exists a randomized algorithm that computes, given a sequence $A = (a_1, \ldots, a_m)$ of members of $N = \{1, 2, \ldots, n\}$ in one pass and using $O\left(\frac{k\log(1/\epsilon)}{\lambda^2} n^{1-1/k}(\log n + \log m)\right)$ memory bits, a number Y so that the probability that Y deviates from F_k by more than λF_k is at most ϵ.*

Proof. Define $s_1 = \frac{8kn^{1-1/k}}{\lambda^2}$ and $s_2 = 2\log(1/\epsilon)$. Then we can assume that the length of the sequence m is known in advance, and then comment on the required modifications if that turns out to not be the case.

The algorithm computes s_2 random variables $Y_1, Y_2 \ldots, Y_{s_2}$ and outputs their median Y. Each Y_i is written as the average of s_1 random variables $X_{ij} : 1 \leq j \leq s_1$ where the X_{ij} are independent, identically distributed random variables. Each of the variables $X = X_{ij}$ is computed may be computed and stored using only $O(\log n + \log m)$ memory bits with the following procedure.

From our sequence of elements A, choose an element a_p by random where p is chosen with uniform randomness from $\{1, 2, \ldots, m\}$. Given a choice of p, suppose that $a_p = l \in N$. Then we can define a useful variable

$$r = |\{q : q \geq p, a_q = l\}|$$

that represents the number of occurrences of l in the sequence A that appear after a_p (inclusive). Now we can define the indicator variable

$$X_i = m_i\left(r^k - (r-1)^k\right).$$

Clearly, we need only store the $\log m$ bits for r and the $\log n$ bits representing a_p which implies that only $O(\log n + \log m)$ memory bits are required as stated earlier.

Using these indicator variables $X = \sum_{i=1}^{m} X_i$ we can construct upper bounds on the estimation errors. We can calculate the expected value of X

$$
\begin{aligned}
E[X] &= E[\sum_{i=1}^{m} X_i] \\
&= \sum_{i=1}^{m} E[X_i] \\
&= \sum_{j=1}^{n} \sum_{X_i=j} \mathrm{E}\left[X_i\right] \\
&= \frac{m}{m}\left[\left(1^k + (2^k - 1^k) + \ldots + (m_1^k - (m_1 - 1)^k)\right) + \right. \\
&\quad \left(1^k + (2^k - 1^k) + \ldots + (m_2^k - (m_2 - 1)^k)\right) + \ldots + \\
&\quad \left(1^k + (2^k - 1^k) + \ldots + (m_n^k - (m_n - 1)^k)\right) \qquad \textit{program@epstop} \\
&= \sum_{i=1}^{n} m_i^k = F_k
\end{aligned}
$$

where the expanding sums come from the fact that among the stream elements equal to a particular j, we're equally likely to have picked the first, second, or the last element, and those correspond to $r = m_j, m_j - 1, \cdots, 1$, respectively.

From the formula for variance $\mathrm{Var}[X] = E[X^2] - \left(E[x]^2\right)$ we can compute an upper bound on the variance by bounding $E[X^2]$;

$$
\begin{aligned}
\mathrm{Var}[X] &\leq E\left[\left(\sum_{i=1}^{m} X_i\right)^2\right] \\
&= \sum_{i=1}^{m} E[X_i^2] \\
&= \frac{m}{m}\left[\left(1^k + (2^k - 1^k)^2 + \ldots + \left(m_1^k - (m_1 - 1)^k\right)^2\right) + \right. \\
&\quad \left(1^{2k} + (2^k - 1^k)^2 + \ldots + \left(m_2^k - (m_2 - 1)^k\right)^2\right) + \ldots + \\
&\quad \left(1^{2k} + (2^k - 1^k)^2 + \ldots + \left(m_n^k - (m_n - 1)^k\right)^2\right) \qquad \textit{program@epstop}
\end{aligned}
$$

Now using the fact that for any two numbers $a > b > 0, a, b \in R$ we can factor $a^k - b^k$ to yield the following inequality

$$
a^k - b^k = (a - b)(a^{k-1} + a^{k-2}b + \ldots + ab^{k-2} + b^{k-1}) \leq (a - b)ka^{k-1}
$$

