The L_p-norm Sketch

A probabilistic sampling-based data structure for estimating L_p norms of data streams

Daniel Tan, Ryan Tolsma, Wanqi Zhu

June 5, 2018

Python Magic!

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

    for cnt, a in enumerate(A):
        # since we don't know m in advance, we
        # update X and r as we go
        # when we encounter the m-th element, we update w.p. 1/m
        to_change = np.random.rand(s2, s1) < 1/(1+cnt)
        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)
```

Figure: L_p norm in 10 lines of code!

Background

Data structures for analyzing data streams differ considerably from more conventional data structures.

The count-min sketch and count-sketch are probabilistic data structures that trade-off **exactness** for **reduced space usage**.

Question: Can this idea be extended to other problems in the context of data streams?

Definitions

1. A **data stream** consists of elements $a_1, a_2, \ldots, a_m$, such that each $a_i \in \{1 \ldots n\}$.
2. A **frequency vector** at time t is an n-dimensional vector $x^{(t)}$ whose components count the frequencies of stream elements i up to that point: $x_i^{(t)} = |\{j | a_j == i, j \leq t\}|$
3. The L_p**-norm** of a vector x is defined as: $||x||_p = (\sum_i x_i^p)^{1/p}$
   3.1 L_1 norm is the sum of frequency counts
   3.2 L_2 norm is the square root of sum of squares of frequency counts, etc.

The L_p-norm Sketch

A data structure that estimates $||x^{(t)}||_p$ for any value of t, using space sublinear in m, n.

Why would we want to do this?

Statistical applications: L_2-norm is used to calculate important statistical quantities like the *surprise index* and *Gini's coefficient of homogeneity*.

Also an indicator of the *skew* of a distribution

How do we do it?

Let's use a sampling-based approach!

Digression 1: Reservoir Sampling

Given a data stream of elements $a_1, a_2, \ldots, a_m$, we want to randomly pick one.

Naive solution: Generate a random index r between 1 and m and pick that index.

The Catch: To do that, we need to know m beforehand. What if m is unknown?

Digression 1: Reservoir Sampling

Solution: Start with $i = 0, r = 0$. While there are still elements in the stream, do:

1. Set $r = i + 1$ with probability $\frac{1}{i+1}$
2. Increment i by one

Claim: After $k \geq 1$ iterations, r is uniformly distributed over $\{1 \ldots k\}$.

Digression 1: Reservoir Sampling

Proof: Induction.

Base case: For $k = 1$, the hypothesis asserts that r is 1 with probability 1. This is true!

Digression 1: Reservoir Sampling

Inductive step: Assume the statement is true for some k. Consider the case $k+1$:

1. Before the step, for any $1 \leq j \leq k$, we have $Pr(r=j) = 1/k$.
2. Regardless of r's value, we update r to $k+1$ with probability $1/(k+1)$. i.e. $Pr(r=k+1) = 1/(k+1)$
3. For any $1 \leq j \leq k$, we now have $Pr(r=j) = k/(k+1) * 1/k = 1/(k+1)$.
4. Thus r is uniform over $\{1 \dots k+1\}$. QED

Intuition

Now that we can randomly pick an element from our data stream, if we could figure out its frequency, we'd be able to accurately estimate L_p.

Fixing t, we define the k^{th} **frequency moment** of a stream as $F_k = \sum_{i=1}^{n} m_i^k$, where $m_i = |\{j | a_j = i, j \leq t\}|$ is the count of the **value** i.

We'll also define $r_j = m_{a_j}$ to be the frequency of the value at **index** j.

Intuition

If our random element is at index X with frequency R, then

$$E[F_k] = \sum_{i=1}^{n} E[m_i^k]$$

$$= \sum_{i=1}^{n} \sum_{a_j = i} m_i^{k-1}$$

$$= \sum_{j=1}^{m} r_j^{k-1}$$

$$= m \cdot R^{k-1}.$$

Intuition

However, since we only pass through the stream once, we can't count the exact frequency. We don't know what appeared before index X!

What we can do, in logarithmic space, is to store the value $v = a_X$ and track the frequency r from index X forward in the stream.
It turns out that this value is enough to estimate R and thus F_k.

Intuition

