CS369G: Algorithmic Techniques for Big Data Spring 2015-2016

Lecture 1: Counting Distinct Elements

Prof. Moses Charikar *Scribes: Tommy Li, Alfred Xue*

1 Overview

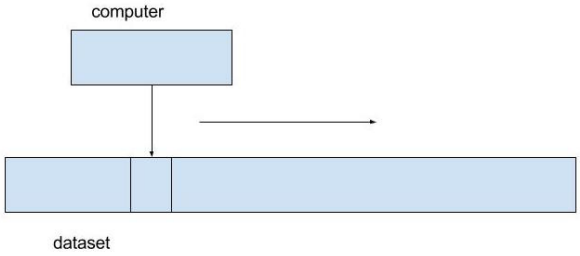
This lecture introduces the Data Stream Model and shows how hash functions and a popular technique called median-of-means may be used to develop a good estimator for the number of distinct elements in a large data set.

2 The Data Stream Model

Imagine you are working with Google's log analysis team. You want to make a query about a data set that is

- so big that it doesn't fit on a single computer, and
- so big that a polynomial running time isn't good enough.

What can you do? To satisfy these constraints, we think about algorithms that follow the Data Stream Model [1]. A computer implementing such an algorithm can only access data in a single pass, and only one piece at a time. Therefore at any given moment it only has access to its own memory and a single piece of the input data.



Here's an example question: Find a counter that uses less than $\log n$ space. This clearly can't be done perfectly, so cutting corners in some way must be necessary. We could cut corners by allowing approximation in the sense that we require that, given the true number k of elements, our counter returns a value between $k(1-\epsilon)$ and $k(1+\epsilon)$. We could also cut corners by allowing some small possibility of failure, by only requiring that our algorithm succeeds with probability $1-\delta$, where δ is some small number (e.g $\delta < 10^{-6}$). In this case, $O(\log \log n)$ bits suffice [2, 3].

3 Counting Unique Entries

Let's return to the story of being in the Google log-analysis team. Say you want to count the number of distinct queries in a list of queries. Now what do you do? One naive approach would be to use a hash table to store all each query as you receive it. However, this would require a very large hash table! The memory required would be at least linear in the number of distinct entries, which would already be too much. Another approach might be to attempt to use some subset of the queries as a representative of the entire data set, e.g. by counting the number k_m of distinct elements in some random selection of m elements of the n entries and estimating the actual number k of distinct elements as $\frac{n}{m}k_m$. It turns out this type of approach fails as well, roughly because you can't reliably gain information about things that appear extremely infrequently reliably. For example, one cannot reliably distinguish between $\underbrace{000\ldots\ldots000}_{n \text{ times}}$ and $\underbrace{000\ldots\ldots000}_{n-m \text{ times}}123...m$ using a random selection of $\frac{n}{m}$ elements. So a different approach is needed.

4 The Estimator

One idea that does work is to use a hash function $h : U \to [0, 1]$. What we do is to hash each entry x_i of the data as we see it, and keep track of the minimum seen hash value in our memory.

Claim: If there are k distinct elements $x_1, x_2, ..., x_k$ then $E\left[\min_{i=1,...,k} h(x_i)\right] = \frac{1}{k+1}$.

Proof. Let $Y = \min(h(x_1), \ldots, h(x_k))$. We will assume for now that the values $h(x_1), \ldots, h(x_k)$ are independently distributed uniform random variables over the interval $[0, 1]$. It follows that $Pr[h(x_i) \leq t] = t$ and by independence $Pr[Y > t] = (1 - t)^k$. Taking the complement of the event we get $Pr[Y \leq t] = 1-(1-k)^k$. We differentiate this cumulative distribution function with respect to t to get $Pr[Y \in [t, t + dt]] = k(1 - k)^{k-1}dt$. We can now use this to compute the expectation of Y, getting:

$$\begin{aligned}
E[Y] &= \int_0^1 tk(1-t)^{k-1}dt \\
&= k\int_0^1 (1-(1-t))(1-t)^{k-1}dt \\
&= k\left(\int_0^1 (1-t)^{k-1}dt - \int_0^1 (1-t)^k dt\right) \\
&= k\left(\frac{1}{k} - \frac{1}{k+1}\right) = \frac{1}{k+1}
\end{aligned}$$

□

This is useful because it means that Y may be used as a tool to estimate k.