Applying this to our current expression for variance leads to

$$
\begin{aligned}
E\left[\left(\sum_{i=1}^{m} X_i\right)^2\right] &\leq m\left[\left(k \cdot 1^{2k-1} + k \cdot 2^{k-1}(2^k - 1^k) + \ldots + k \cdot m_1^{k-1}(m_1^k - (m_1 - 1)^k)\right) + \right. \\
&\quad \left(k \cdot 1^{2k-1} + k \cdot 2^{k-1}(2^k - 1^k) + \ldots + k \cdot m_2^{k-1}(m_2^k - (m_2 - 1)^k)\right) + \ldots + \\
&\quad \left(k \cdot 1^{2k-1} + k \cdot 2^{k-1}(2^k - 1^k) + \ldots + k \cdot m_n^{k-1}(m_n^k - (m_n - 1)^k)\right) \qquad \textit{program@epstop} \\
&\leq m \sum_{i=1}^{n} k m_i^{2k-1} \\
&= kmF_{2k-1} = kF_1F_{2k-1}
\end{aligned}
$$

Now, applying our results from the Lemma, we can easily see from the definition of our random variables Y_i that

$$\text{Var}[Y_i] = \text{Var}[X]/s_1 \leq E[X^2]/s_1 \leq kF_1F_{2k-1}/s_1 \leq kn^{1-1/k}F_k^2/s_1$$

considering that

$$E[Y_i] = E[X] = F_k.$$

Finally, by applying Chebyshev's Inequality and using the definition of s_1, for every fixed i we have that

$$\Pr\left[[]\left[Y_i - F_k\right] > \lambda F_k\right] \leq \frac{\text{Var}[Y_i]}{\lambda^2 F_k^2} \leq \frac{kn^{1-1/k}F_k^2}{s_1\lambda^2 F_k^2} = \frac{kn^{1-1/k}/\lambda^2}{s_1} = \frac{1}{8}$$

This implies that the probability that a single Y_i deviates from F_k by more than λF_k is at most $\frac{1}{8}$, and by applying the Chernoff bound it can be easily seen that the probability that more than $s_2/2$ of the variables Y_i deviate by more than λF_k from F_k is at most ϵ. Whenever this does not happen, the median of the Y_i presents a very reasonable estimate of the quantity F_k as needed. □

While the proof above assumed that the sequence length m was known in advance, we can easily extend it to the unknown case utilizing the standard process for sampling a variable from a sequence with uniform randomness. In the general case, with a sequence of elements $B = \{b_1, \ldots\}$ of unknown length, we start by choosing our first value $y = b_1$. Then, for each i, with probability $\frac{1}{i}$ we set $y = b_i$. Then, this process samples a single element from B with uniform randomness as desired.

Similarly, in the F_k estimation, for each value of m we may replace a_l with a_m by the process just described. If we do replace the value then we set $r = 1$, otherwise a_l remains as is and increases by 1 in the case $a_m = a_l$.

4 Implementation

We implemented the sampling-based frequency moment estimator in Python. The data structure itself is very simple, requiring only about 10 lines of Python code:

```
def lp_norm_sketch(A, k, n, _lambda, _epsilon):
    # compute the number of counters we need
    s1 = math.ceil(8*k*n**(1-1/k)/(_lambda**2))
    s2 = math.ceil(2*math.log(1/_epsilon))

    X = np.zeros((s2, s1), dtype=int)  # the random element we care about
    r = np.zeros((s2, s1))  # stores the frequency count

    # Update counters in parallel as we pass over the stream
    for cnt, a in enumerate(A):
        to_change = np.random.rand(s2, s1) < 1/(cnt+1)
        X[to_change] = a
        r[to_change] = 0  # reset counter as needed
        r[X == a] += 1

    return np.median(np.mean((cnt+1)*(r**k - (r-1)**k), axis=1), axis=0)
```

5 Performance Benchmark

We tested our code on for various choices of parameter. Here is a sample plot from a test with 100 independent estimates of F_2 , with $n = 10$, $m = 100$, $\lambda = 0.1$, $\epsilon = 0.01$.