Among all R appearances of value v in the data stream $\{x_{j_1} = x_{j_2} = \cdots = x_{j_R}\}$, we're equally likely to have chosen index X to be any one of $j_1, \cdots, j_R$, and so r is uniformly distributed among $1, 2, \cdots, R$. If we have some function f on r, we want on expectation

$$f(1) + \cdots + f(R) = R \cdot mR^{k-1}$$

Intuition

If we have some function f on r, we want on expectation

$$f(1) + \cdots + f(R) = R \cdot mR^{k-1}$$

One such function is $f(r) = m(r^k - (r-1)^k)$.

Intuition

Outline:

1. Use reservoir sampling to randomly pick an element X.
2. Keep track of its frequency count r going forward.
3. Use $m(r^k - (r-1)^k)$ to estimate F_k
4. Repeat and take mean & median for good error bonding.

Formal Analysis

Theorem: 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.

Formal Analysis continued...

1. Define $s_1 = \frac{8kn^{1-1/k}}{\lambda^2}$ and $s_2 = 2\log(1/\epsilon)$.
2. Assume that m is known in advance.
3. We will use $s_1 \cdot s_2$ random variables.

Formal Analysis Continued...

1. The algorithm computes s_2 random variables $Y_1, Y_2 \ldots, Y_{s_2}$ and outputs their median Y.
2. 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.
3. 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.

Calculating X_i

1. 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 $a_p = l \in N$.
2. Define $r = \left|\{q : q \geq p, a_q = l\}\right|$ to represent the number of occurrences of l in the sequence A that appear after a_p (inclusive).
3. Define $X_i = m_i \left(r^k - (r-1)^k\right)$.

Calculating $E[X]$

$$
\begin{aligned}
E[X] &= E[\sum_{i=1}^{m} X_i] \\
&= \sum_{i=1}^{m} E[X_i] \\
&= \frac{m}{m} \Big[\big(1^k + (2^k - 1^k) + \ldots + (m_1^k - (m_1 - 1)^k)\big) \\
&+ \big(1^k + (2^k - 1^k) + \ldots + (m_2^k - (m_2 - 1)^k)\big) \\
&+ \ldots + \big(1^k + (2^k - 1^k) + \ldots + (m_n^k - (m_n - 1)^k)\big) \\
&= \sum_{i=1}^{n} m_i^k = F_k
\end{aligned}
$$

Upper Bounding Var$[X]$

Definition: $\mathrm{Var}[X] = E[X^2] - (E[X])^2$ So we can upper bound by $\mathrm{Var}[X] \leq E[X^2]$.

Upper Bounding Var$[X]$

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

Upper Bounding Var[X]

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

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

Upper Bounding Var[X]

We can apply this result to show that

$$
\begin{aligned}
E\left[\left(\sum_{i=1}^{m} X_i\right)^2\right] &\leq m \sum_{i=1}^{n} k m_i^{2k-1} \\
&= kmF_{2k-1} \\
&= kF_1 F_{2k-1}
\end{aligned}
$$

Upper Bounding $\mathrm{Var}[X]$

$$\begin{aligned}\mathrm{Var}[Y_i] = \mathrm{Var}[X]/s_1 &\leq E[X^2]/s_1 \\ &\leq kF_1F_{2k-1}/s_1 \\ &\leq kn^{1-1/k}F_k^2/s_1\end{aligned}$$

Error Probability

Use Chebychev's Inequality!. *Reminder:*
$s_1 = \frac{8kn^{1-1/k}}{\lambda^2}$

$$\begin{aligned}
\Pr[[Y_i - F_k] > \lambda F_k] &\le \frac{\mathrm{Var}[Y_i]}{\lambda^2 F_k^2} \\
&\le \frac{kn^{1-1/k} F_k^2}{s_1 \lambda^2 F_k^2} \\
&= \frac{kn^{1-1/k}/\lambda^2}{s_1} \\
&= \frac{1}{8}
\end{aligned}$$

Error Probability

We can apply the Chernoff Bound to show that the probability that $\frac{s_2}{2}$ of our Y_i deviate by more than λF_k from F_k is at most ϵ. In this case the median of our Y_i will provide a good estimate!.

Algorithm Space Complexity

1. There are s_2 variables Y_i.
2. Each Y_i has s_1 X_{ij}.
3. Each X_{ij} uses $O(\log(m) + \log(n))$ space.
4. Total Space Usage: $s_1 s_2 O(\log(m) + \log(n)) = O\left(\frac{k \log(1/\epsilon)}{\lambda^2} n^{1-1/k} (\log n + \log m)\right)$.
5. Time per query/update:
   $s_1 s_2 = O\left(\frac{k \log(1/\epsilon)}{\lambda^2} n^{1-1/k}\right)$.

QED

Python Magic!

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

    for cnt, a in enumerate(A):
        # since we don't know m in advance, we
        # update X and r as we go
        # when we encounter the m-th element, we update w.p. 1/m
        to_change = np.random.rand(s2, s1) < 1/(1+cnt)
        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)
```

