COMS E6998-9: Algorithms for Massive Data (Spring'19) Jan 22, 2019

Lecture 1: Introduction, Approximate Counting

Instructor: *Alex Andoni* Scribes: *Sihyun Lee*

1 Introduction

The course is about algorithms for massive data, in situations where classical algorithms (e.g. those covered in CSOR4231 Analysis of Algorithms) are not good enough because there is too much data compared to the available resources.

2 Course Topics

We introduce some algorithms covered in the course and the scenarios they can be useful.

• **Streaming Algorithms.** Consider a situation where you manage a router, to which a traffic of data is entered. Your goal is to answer queries about the data so far upon request. For example, you might be asked to answer how many different IPs you have seen so far, or to output the most frequent IP that entered your router. Such a setting, where you have to view data online and answer questions, is called **streaming**.

The context relevant to our course is when the traffic of data is much greater than the space available in the router, so that you cannot keep track of all the data at once. Many of these tasks are impossible to do exactly and with limited memory. Therefore, we consider some relaxations to the requirements:

(1) We cannot handle every single type of questions that are asked. We only consider specific questions that can be answered. For this lecture, we consider the specific problem of counting how many requests have been sent so far.

(2) We allow approximations. We say that an approximate answer is an α-approximation if:

$$(\text{true answer}) \leq (\text{approximate answer}) \leq \alpha(\text{true answer})$$

We often consider $\alpha = 1 + \epsilon$. As ϵ approaches 0, our approximation becomes more accurate.

(3) We allow randomization, and say that the approximation in **(2)** holds with a certain probability.

• **Distribution Testing and Sketching** Consider a situation where you are given independent and identically distributed (IID) samples. How can you determine if the distribution is uniform? More generally, how can you tell if two distributions are identical based on the samples? We study algorithms to find aspects of the entire data even when we have limited processing power so that we can access only a small portion of the entire data.

As another scenario, we can think of a situation where we want to find a way of comparing two data points efficiently and determine whether they have almost equal positions in the original data. This is

done by "sketching" two points in lower-dimensional space and comparing their sketched values.

• **Matrix Operations.** Consider solving the least-squares regression problem: given an $n \times d$ matrix A and an n-dimensional vector b, we want to compute a d-dimensional vector x that is the closest solution to $Ax = b$, or, in more exact terms, minimizes $\|Ax - b\|_2$.

With classical matrix algorithms, this can be done in $O(n^\omega)$ time, where the best known value of ω is between 2.3 and 2.4. However, when regression and other matrix operations are sometimes done on massive data, even an $O(n^2)$ runtime means that the algorithm is not scalable with respect to the data size. Therefore, we discuss ways to make efficient approximations on general matrices, or to compute faster answers in case the input matrices and vectors are sparse.

• **Massively Parallel Algorithms.** We know that no single machine can see all of the massive amounts of data at once. Thanks to the advancement of parallel processing, we can now improve the running time of our algorithms, and our processor can have access to all of the data at once. In this course we will discuss many algorithms that utilize parallel processing.

3 Problem: Counting

Here we give an algorithm for counting in a streaming perspective. Say that we are maintaining a router that counts the number of packets that has been received. Surely, to keep track of up to n different packets, our router should have a memory of at least $\log_2 n$ bits. We cannot do better without allowing approximation and randomization.

Morris' Algorithm ('78) is an algorithm that approximately computes n with $O(\log \log n)$ bits only. We present the algorithm and argue for its approximate correctness.

Algorithm.

1. Set a counter variable X, and initialize it to 0.
2. For each new packet, we increment X with probability $\frac{1}{2^X}$; otherwise, X remains unchanged.
3. For a query for the number of packets received, return the estimator $E = 2^X - 1$.

To continue with the analysis, we denote X_n the value of X after n increments; denote $E_n = 2^{X_n} - 1$. $X_1, ...X_n$ and $E_1, ...E_n$ are random variables.

Claim 1. $\mathbb{E}[E_n] = n$.

Proof. We use induction on n. For the base case, we have that $E_0 = 2^{X_0} - 1 = 0$.

Assume by induction that $\mathbb{E}[E_{n-1}] = n-1$. Then we can compute $\mathbb{E}_{X_1,...X_n}[E_n] = \mathbb{E}_{X_1,...X_{n-1}}[\mathbb{E}_{X_n}[E_n]]$, where the variables in the subscript indicate which variables the expectation is taken over. We proceed with the computation, but we compute $\mathbb{E}[2^{X_n}]$ for convenience:

$$\begin{aligned}
\mathbb{E}_{X_1,...X_{n-1}}[\mathbb{E}_{X_n}[E_n]] &= \mathbb{E}_{X_1,...X_{n-1}}\left[\frac{1}{2^{X_{n-1}}}2^{X_{n-1}+1} + (1 - \frac{1}{2^{X_{n-1}}})2^{X_{n-1}}\right] \\
&= \mathbb{E}_{X_1,...X_{n-1}}\left[2 + 2^{X_{n-1}} - 1\right] = 2 + \mathbb{E}[2^{X_{n-1}} - 1] \text{ (by linearity of expectation)} \\
&= n + 1 \text{ (by inductive hypothesis)}
\end{aligned}$$

Then it follows that $\mathbb{E}[E_n] = \mathbb{E}[2^{X_n} - 1] = n$. □

Showing that the expectation of E_n is n does not suffice to guarantee the performance of our algorithm. We need further knowledge about how far our random variables can deviate from its expected value, and review some useful inequalities.

Markov's Inequality. Let X be a positive random variable. Then, for any $\lambda > 0$,

$$\Pr[X \geq \lambda] \leq \frac{\mathbb{E}[X]}{\lambda}$$

For example, for any positive random variable X, we get $\Pr[X \geq 10\mathbb{E}[X]] \leq 0.1$, so with 90% confidence we can guarantee that X does not deviate from its mean by a factor of 10. Markov's Inequality shows only an upper bound for X, so we present another inequality that can be applied for both upper and lower bounds.