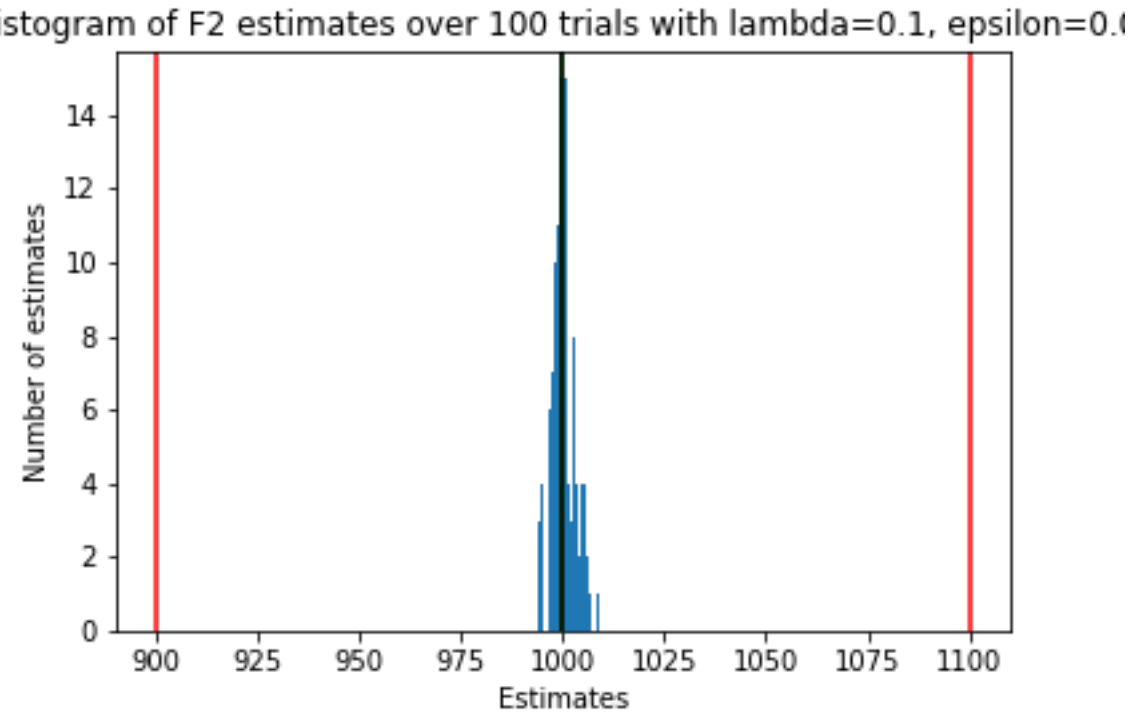

Figure 1: Sample plot. The other plots mostly look like this too.

Clearly, the algorithm runs within the expected error margin, providing empirical validation of the theoretical result given. In addition, we tested the algorithm for various values of k, λ and ϵ (plots not shown), and verified that the plots mostly look like the one shown here.

6 Interesting component

6.1 Part 1: Tighter theoretical bounds

The Hoeffding's Lemma states that for independent bounded variables $Z_1, \ldots, Z_n$ with $Z_i \in [a, b]$ for all i where $-\infty < a \leq b < \infty$ then

$$Pr[\frac{1}{n}\sum_{i=1}^{n}(Z_i - E[Z_i]) \geq t \leq \exp(\frac{-2nt^2}{(b-a)^2}.$$

where $0 < t < \sum_{i=1}^{n}(Z_i - E[Z_i])$.

We can apply this inequality to our AMS sampling algorithm if we know that the frequency of any given element is bounded by some certain value $N \in \{0, \ldots, n\}$. By definition we have that

$$Y_i = \frac{1}{s_1}\sum_{j=1}^{s_1} X_j$$

Since we know that for each j $E[X_j] = F_k$ it follows that

$$Y_i - F_k(\frac{1}{s_1}\sum_{j=1}^{s_1} X_j) - F_k = Y_i - E[Y_i]$$

Then we have the following expression for our error probablities

$$Pr[Y_i - F_k \geq \lambda F_k] \leq \exp(\frac{-2\lambda^2 F_k^2}{N^2})$$

which is a tighter bound than what we had achieved earlier when N not too large. In the limit as $N \to \infty$ this will be less tight than the inequalities and results seen by Chebychev and the Chernoff bounds.

6.2 Part 2: Tweaking the constants

During the experiments with the data structure, we noticed a few things.

1. The theoretical bounds are quite weak - the data structure seems to do a lot better in practice than it does on paper.
2. The runtime may be quite high for values of $\lambda < 0.1$. This is because we require $s_1 = O(\lambda^{-2})$ counters in each row of the grid, making each update run in $O(\lambda^{-2})$ time.
3. In contrast, the runtime grows very slowly with ϵ, as expected: the dependence is only $\log \epsilon^{-1}$, so the algorithm can be run in a reasonable time even for very small values of ϵ.

