COMP 441 — Large Scale Machine Learning	Jan 13, 2016
Lecture 2	
Lecturer: Anshumali Shrivastava	*Scribe By: Jake Kornblau*

This scribe may contain errors, please do not cite. Please email if you find any mistakes.

1 Sampling and Bounds

In the modern big data world, when the data is exceedingly large or coming in as a stream, typically and approximate estimates suffice for most practical purposes. This is because:

- approximations are good estimators of the exact answers
- exact answers are infeasible to calculate on these data sets

1.1 Bounds

Estimating answers from a random sample is one of the fundamental techniques to get approximate answers. Since the answers are not the exact, we would like to be able to bound the error on our estimates. These bounds are usually achieved using *Tail Inequalities*, also sometimes known as concentration inequalities.

1.1.1 Markov Inequality

The Markov inequality states that $Pr(x \geq a) \leq \frac{E[x]}{a}$ where $x \geq 0$

Proof:

$$\begin{aligned} aPr(x \geq a) &= a \sum_{x:x \geq a} Pr(X = x) \\ &= \sum_{x:x \geq a} aPr(X = x) \\ &\leq \sum_{x:x \geq a} xPr(X = x) \text{ as } x \geq 0 \\ &\leq \sum_{x:x \geq 0} xPr(X = x) = E[x] \text{ as additional, positive terms are added} \end{aligned}$$

One important thing to node is that this inequality is a weak bound as it does not prove that the tail of the probability distribution decreases exponentially.

1.1.2 Chebyshev's Inequality

The Chebyshev's Inequality states that $Pr(|x - \mu| \geq A) \leq \frac{Var(A)}{A^2}$ where $\mu = E[x]$
The proof sketch is as follows:, Let $y = (x - \mu)^2$, then $E[y] = Var(x)$ and $y \geq 0$. Applying the Markov Inequality to y with $a = A^2$ yields a proof of the Chebychev's Inequality
Remarks: This inequality is very useful if we have an upper bound on the variance of the random variable. Intuitively, the probability that a random variable takes a value away from its mean sharply decreases in proportion to its variance.

1.1.3 Chernoff Bounds

There are many form of Chernoff Bounds. It is a very handy inequality for showing sharp concentration of sum of many independent random variables. Formally:
Let $X_1, \ldots, X_n \overset{independent}{\sim} Bernoulli(p_i)$, $x = \sum_{i=1}^{n} X_i$ and
$\mu = E[x] = \sum_{i=1}^{n} = E[X_i] = \sum_{i=1}^{n} p_i$.
Under these conditions, Chernoff's Bounds states that for any $\epsilon > 0$,

$$Pr(X > (1+\epsilon)\mu) \leq e^{-\mu\frac{\epsilon^2}{4}} \text{ and } Pr(X < (1+\epsilon)\mu) \leq e^{-\mu\frac{\epsilon^2}{2}}$$

An alternative form has expression in terms of n (instead of ϵ) as $Pr(X - \mu > A) < e^{\frac{A^2}{2n}}$.
Remarks: Chernoff Bounds are important as it prove that for n independent events with some probability of occurrence, the probability that the sum of them deviates from the expected behavior, decreases exponentially.
Problem Solving Tips: Typically if we can write the "event of interest" as sum of many independent random variable (usually indicator random variables) then we can show that this event of interest won't deviate "much" from its expected behavior.

1.2 Generic Random Sampling for estimation

Say we wish to estimate the proportion k of elements, with some property p, in a population S with $|S| = N$. Typically, you would take a random sample $\bar{S}$ of size n and observe $\bar{k}$ in the sample. We could then estimate $\hat{k} = \frac{N}{n} \times \bar{k}$. It is a good exercise to show that this estimate is sharply concentrated if size of $\bar{S}$ is large enough.
In modern setting many times getting the random sample $\bar{S}$ itself is non-trivial.

2 Reservoir Sampling

Problem: How would one get an unbiased random sample from a stream of incoming data where we don't know the size of the population until we've seen the last element of the stream?

Formal Statement: You are given a stream of elements x_t arriving at time t. We only have storage of size k. After some time the stream ends (we don't know when). We want to get a random sample of size k from among all the elements seen, i.e. if we have seen n elements then the probability that any element x_i is in the storage buffer has probability $\frac{k}{n}$. The catch is that we do not know n beforehand, we only know k and at any time t the total elements seen so far.
Use Case in Real World: Suppose, you are monitoring a live twitter feed and you want to generate a perfectly random sample of k (size of memory you have) tweets from the total tweets seen. This sample can be used for estimation For example how many tweets are there with a particular sentiments, etc.

The solution is known as Reservoir Sampling and it works as follows.

1. Store the first k elements.
2. Every time we see the i^{th} element we select to keep it with $\frac{k}{i}$ probability and randomly knock off any one of the already chosen k elements uniformly.

It can be shown that at every point of time, we have a perfect random sample of size k, i.e. this process ensures that every element seen so far is present in the buffer with equal probability.

Proof: We will use induction to prove that we get a random sample:

- Inductive hypothesis: each item being in the buffer is $\frac{k}{m}$ where m is the number of elements seen in the stream.
- Base case: Place the first k elements of the stream into the buffer. Each element in the buffer was selected with probability $\frac{k}{m} = 1$.
- Inductive Step: Item x_{m+1} is seen and is accepted with probability $\frac{k}{m+1}$. If it is selected then one of the items currently in the buffer is removed according to a uniform distribution with $p = \frac{1}{k}$. The probability of any given item remaining in the buffer is
  $Pr(\text{it was in the buffer}) \times (1 - Pr(x_{m+1} \text{ was accepted}) Pr(\text{it was removed from the buffer}))$
  $= \frac{k}{m}(1 - \frac{k}{m+1}\frac{1}{k}) = \frac{k}{m}(1 - \frac{1}{m+1}) = \frac{k}{m+1}$
  Thus, the probability that any item is in the buffer is $\frac{k}{m+1}$.

Practical Concerns: Noticing that as
$m \to \infty, \frac{k}{m+1} \to 0, \frac{m}{m+1} \to 1$, it's apparently that as the stream continues, nothing is generally done to any individual item. A more practical solution is: At step $m+1$ as $\frac{k}{m+1} \to 0$ we can randomly choose k indices between m and $m+m+1$ and pick those elements only, skipping the rest. This will save the coin flipping cost for every element between m and $2m+1$. Although this is not exact because the exact probability $\frac{k}{m+1}$ changes as we progress, but nevertheless a good approximation.

Another Practical Method (More Suited for Distributed Implementation): another method is unrelated to the first two. Let $X_1, \ldots, X_N$ be the data elements in the stream. Additionally, let each X_i have a corresponding random number r_i where
$r_i, \ldots, r_N \overset{iid}{\sim} Uniform$. We could then generate a random sample by merely picking the k elements with the largest corresponding r values (Why ?). This is effectively done by generating a random number for each entry that come in through the stream. The first k elements of the stream are placed into a min heap (a heap where the root is the minimum element of the heap). Should the random number generated for the incoming element be greater than the root of the heap, the root is popped off of the heap and the new element is added to the heap. Since the size of the heap is k and adding a new element is $\log(k)$[1], this method is $O(n \log(k))$ in worst case. However, in practice it typically runs in $n + k\log(k)$ for $n >> k$ because we only add to heap if the selected element is bigger than the root, which is rare most of the time.

[1] Read more about "Binary heaps" in your algorithms book