0368-3248-01-Algorithms in Data Mining **Fall 2011**

Lecture 2: Sampling

Lecturer: Edo Liberty

Warning: *This note may contain typos and other inaccuracies which are usually discussed during class. Please do not cite this note as a reliable source. If you find mistakes, please inform me.*

Suppose you work for a company that runs a popular search engine. Also, suppose you are a top engineer there and your boss comes to you and asks: "Say, can you check how many people visit the second results page and then return to the first one? And while you're at it, how does it compare to last year's numbers?". Having not anticipated this question in advance, you would have been in a serious predicament was it not for that data mining class you once took... So, you made sure to keep a decently sized running uniform sample of all search interactions. You quickly compute the number of such events in the sample, multiply by the right scaling coefficient and get the answer. Your boss loves you.

1 Sampling

Given a large set of elements U, ($|U| = N$) select a subset of elements $\hat{U}$ ($|\hat{U}| \approx n$) such that from $\hat{U}$ the size of any subset $S \subset U$ can be estimated.[1] Pick each element from U independently into our set $\hat{U}$ with probability $p = n/N$. Let the variable X_i be 1 if element i is picked and 0 else. The number of picked elements is $\sum_{i=1}^{N} X_i$ and its expectation is

$$E[\sum_{i=1}^{N} X_i] = \sum_{i=1}^{N} E[X_i] = \sum_{i=1}^{N} \frac{n}{N} = n$$

Let $\hat{S}$ be the set of elements both in S and in $\hat{U}$, i.e., $\hat{S} = S \cap \hat{U}$. Equivalently, let s_i be 1 if item i is in S and zero else. Let $Z = \frac{N}{n}|\hat{S}|$ be our estimator of $|S|$:

$$E[Z] = \frac{N}{n} \sum_{i=1}^{N} E[X_i s_i] = \frac{N}{n} \sum_{j=1}^{|S|} \frac{n}{N} = |S|$$

The question is: how close is Z to $|S|$? For this we need the Chernoff bound.

Lemma 1.1 (Chernoff bound)**.** *Let $X_1, \ldots, X_n$ be independent Bernoulli trials $\Pr[X_i = 1] = p_i$. And let $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X] = \sum_{i=1}^{n} p_i$.*

$$\Pr[X > (1+\varepsilon)\mu] \leq e^{-\mu\varepsilon^2/4} \tag{1}$$

$$\Pr[X < (1-\varepsilon)\mu] \leq e^{-\mu\varepsilon^2/2} \tag{2}$$

We will prove this soon but let's first use it to solve our problem.

In our case, we go over the elements of S and count how many of them were sampled into $\hat{U}$. Since each element is taken independently with probability $p = n/N$ we have that $\mu = E[X] = |S|n/N$. Substituting into the Chernoff bound we get:

$$\Pr[X > (1+\varepsilon)|S|n/N] \leq e^{-|S|n\varepsilon^2/4N} \tag{3}$$

$$\Pr[X < (1-\varepsilon)|S|n/N] \leq e^{-|S|n\varepsilon^2/2N} \tag{4}$$

[1] It us easy to see that no deterministic algorithm can achieve this.

Or, using the union bound (below) and substituting $Z = XN/n$:

$$\Pr[|Z - |S|| > \varepsilon|S|] \leq 2e^{-|S|n\varepsilon^2/4N}$$

By demanding that $e^{-|S|n\varepsilon^2/4N} \leq \delta$ (the failure probability be less than δ) we get:

$$n \geq \frac{4}{\varepsilon^2}\frac{N}{|S|}\log(\frac{2}{\delta})$$

For example, if $|S|$ is the size of $10^{-5}N$ and we want to have a 10% accuracy with probability at least 0.99, we must keep a sample of roughly $250,000,000$ elements, regardless of N. That doesn't sound like a small number but consider that fact that this is roughly the number of search sessions per one day! (according to external reports)

2 The union bound

Lemma 2.1. *For any set of m events $A_1, \ldots, A_m$:*

$$\Pr[\cup_{i=1}^{m} A_i] \leq \sum_{i=1}^{m} \Pr A_i.$$

In words, the probability that one or more events happen is at most the sum of the individual event probabilities.

This simple notion is going to become very handy. Given the above setup, assume we want to estimate size of m different subsets on the same sampled set and assume that we are not willing to compromise on the quality. This means that we demand:

$$\forall i \quad \Pr[\left|Z^i - |S^i|\right| > \varepsilon|S^i|] \leq \delta$$

Let f_i be the event that we fail in estimating the size of S^i, i.e., $\left|Z^i - |S^i|\right| > \varepsilon|S^i|$.

$$\Pr[\cup_{i=1}^{m} f_i] \leq \sum_{i=1}^{m} \Pr[f_i] \leq m \max(\Pr[f_i])$$

Or, the probability of failure in at least one event out of m is at most m times the maximal probability of failure. By demanding that $m \max(\Pr[f_i]) \leq \delta$ we succeed in all events.

$$\max_i \; 2e^{-|S_i|n\varepsilon^2/4N} \leq \delta/m \quad \rightarrow \quad n \geq \frac{4}{\varepsilon^2}\frac{N}{\min|S_i|}\log(\frac{2m}{\delta})$$

Note that we only "pay" a logarithmic factor in m. Thus, sampling only a slightly larger set guarantees us accuracy for a very large number of different sets simultaneously.

3 Fisher Yates shuffling

Suppose you need to shuffle a very long stream of elements. How would you go about doing that? The algorithm goes as follows: go over the elements one by one. In step i generate a random number j between 0 and i and switch elements a_i and a_j (if $i = j$ then a_i remains in place). To prove this gives a truly uniform shuffle of the vector, we prove that each element is places in any location with probability exactly $1/n$. We prove this by recursion. Assume that after k steps all the elements $\{a_1, \ldots, a_k\}$ are uniformly shuffled, i.e. $\Pr[a_i \text{ is in position } j] = 1/k$ for all $i, j \leq k$. Now, advance one step in the algorithm. Element a_i is in position $j \leq k$ only if it was there in the last step of the algorithm (w.p. $1/k$) and if it was not swapped with a_{k+1}, w.p. $k/(k+1)$. Multiplying the two gives $1/(k+1)$. The element occupying the last position and the position of a_{k+1} are trivial. This completes the proof by induction.

4 Reservoir Sampling

In reservoir sampling, we want to keep exactly n elements out of a stream of N uniformly at random. The trivial thing to do is to shuffle the stream and take the first n. This is clearly correct but can we do any better?

We can simulate the Fisher-Yates algorithm and discard any element which is out of the range of $[0, \ldots, n]$. In other words, do the following: First, take the first n elements in the stream. For element a_i in the stream, generate a random number r between 0 and i. If $r \geq n$ discard the element. If $r < n$, discard element r from your collection and insert element a_i (the indices begin at zero).