Obviously, the dependence on λ^{-2} is an inherent property of the data structure. But it seems natural to ask: can we get away with reducing s_1 by a constant factor? In the paper, the value given is $s_1 = \frac{8kn^{1-1/k}}{\lambda^2}$. We consider replacing the constant 8 in the expression with a general constant q and experiment with varying values of q.

6.2.1 Histogram of Estimates for Varying q

The results of estimates over different values of q are summarized on the next page. As observed, decreasing the value of q increases the spread of the distribution. Notably, there seems to be a sweet spot: setting $q = 0.1$ still maintains the probabilistic guarantee of the data structure of error within $1 \pm \lambda$. In addition, $q = 0.1$ represents an 80x speedup in runtime over $q = 8$ - this is the difference between running in minutes and running in seconds. In practice, fine-tuning the constant q could be very useful for high-throughput data streams where the rate of influx of new information is very high and speed is a major requirement.

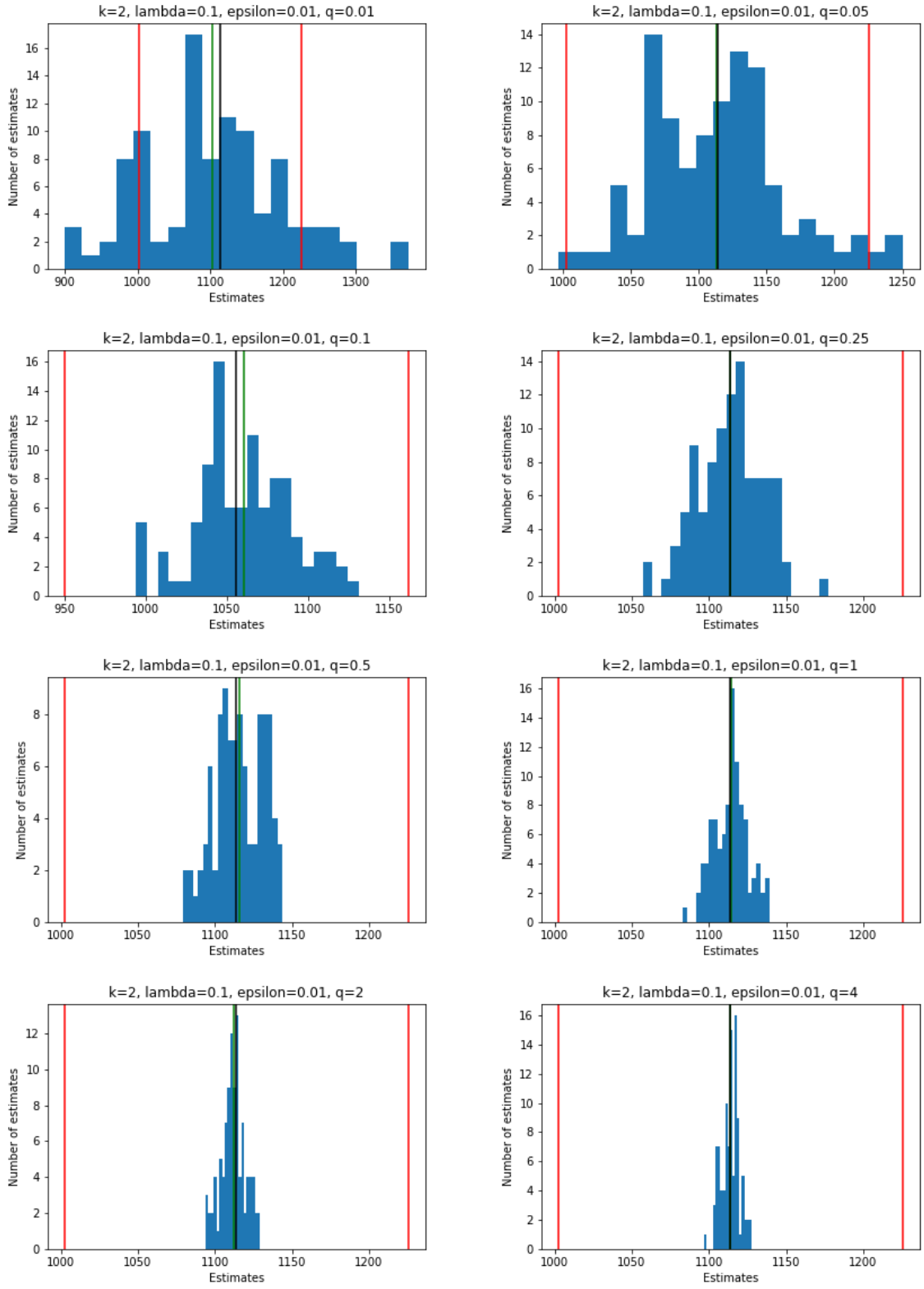

Figure 2: Estimate distributions for $q = 0.01, 0.05, 0.1, 0.25, 0.5, 1, 2, 4$ respectively