Figure: L_p norm in action!

Error Bounds

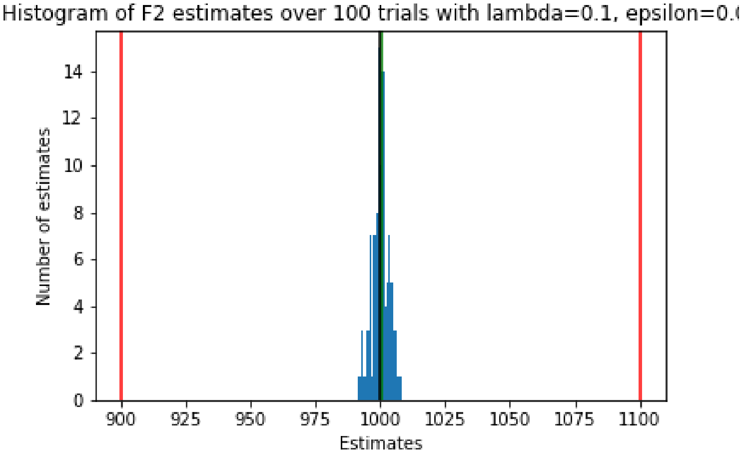

Figure: $K = 2$, $\lambda = 0.1$, $\epsilon = 0.01$

Varying K

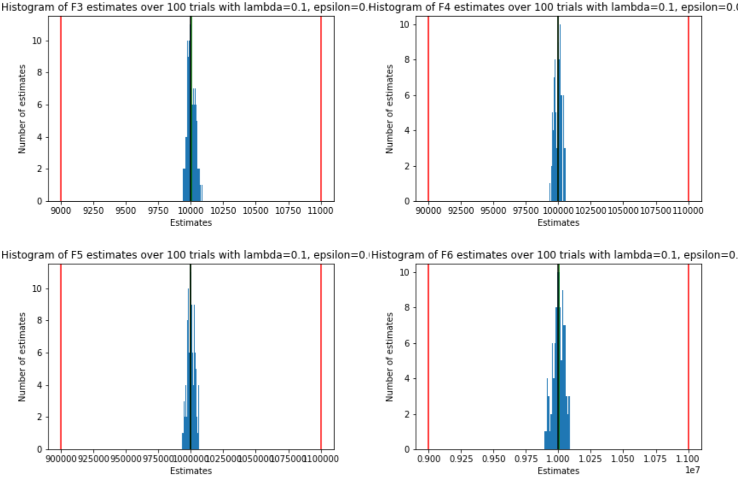

Figure: $K = 3, 4, 5, 6$ respectively, $\lambda = 0.1$, $\epsilon = 0.01$

Varying Delta

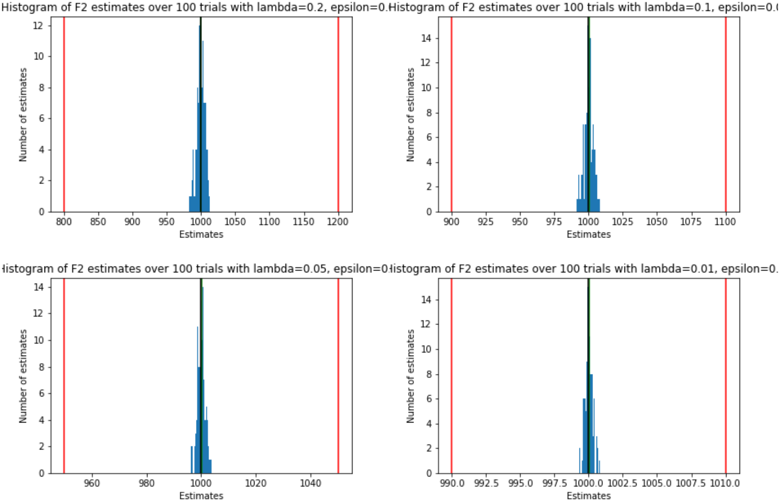

Figure: $K = 2$, $\lambda = 0.2, 0.1, 0.05, 0.01$, $\epsilon = 0.01$

Varying Epsilon

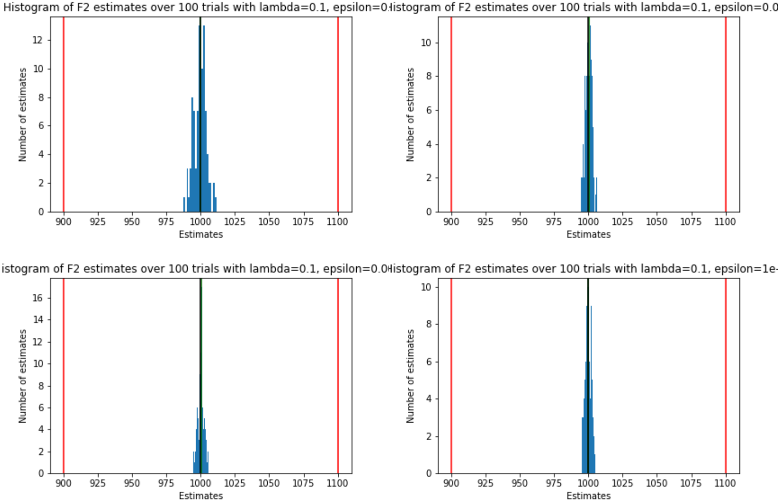

Figure: $K = 2$, $\lambda = 0.1$, $\epsilon = 1e-1, 1e-3, 1e-4, 1e-5$ respectively

Python Magic!

```
In [6]: n = 50
        m = 10000
        A = np.random.randint(n, size=(m, ))
        A

Out[6]: array([46, 11,  1, ...,  9,  2, 29])

In [35]: # A = np.arange(1000)

In [7]: uniq, cnts = np.unique(A, return_counts=True)
        k = 2
        np.sum(cnts**k)

Out[7]: 2010016

In [8]: %lprun -f lp_norm_sketch lp_norm_sketch(A, k, n, 0.1, 0.1)
```

```
Total time: 7.91206 s
File: <ipython-input-3-b3c955db71cd>
Function: lp_norm_sketch at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def lp_norm_sketch(A, k, n, _lambda, _epsilon):
     2                                               # compute the number of counters we need
     3         1         22.0     22.0      0.0      s1 = math.ceil(8*k*n**(1-1/k)/(_lambda**2))
     4         1          4.0      4.0      0.0      s2 = math.ceil(2*math.log(1/_epsilon))
     5
     6         1        293.0    293.0      0.0      X = np.zeros((s2, s1))  # the random element we care about
     7         1         34.0     34.0      0.0      r = np.zeros((s2, s1))  # stores the frequency count
     8
     9     10001      19916.0      2.0      0.3      for cnt, a in enumerate(A):
    10                                                   # since we don't know m in advance, we
    11                                                   # update X and r as we go
    12                                                   # when we encounter the m-th element, we update w.p. 1/m
    13     10000    6122209.0    612.2     77.4          to_change = np.random.rand(s2, s1) < 1/(1+cnt)
    14     10000     184538.0     18.5      2.3          X[to_change] = a
    15     10000     134687.0     13.5      1.7          r[to_change] = 0  # reset counter as needed
    16     10000    1449350.0    144.9     18.3          r[X == a] += 1
    17
    18         1       1005.0   1005.0      0.0      return np.median(np.mean((cnt+1)*(r**k - (r-1)**k), axis=1), axis=0)
```

Figure: Random numbers are expensive