5 Y alone is not good enough

While the fact that $E[Y] = \frac{1}{k+1}$ brings us closer to having a good counter, we still don't have any concrete guarantees about how well our counter will perform. How far does Y tend to stray from its mean? We compute the variance of Y as

$$\begin{aligned}
Var[Y] &= E[Y^2] - E[Y]^2 \\
&= \int_0^1 t^2 k(1-t)^{k-1} dt - \left(\frac{1}{k+1}\right)^2 \\
&= k\int_0^1 (1-(1-t))^2(1-t)^{k-1} dt - \left(\frac{1}{k+1}\right)^2 \\
&= k\left[\int_0^1 (1-t)^{k-1} - 2\int_0^1 (1-t)^k + \int_0^1 (1-t)^{k+1}\right] - \left(\frac{1}{k+1}\right)^2 \\
&= k[\frac{1}{k} - \frac{2}{k+1} + \frac{1}{k+2}] - \frac{1}{(k+1)^2} \\
&= \frac{(k+1)^2(k+2) - 2k(k+1)(k+2) + k(k+1)^2 - (k+2)}{(k+1)^2(k+2)} \\
&= \frac{k}{(k+1)^2(k+2)} \\
&\leq \frac{1}{(k+1)^2} \\
&= E[Y]^2
\end{aligned}$$

One might hope that the Chebyshev's inquality may be used to show that with Y probability, Y is "near" $E[Y]$. Unfortunately this is not the case: We get that $Pr(|Y - E[Y]| > \epsilon E[Y]) <= \frac{Var[Y]}{(\epsilon E[Y])^2} <= \frac{1}{\epsilon^2}$, which is a useless bound for small ϵ.

6 Taking a mean of estimators

Some kind of improvement is necessary. We attempt to reduce the variance by tracking multiple independent versions $Y_1, \ldots, Y_t$ of Y with corresponding hash functions $h_1, \ldots, h_t$, and use their mean $Z := \frac{Y_1 + Y_2 + \ldots Y_t}{t}$ as our new estimator.

By linearity of expectation, $E[Z] = E[Y]$. Because $Y_1, \ldots, Y_n$ are independent, $Var[Z] = \frac{1}{t^2}\sum_1^t Var[Y_i] = \frac{Var[Y]}{t} \leq \frac{E[Y]^2}{t}$. It follows that

$$Var[Z] \leq \frac{(E[Z])^2}{t}$$

This is great news, because applying Chebyshev's inequality we get

$$\Pr\left(|Z - E[Z]| \geq \epsilon \cdot E[Z]\right) \leq \frac{Var(Z)}{(\epsilon \cdot E[Z])^2} \leq \frac{1}{\epsilon^2 t}$$

This means that by increasing t, we can reduce the probability of the bad event $|Z - E[Z]| \geq \epsilon E[Z]$. In particular by setting t to $\frac{4}{\epsilon^2}$ we can bound the probability of failure by $\frac{1}{4}$.

7 Median of Means

From the discussion in the previous section it is clear that we could just keep on increasing the value of t and keep on reducing the probability of returning a value outside the desired range $[(1-\epsilon)E[Z], (1+\epsilon)E[Z]]$. But this has large space requirements and it turns out we can do much better using a popular method called *median of means*:

1. Lets $Z_1, \ldots, Z_s$ be independent copies of the estimator Z from the previous section, with each estimator Z_i corresponding to the mean of the minimum seen value of $\frac{4}{\epsilon^2}$ hash functions.
2. We return median$(Z_1, Z_2, \ldots, Z_s)$ as our estimate of the number of distinct elements.

How big does s have to be in order for median$(Z_1, Z_2, \ldots, Z_s)$ to be good (i.e. within the desired range) with at least probability $1 - \delta$? There are two ways the median can be bad: it can be too low, or it can be too high. In either case, at least half of the estimators $Z_1, \ldots, Z_s$ lie outside the desired range. For each Z_i, let A_i be an indicator variable that returns 1 if Z_i is outside the desired range, and 0 otherwise. According to the previous section's reasoning, for all i we have $E[A_i] \leq \frac{1}{4}$. The expected value of the sum $A = A_1 + \ldots + A_s$ of these indicator variables is $\leq \frac{s}{4}$.

We get

$$\begin{aligned}
\Pr[\text{median}(Z_1, Z_2, \ldots, Z_s) \text{ is bad}] &\leq \Pr[\text{at least half of } z_1, z_2, \ldots z_i \text{ are bad}] \\
&= Pr(A \geq \frac{s}{2}) \\
&\leq e^{-(2\ln 2 - 2 + 1)\frac{s}{4}} \\
&\leq e^{\frac{-s}{11}}
\end{aligned}$$

Where we applied the Chernoff bound in line three using the fact that $A_1, \ldots A_s$ were independent. This shows that in order to guarantee that $\Pr[\text{median}(Z_1, Z_2, \ldots, Z_s) \text{ is bad}] \leq \delta$, it it is sufficient to let $s = 11 \ln \frac{1}{\delta}$.

References

[1] R. Morris, Communications of the ACM, 1978

[2] P. Flajolet, G. N. Martin, Probabilistic counting algorithms for data base applications, *Journal of Computer and System Sciences*, 31(2), pp.182–209, 1985.

[3] N. Alon, Y. Matias, M. Szegedy, The space complexity of approximating the frequency moments. *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 20-29, ACM, 1996.