6.2.2 Varying λ with $q = 0.1$

We tested various values of λ with the new value of $q = 0.1$ instead of 8. The results shown below suggest that the probabilistic guarantee is still maintained no matter what we set λ to.

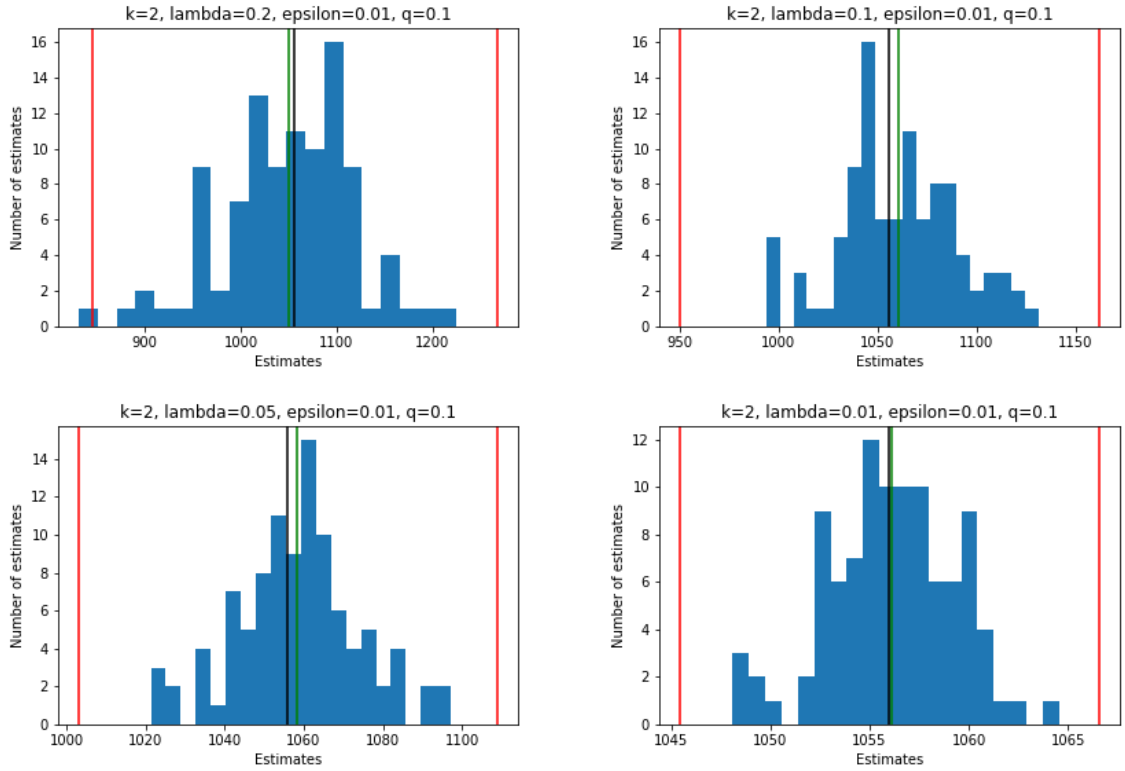

Figure 3: Estimate distributions for $q = 0.1$, $\lambda = 0.2, 0.1, 0.05, 0.01$

In conclusion, we can achieve speedup by a large constant factor (80x) by fine-tuning the parameter q in s_1 while still conforming to the algorithm's probabilistic guarantee. For further testing, the code is available here[1].

6.3 Theoretically optimum q value

In addition to testing the constants in practice, we can analyze mathematically how to minimize the space and time usage, which are both porportional to $s_1 s_2$, varying both $s1$ and $s2$.

