Algorithms for Data Science: Lecture 4

Barna Saha

1 The Chernoff + Union Bound

Often we will need to use the Chernoff bound and the union bound together. Recall the Union Bound.
Union Bound.

$$\Pr[B_1 \cup B_2 \cup .... \cup B_n] \leq \sum_{i=1}^{n} \Pr[B_i]$$

The idea behind the Chenoff+Union bound method is as following. Using the Chernoff bound we will often be able to show that probability of a single bad event is minuscule. Hence, even if there are many such bad event, each happening with minuscule probability, using the Union bound over them we get

$$\Pr[\text{any of the bad event happens}] = \Pr[Bad_1 \cup Bad_2 \cup ... \cup Bad_{large}] \leq \sum_{i=1}^{large} \Pr[Bad_i]$$

Since for each i, $\Pr[Bad_i] = minuscule$, even when taking sum over the large number of bad events we have

$$\sum_{i=1}^{large} \Pr[Bad_i] = \sum_{i=1}^{large} minuscule = small.$$

Example 1 (Reservoir sampling gives uniform sampling)**.**

We have seen the Chernoff+Union bound in action in the previous section when we analyzed the outcome of reservoir sampling for items in $[1, 100]$ over m iterations.

There the bad event Bad_i represents the event that item i is not sampled in the range $\frac{m}{100} \pm \frac{m}{200}$. Using the Chernoff bound, for each i $\Pr[Bad_i]$ is minuscule.

Therefore, the probability that at least one of the bad event happens which will leave us unconvinced about the uniformity of reservoir sampling is at most $100 * minuscule = small$, by taking union bound over the 100 bad events.

Lets take another example, and follow the above argument rigorously.

Example 2 (Random Load Balancing[1])**.**

Suppose you are a content delivery network–say, YouTube. Suppose that in a typical five-minute time period, you get a million content requests, and each needs to be served from one of your, say, 1000 servers. How should you distribute the requests (let's call them "jobs") across your servers to balance the load? You might consider a round-robin policy, or a policy wherein you send each job to the server with the lowest load. But each of these requires maintaining some state and/or statistics, which might cause slight delays. You might instead consider the following extremely simple and lightweight policy, which is surprisingly effective: assign each job to a random server.

[1]https://www.cs.princeton.edu/courses/archive/fall09/cos521/Handouts/probabilityandcomputing.pdf

Let us use n to denote the number of jobs and k to denote the number of servers. Hence, $n = 10^6$ and $k = 10^3$.

Define an indicator random variable X_j^i which will be 1 if the job j is assigned to server i and 0 otherwise. Then $X_j = \sum_{i=1}^{n} X_j^i$ denote the load on machine i. We have

$$\mathbf{E}[X_j] = \sum_{i=1}^{n} E[X_j^i] = \frac{n}{k}$$

Then, we have by the Chernoff bound

$$\Pr\left[X_j > \frac{n}{k} + 3\sqrt{\ln k}\sqrt{\frac{n}{k}}\right] = \Pr\left[X_j > \frac{n}{k}\left(1 + \frac{3\sqrt{\ln k}}{\sqrt{\frac{n}{k}}}\right)\right]$$
$$\leq e^{-\frac{n}{3k}\frac{9\ln k}{\frac{n}{k}}} = \frac{1}{k^3}$$

Let us call the above $X_j > \frac{n}{k} + 3\sqrt{\ln k}\sqrt{\frac{n}{k}}$ as bad event Bad_j. Then,

$$\Pr\left[\text{maximum load on any machine is } > \frac{n}{k} + 3\sqrt{\ln k}\sqrt{\frac{n}{k}}\right] = \Pr\left[\text{any of the bad events happen}\right]$$
$$= \Pr\left[Bad_1 \cup Bad_2 \cup \ldots.. \cup Bad_k\right] \leq \frac{k}{k^3} = \frac{1}{k^2}$$

Hence,

$$\Pr\left[\text{the maximum load is } \leq \frac{n}{k} + 3\sqrt{\ln k}\sqrt{\frac{n}{k}}\right] \geq 1 - \frac{1}{k^2}$$

For $n = 10^6$ and $k = 10^3$, we get the maximum load is at most $10^3 + 250$ with probability $\geq 1 - \frac{1}{10^6} = 1 - 0.000001 = 0.999999$

2 Boosting by Median

Suppose there is a traffic sensor measuring the speed of vehicles passing by. It is often difficult to obtain an exact measurement, and instead the sensor returns a measurement that is noisy. Suppose, the actual speed of a vehicle is A. Imagine, that the sensor returns an estimate within $A \pm x$ with probability $= \frac{2}{3}$. So x is the error range which we are willing to tolerate. However, we are not satisfied with only a confidence parameter of $\frac{2}{3}$. Rather we would like to be confident that our obtained estimate is within the tolerated error range with probability, say ≥ 0.999.

Boosting by median is a generic technique that allows us to amplify the confidence of an event whenever that event happens with probability at least $\frac{1}{2}$.

Instead of using one traffic sensor, let us use $t = 2m + 1$ traffic sensor each reporting an independent measurement. Let X_i be the random variable denoting the measurement from the ith sensor.

Arrange the measurements in non-decreasing order, and report the median estimate.

We now calculate the probability that the median estimate is not within the error range. Let us use X_{med} to denote the median.

X_{med} is a bad estimate if either of the following two bad events happens.

(1) $X_{med} > A + x$: this event can only happen if all the m estimates with value higher than X_{med} have also value $> A + x$.

(2) $X_{med} < A - x$: this event can only happen if all the m estimates with value lower than X_{med} have also value $< A - x$.