Chebyshev's Inequality. For any (not necessarily positive) random variable X and $\lambda > 0$,

$$\Pr[|X - \mathbb{E}[X]| \geq \lambda] \leq \frac{\mathrm{Var}[X]}{\lambda^2}$$

For example, by setting $\lambda^2 = 10\mathrm{Var}[X]$, we can find that with probability at least 90%, $X = \mathbb{E}[X] \pm \lambda$ (this is a notation saying that $X \in [\mathbb{E}[X] - \lambda, \mathbb{E}[X] + \lambda]$).

In order to apply Chebyshev's Inequality, we have to know the variance of the random variable E_n. Since E_n and 2_n^X differ only by a constant, their variances are equal, so we need only prove the following:

Claim 2. $\mathrm{Var}[2^{X_n}] \leq \frac{3}{2}n(n-1) + 1$.

Proof. We alternatively prove that $\mathbb{E}[2^{2X_n}] \leq \frac{3}{2}n(n-1) + 1$. Then, by definition of variance, we have $\mathrm{Var}[2^{X_n}] = \mathbb{E}[2^{2X_n}] - (\mathbb{E}[2^{X_n}])^2 \leq \mathbb{E}[2^{2X_n}]$ which completes the proof.

We use induction on n. For the base case, when $n = 0$, our $X_0 = 0$ all the time, so $\mathbb{E}[2^{2X_0}] = 1$ so the inequality holds tight. Now we rearrange the expressions:

$$\begin{aligned}
\mathbb{E}_{X_1,\ldots X_n}[2^{2X_n}] &= \mathbb{E}_{X_1,\ldots,X_{n-1}}[\mathbb{E}_{X_n}[2^{2X_n}]] = \mathbb{E}_{X_1,\ldots X_{n-1}}\left[\frac{1}{2^{X_{n-1}}}2^{2(X_{n-1}+1)} + (1 - \frac{1}{2^{X_{n-1}}})2^{2X_{n-1}}\right] \\
&= \mathbb{E}_{X_1,\ldots X_{n-1}}\left[2^{2X_{n-1}} + 3\frac{2^{2X_{n-1}}}{2^{X_{n-1}}}\right] = \mathbb{E}_{X_1,\ldots X_{n-1}}[2^{2X_{n-1}}] + 3\mathbb{E}_{X_1,\ldots X_{n-1}}[2^{X_{n-1}}] \\
&\leq \frac{3}{2}(n-1)(n-2) + 1 + 3n = \frac{3}{2}n(n-1) + 1
\end{aligned}$$

where the final expressions for the expectations were evaluated according to the inductive hypothesis. □

It follows that when $n \geq 10$, then $\mathrm{Var}[E_n] \leq \frac{3}{2}n(n-1) + 1 \leq \frac{3}{2}n^2$. Combining Claim 1, Claim 2, and Chebyshev's Inequality, we can conclude that $E_n = \mathbb{E}[E_n] \pm \sqrt{10\mathrm{Var}[E_n]} = n \pm 3n$ with probability at least 90%. This gives a multiplicative upper bound but a vacuously true lower bound, so we should try out something different in order to reduce the variance. We use a technique generally used in order to maintain the mean and reduce the variance, which is to run multiple experiments and take the average.

Morris+ Algorithm.

1. Run k IID copies of the basic Morris' Algorithm. Denote the X values in the ith run $X^{(i)}$.
2. Upon a query, return the estimate $E = \frac{1}{k}\sum_{i=1}^{k}(2^{X^{(i)}} - 1)$

Claim 3. $\mathbb{E}[E] = n$.

Proof. By linearity of expectation, $\mathbb{E}[E] = \mathbb{E}[\frac{1}{k}\sum_{i=1}^{k} 2^{X^{(i)}} - 1] = \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}[2^{X^{(i)}} - 1] = \frac{1}{k}\sum_{i=1}^{k} n = n$. □

Claim 4. $\mathrm{Var}[E] \leq \frac{3}{2}\frac{n^2}{k}$ *for* $n \geq 10$.

Proof. $\mathrm{Var}[E] = \mathrm{Var}[\frac{1}{k}\sum_{i=1}^{k} 2^{X^{(i)}} - 1] = \frac{1}{k^2}\mathrm{Var}[\sum_{i=1}^{k} 2^{X^{(i)}}] = \frac{1}{k^2}\sum_{i=1}^{k} \mathrm{Var}[2^{X^{(i)}}] \leq \frac{1}{k^2}\sum_{i=1}^{k} \frac{3}{2}n^2 = \frac{3}{2}\frac{n^2}{k}$. □

Now we combine Claim 3, Claim 4, and Chebyshev's Inequality to provide a performance bound.

Claim 5. *When* $k = 15\frac{1}{\epsilon^2}$*, we have a* $(1+\epsilon)$*-approximation with 90% probability.*

Proof. We see that $k = 10 \cdot \frac{3}{2}\frac{1}{\epsilon^2}$. From Claim 4, we have that $\mathrm{Var}[E] \leq \frac{1}{10}n^2\epsilon^2$. Using Chebyshev's Inequality with $\lambda^2 = 10\mathrm{Var}[E] = \epsilon^2 n^2$, we find that with probability at least 90%, $E = \mathbb{E}[E] \pm \lambda = n \pm n\epsilon = (1 \pm \epsilon)n$. □

Therefore, we can compute n within a multiplicative factor of ϵ off with 90% confidence by running Morris' Algorithm $k = O(\frac{1}{\epsilon^2})$ times.