Fixing λ and ϵ, we know from above that $s_1 = \frac{q_1 k n^{1-1/k}}{\lambda^2}$ and $s_2 = q_2 \log 1/\epsilon$ for constants q_1, q_2. These big-O scales are optimal provided that our bound in $Var[X]$ is tight (which can be improved, but not asymptotically), and so we will only analyze the constants for practical improvements. In order to apply Chernoff bound, we need

$$Pr[Y_i - F_k \geq \lambda F_k] \leq \frac{k n^{1-1/k}}{s_1 \lambda^2} = 1/q$$

to be less than $\frac{1}{2}$. Furthermore, in the Chernoff bound, we have

[1] https://github.com/wanqizhu/lp-norm-sketch

$$
\begin{aligned}
& e^{\frac{-n(1/2-p)^2}{2p}} \leq \epsilon \\
\implies & e^{-\frac{q_2 \log 1/\epsilon (1/2-1/q_1)^2}{2/q_1}} \leq \epsilon \\
\implies & e^{-\log 1/\epsilon \cdot \frac{q_1 q_2 (1/2-1/q_1)^2}{2}} \leq \epsilon \\
\implies & \epsilon^{\frac{q_1 q_2 (1/2-1/q_1)^2}{2}} \leq \epsilon \\
\implies & \frac{q_1 q_2 (1/2-1/q_1)^2}{2} \geq 1 \\
\implies & q_1 q_2 \geq \frac{2}{(1/2-1/q_1)^2}
\end{aligned}
$$

where the second to last line has the inequality flipped since $\epsilon < 1$.

We note that $c_1 c_2 = C \cdot q_1 q_2$ for a constant C that's fixed given ϵ and λ, so it suffices to minimize $q_1 q_2$. From above, we need $q_1 > 2$ and want to minimize $\frac{2}{(1/2-1/q_1)^2}$. This would suggest us making q_1 as big as possible, since $\frac{2}{(1/2-1/q_1)^2}$ is minimized at $q_1 = \inf$ with value 8. However, we need to note that s_1 and s_2 are integers, and in order to ensure our ϵ and λ bounds, we need to take the ceiling of their respective expressions given any q_1, q_2. So once q_2 gets small enough that $s_2 = q_2 \log 1/\epsilon \leq 1$, increasing q_1 will no longer affect q_2 and thus will only increase $q_1 q_2$.

As such, the theoretically optimum constants are choosing $q_2 = \frac{1}{\log 1/\epsilon}$ and choosing q_1 such that $q_1 q_2 (1/2 - 1/q_1)^2/2 = 1$, and setting $s_1 = \lceil \frac{q_1 k n^{1-1/k}}{\lambda^2} \rceil$ and $s2 = \lceil q_2 \log 1/\epsilon \rceil = 1$.

This result is very surprising, as it essentially collapses the grid into a single row, foregoing the median and just taking the mean over all $s_1 X$'s! In practice, choosing constants as from the previous section seem to give better results, and this is likely due to the not-so-tight bound on $Var[X]$.

We hope this was suitably interesting :)

7 L_p-norm Estimation with hashing

In addition to the sampling-based approach described above, there exists another, quite different, approach to estimating L_p-norms, based on projections onto random hyperplanes. This new approach has the advantage of being able to deal with fractional and negative updates as opposed to only incrementing by 1, and works for non-integer values of $p \in (0, 2]$. We did not do anything interesting for this data strucutre, but we include it here for the sake of completeness.

Essentially, what we want is an estimator whose expected value is $||x^{(t)}||_p$, i.e. an *unbiased estimator*. We also want it to satisfy certain concentration bounds, such that with "reasonably high" probability, the estimator does not deviate "too much" from the expected value. We will now proceed to formalize these notions.

7.1 Obtaining an unbiased estimator

We start with the simpler case of $p = 2$, i.e. we are performing L_2-norm estimation. The hashing-approach is based on the following observation.

Lemma 7.1.1. *Suppose we are given a n-dimensional vector $x^{(t)}$ and a n-dimensional vector r of pairwise independent normal variables r_i $N(0, 1)$. If we define $Z = \sum r_i x_i$, then Z^2 is an unbiased estimator of $||x^{(t)}||_2^2$.*

Proof.

$$E[Z^2] = E[(\sum_i r_i x_i)^2] \tag{4}$$

$$= E[\sum_i \sum_j r_i r_j x_i x_j] \tag{5}$$

$$= E[\sum_i x_i^2] \tag{6}$$

$$= ||x_i||_2^2 \tag{7}$$

Here, step 6 is justified by the fact that $E[r_i] = 0$, and since each pair of r_i, r_j are independent, we have that $E[r_i r_j] = E[r_i]E[r_j] = 0$ for $i \neq j$. Furthermore, we also have $E[r_i^2] = Var(r_i) + E[r_i]^2 = 1$. Hence, Z^2 is an unbiased estimator of the L_2-norm as required.