So X_{med} can be a bad estimate if at least $m + 1$ estimates out of $2m + 1$ are out of the tolerated error range. We will bound this probability using the Chernoff bound.

Define an indicator random variable Y_i which is 1 if either $X_i > A + x$ or $X_i < A - x$ and 0 otherwise. Then $Y = \sum_{i=1}^{t} Y_i$ denote the number of sensors for which the reported value is not within the tolerated error range. We have $\mathbf{E}[Y] = t\mathbf{E}[Y_i] = t \Pr(X_i < A - x \text{ or } X_i > A + x) = \frac{t}{3}$.

Then by the Chernoff bound,

$$\Pr[Y \geq m + 1] \leq \Pr[Y \geq \frac{t}{2}] \leq \Pr\left[Y \geq \frac{t}{3}(1 + \frac{1}{2})\right] \leq e^{-\frac{t}{36}}$$

Take $t = 36 \ln 10000 \leq 332$, then

$$\Pr\left[X_{med} \notin [A \pm x]\right] \leq \Pr[Y \geq m + 1] \leq \frac{1}{10000}$$

Or,

$$\Pr\left[X_{med} \in [A \pm x]\right] \geq 1 - \frac{1}{10000} = 0.9999$$

This shows that median is a robust estimator.

3 Extensions of Reservoir Sampling

Recall the *reservoir sampling* problem that we learnt in the first class. We have to sample s elements uniformly at random from $1, 2, ..., N$ where N is unknown. The algorithm was simple.

- Maintain the first s items $a_1, a_2, .., a_s$ in the reservoir.
- For $t = s + 1, ...$
  - Sample the t element with probability $\frac{s}{t}$.
    - If the t element is sampled then
      - Select a position j in $\{1, 2, .., k\}$ uniformly at random and replace the jth element in the reservoir with the newly sampled t-th element.

Exercise 1. *What is the expected number of insertions in the reservoir?*

Drawback. The above algorithm is extremely sequential. It processes one element at a time. Can we obtain a faster distributed algorithm? For example, such an algorithm in used in *Cloudera ML*, an open-source collection of data preparation and machine learning algorithms for Hadoop.

3.1 Distributed Reservoir Sampling

For every item a_i select a random number R_i chosen uniformly at random (from the uniform distribution) from $[0, 1]$, and keep the s elements that have the largest values.

Can be implemented in the MapReduce framework (to be discussed later).

As a sequential algorithm, it has higher update time. You may need to maintain a min-heap to extract and compare with the element in the reservoir with minimum value.

3.1.1 Analysis

Lemma 1. *Consider two random variables U_1 and U_2 chosen according to uniform distribution from $[0,1]$, then $\Pr(U_1 \le U_2) = \frac{1}{2}$.*

Proof.

$$\Pr(U_1 \le U_2) = \int_{U_2=0}^{1} \int_{U_1=0}^{U_2} dU_1 dU_2 = \int_{U_2=0}^{1} U_2 dU_2 = \frac{1}{2}$$

□

Lemma 2. *Consider n random variables $U_1, U_2, ..., U_n$ chosen according to uniform distribution from $[0,1]$, then $\Pr(U_1 \le U_2 \le .... \le U_n) = \frac{1}{n!}$.*

Proof.

$$\Pr(U_1 \le U_2 \le .... \le U_n) = \int_{U_n=0}^{1} \int_{U_{n-1}=0}^{U_n} .... \int_{U_2=0}^{U_3} \int_{U_1=0}^{U_2} dU_1 dU_2 ... dU_n = \frac{1}{n!}$$

□

Therefore, if we arrange the items in non-decreasing order according to the value of the associated random variable, we get a random permutation.

Hence, the probability that an item is in the last s positions (indicating s largest values) is

$$\Pr(\text{item } a_i \text{ is in the last } s \text{ positions}) = \frac{s(n-1)!}{n!} = \frac{s}{n}.$$

3.2 Weighted Reservoir Sampling

In the weighted reservoir sample, every item in the set has an associated weight, and we want to sample such that the probability that an item is selected is proportional to its weight. Therefore, if item i has weight w_i and there are N items with N being unknown, we want that the ith item is selected with probability proportional to $\frac{w_i}{W}$ where $W = \sum_{i=1}^{N} w_i$.

The weighted reservoir sampling is based on the same idea as the distributed reservoir sampling algorithm described above. For each item i in the stream, we compute a score as follows: first, generate a random number U_i between 0 and 1 following the uniform distribution, and then take the w_ith root of U_i. Return the s items with the highest score as the sample. Items with higher weights will tend to have scores that are closer to 1, and are thus more likely to be picked than items with smaller weights.

The analysis of the algorithm is based on the following lemma.

Lemma 3. *Consider two random variables U_1 and U_2 chosen according to uniform distribution from $[0,1]$. For some $w_1, w_2 > 0$, set $X_1 = U_1^{\frac{1}{w_1}}$ and $X_2 = U_2^{\frac{1}{w_2}}$ then $\Pr(X_1 \le X_2) = \frac{w_2}{w_1+w_2}$.*

Proof.

$$\begin{aligned}\Pr(X_1 \le X_2) &= \Pr(U_1^{\frac{1}{w_1}} \le U_2^{\frac{1}{w_2}}) \\ &= \Pr(U_1 \le U_2^{\frac{w_1}{w_2}}) \\ &= \int_{U_2=0}^{1} \int_{U_1=0}^{U_2^{\frac{w_1}{w_2}}} dU_1 dU_2\end{aligned}$$

$$= \int_{U_2=0}^{1} U_2^{\frac{w_1}{w_2}} \, dU_2$$
$$= \frac{w_2}{w_1 + w_2}$$

□

Exercise 2. *Complete the proof of the weighted reservoir sampling.*