7.2 Analyzing the variance

Now we have an unbiased estimator, which gives us the correct value in expectation. However, we are also interested in the concentration of the estimator around its expected value; intuitively, if the estimator deviates greatly from its expected value most of the time, then it is not a very good estimator. We now make the following claim about its variance.

Lemma 7.2.1. *Given the estimator Z^2 described above, we have that $Var(Z^2) \leq 3(\sum_i x_i^2)^2$.*

Proof. *We have that:*

$$E[Z^4] = E[(\sum_i r_i x_i)^4] \tag{8}$$

$$= E[(\sum_i r_i x_i)(\sum_j r_j x_j)(\sum_k r_k x_k)(\sum_l r_l x_l)] \tag{9}$$

By linearity of expectation, we can expand the expression on the right and take the expectation of each term, and add up the expected values to get the overall expected values. After expanding, the terms on the right hand side are all degree 4 polynomials in r. Now observe that we can break the terms down into:

1. *Terms of the form $E[r_i^4 x_i^4]$, of which there is 1 for each choice of i. Using without proof the fact that $E[r_i^4] = 3$, we see that the sum of such terms gives us $3\sum x_i^4$.*
2. *Terms of the form $E[r_i^2 r_j^2 x_i^2 x_j^2]$, of which there are 6 for each unique unordered tuple (i, j). Since r_i, r_j are independent, and $E[r_i^2] = 1$ as proved earlier, the sum of such terms is just $6(\sum_{i<j} x_i^2 x_j^2)^2$.*
3. *Terms which contain one or more odd powers of r_i, in which case the term vanishes as $E[r_i] = 0$.*

Thus, we have that:

$$Var(Z^2) = E[Z^4] - E[Z^2]^2 \tag{10}$$

$$\leq E[Z^4] \tag{11}$$

$$= 3\sum x_i^4 + 6(\sum_{i<j} x_i^2 x_j^2)^2 \tag{12}$$

$$= 3(\sum_i x_i^2)^2 \tag{13}$$

as desired.

7.3 Applying the Chebyshev Inequality

To recap, we now have an estimator Z^2 such that:

1. $\mu = E[Z^2] = \sum x_i^2$
2. $\sigma^2 = Var(Z^2) \leq 3(\sum x_i^2)^2$

Thus, by the Chebyshev inequality, $Pr(|Z^2 - \mu| > c\sigma) \leq 1/c^2$. Unfortunately, for the given values of μ and σ, this is not a very good bound. As a simple example, in order to achieve a confidence of 75%, we require an interval spanning 6 standard deviations in both directions of the mean. We would like to derive an estimator with less variance around the mean. Fortunately, this is readily achievable, as we explain in the next section.

7.4 The Magic of Independent Trials

We suppose Z^2 is an estimator as described above. We consider the estimate derived by instantiating w pairwise independent such estimators $\{Z_j^2\}$, querying all of them, and taking the average estimate $Y = \frac{1}{w}\sum Z_j^2$ of these estimates. Intuitively, this allows us to average out the "outlier" observations to obtain an overall estimate which is closer to the mean. We formalize this concept in the following lemma.

Lemma 7.4.1. *Let $\{Z_j^2\}$ be w independent estimators as previously described. Then, given any $\epsilon > 0$, we can choose w such that the new estimator $Y = \frac{1}{w}\sum Z_j^2$ satisfies:*

1. $E[Y] = ||x||_2^2$
2. $Pr(|Y - E[Y]| > \epsilon||x||_2^2) \leq 1/e$

Proof. *We note that variance of Y is just $\frac{3}{w}||x||_2^2$. Thus, by the Chebyshev inequality, we have:*

$$Pr(|Y - E[Y]| > c\sqrt{3/w}||x||_2^2) \leq 1/c^2 \tag{14}$$

We can choose $w = \lceil \frac{3e}{\epsilon^2} \rceil$ and $c = \sqrt{e}$ to obtain:

$$Pr(|Y - E[Y]| > \epsilon||x||_2^2) \leq 1/e \tag{15}$$

We now show that we can achieve an arbitrarily low failure rate by taking the median estimate of multiple such Y estimators.

Lemma 7.4.2. *Let $\{Y_i\}$ be d estimators as described in the previous lemma and define Y' to be the estimate derived by taking the median of these d estimators. Then given any $\delta > 0$, we can choose d such that $Pr(|Y - E[Y]| > \epsilon||x||_2^2) \leq \delta$.*

Proof. *The median estimate will only be off by more than a factor of ϵ if at least $d/2$ of the individual Y-estimates are also off. Each of the d estimators fails with probability $1/e$. Thus we can model the number of failures as a Binomial distribution $B(d, 1/e)$. By the Chernoff bound, we have that:*

$$Pr(X > d/2) \leq e^{\frac{-d(1/2-1/e)^2}{2/e}} \tag{16}$$

$$= e^{-Cd} \text{ for some constant } C \tag{17}$$

Thus by setting $d = C^{-1}\log(\delta^{-1})$ we derive the desired bounds.

7.5 Implementation Details

Having completed the theoretical analysis, we can now formally describe the data structure in full. Given parameters $\epsilon, \delta > 0$, we pick $w = O(\epsilon^{-2})$, $d = O(\delta^{-}1)$. We instantiate d copies of a Y-estimator, which in turn consists of w Z-estimators, each of which consists of a single hash function that hashes stream elements x_i to values r_i drawn from a standard normal distribution and maintains the sum of $\sum_i r_i x_i$ across updates.

When queried, it computes the median-of-means estimate as described in the previous section.

The upside is that the required space is simply $O(\epsilon^{-2} \log \delta^{-1})$, without any dependence on stream length m or number of elements n, and so this data structure is suitable for use in very long data streams or where the number of elements is impossibly large. In addition, this data structure easily handles fractional and negative updates, which may be required in certain applications e.g. sensing the concentration of chemicals in a river. The downside is that updates and queries require $O(\epsilon^{-2} \log \delta^{-1})$ time, so in practice ϵ and δ must be selected carefully so as to not make the runtime explode.

One minor point is that in order for the analysis up to now to hold, we require that each hash function h_j is 4-independent, and furthermore that any two such hash functions h_j, h_k are pairwise independent.

7.6 Extension to general L_p norms

The approach above can be generalized to estimating L_p-norms for any $p \in (0, 2]$. The key is that instead of drawing r_i from a standard normal distribution, r_i have to be drawn from what is known as a $p - stable$ distribution. A p-stable distribution U is one that has the property $x_1U + x_2U + \cdots + x_nU$ has the distribution $(x_1^p + x_2^p + \cdots + x_n^p)^{1/p}U$. Then Z^p is an estimator of the L_p-norm of the frequency vector.

These functions are not well understood for values of $p \neq 0.5, 1, 2$, but it is still possible to generate variables according to their distribution through various clever and complicated sampling methods which are beyond the scope of this paper. For interested readers, the wikipedia page[2] provides an excellent introduction to such distributions.

8 Lower Bounding on the Space Requirements

We outline ideas in this section that show any algorithm that can estimate L_k-norm within a fixed (ϵ, λ) error bound needs at least $\Omega\left(n^{1-5/k}\right)$ memory bits. In particular, this means estimating the most frequent element count, F_{inf}, requires at minimum $\Omega(m)$ space.

The central idea is reducing the problem to a communication complexity problem, as outlined in Alon, Matias, and Szegedy. The proof is omitted here, but the intuition is that given a black box L_k norm estimator, we can model a communication problem where two people, each having x and y, want to compute $f(x, y)$ for some binary function f using as few bits as possible. It turns out that certain problems, such as the disjointness function, require $\Omega(n)$ space and can be uniquely determined by F_{inf}, which can be computed by one person using our black box on x and then passing his memory contents to the other to run on y. This shows that any such black box will use $\Omega(m)$ space.

9 Conclusion

In conclusion, we have described two fundamentally different approaches to the problem of L_p-norm estimation, each with their strengths and weaknesses. The sampling-based approach can estimate the L_p-norm for any integer value of $p >= 2$, whereas the hashing-based approach can estimate the L_p-norm for real values of $p \in (0, 2]$. We have also briefly explored some theoretial lower and upper bounds on the space complexity of L_p-norm estimation.

[2] https://en.wikipedia.org/wiki/Stable_distribution