AMDM-2

Abhilash Jain

October 2018

1 Problem 1

1.1 Jaccard Co-efficient

$$J(A,B) = \frac{\|A \cap B\|}{\|A \cup B\|} \tag{1}$$

Lets take 3 examples for equation 1:

1. A case where A=B, then from the definition of$\|A \cap B\|$ and $\|A \cup B\|$ the min and max values will be the same. There-fore the Jaccard value signifying that they are the same
2. The example given in the problem, were the Jaccard value is 3/13 which also tells us that there is some similarity
3. The last one would be where there are no similar alphabets in either A or B, then the Jaccard value is 0, Which signifies that they are dissimilar

In Essence, the Jaccard Co-efficient takes into account the alphabets which are common in both the sets and their occurrences,and in the cases of extreme dissimilarity it will tend to zero.There-fore it is well motivated and meaninful

1.2 locality-sensitive hashing (LSH) scheme

To apply the Min Hash function directly, lets reformat the way we define our Sets, (the reasoning will be stated) For example,
S1={(a,2),(b,1),(d,1)}
S2={(a,1),(b,2),(c,1)} and define them like this :

Item	S1	S2
(a_1,1)	1	1
(a_2,1)	1	0
(b_1,1)	1	1
(b_2,1)	0	1
(c_1,1)	0	1
(d_1,1)	1	0

Mathematically we are basically transforming
S = { $(w_1, n)...(w_i, n)$ }, where n $\in N$ and i $\in N$ which range from 1.....i,are

the distinct words w
Will get transformed to: S' i.e (w, n)={ $(w_1, 1)....(w_n, 1)$}, where n $\in N$ which range from 1...n, are the words w present individually. Now clearly we can assess that it will be easier to define appropriate Signature Matrix's on the above data set and able to get good MinHash functions as now we represent these 'bags' as normal sets.
Lets now prove the LSH criteria that the probability that the *minhash* function for a random permutation of rows has the value for two bags is equal to the Jaccard similarity for those bags.

$$Pr[h(S_1) = h(S_2)] = J(S_1, S_2)$$

Now in this example with two bags $S_1 S_2$ we can divide the characteristic matrix of the permutation of the rows into 3 classes, namely:

- Type X rows that have 1 in both columns for S_1 and S_2
- Type Y rows that have 1 in one column and 0 in the other
- Type Z rows that have 0 in both columns

Let x, y, z be the number of X, Y and Z rows respectively
It's easy to see that the similarity between the two sets is exactly given by S=x/(x+y) which is equal to the Jaccard similarity – $x = S_1 \cap S_2$ and $x + y = S_1 \cup S_2$ (Because y is the symmetric difference between $S_1 and S_2$ i.e $y = S1 \triangle S_2$) and since encountering X type row means that$h(S_1) = h(S_2)$ and encountering Y type row gives us $h(S_1) \neq h(S_2)$There-fore the $Pr[h(S_1) = h(S_2)] = x/(x+y)$ which is exactly the Jaccard similarity.

1.3 Algorithm

The *minhash* function family can be implemented to get a *minhash* signature matrix for multiple hash function h_i. This can then act as a base through which we can pre-process all the elements of a dataset. We also divide the signature matrix that has n rows(hence n hash functions h_i) into b bands of r rows each. Now we can use another hash function $f()$ that can bin the particular column vector of length r of a specific band into many buckets. The buckets for different bands are separate. Hence only if the vectors of the signature matrix are similar in those r rows they get into the same bucket for that band.

When we are given a query we can apply the same prepossessing steps and find what all buckets does the query fall into in each band. All the elements in those bands are considered *candidate elements* The probability of collision of candidate pair is $1 - (1 - s^r)^b$.We can choose the values of b and r such that we increase the probability of more similar items leading to collisions in the buckets of a band. The *LRU* book tells us that for a threshold of s above which we compare similarity, the value of $(1/b)^{1/r}$.

Let us now define a function which pre-processes the data while bucketing and returns the Jaccard Similarity
Useful Definitions:

- X : The dataset that contains m bags as data points

- n : The number of hash-functions to use
- t : The threshold value that is considered for similarity
- q : The query data point

Algorithm 1 Similarity Test

```
1:  procedure INPUT((X,q,n,t))
2:      Compute b,r
3:      Choose n hash functions, f bucket function
4:      loop: for i in b
5:          loop: for x in X
6:              Compute colvect = [h_i1(x)....h_ir(x)]
7:              Store bucket_i [f(colvect)]+=x_i
8:      Initialize Z=0
9:      loop: for i in b
10:         Compute colvectQ = [h_i1(q)...h_ir(q)]
11:         Z = Z ∪ bucket_i[f(colvectQ)]
12:     similar = all z in Z such that JaccardSim(q,z)≥ t
```

2 Max function

Let us first define the replacement function as,

$$R_i = \begin{cases} 1 & \max \leq \text{A[i]} \\ 0 & \max > \text{A[i]} \end{cases}$$

At each step i, the value is 1 if we replace the existing max value at that point and 0 otherwise. Lets now enumerate the probabilities for replacement at each step,

$P(R_1) = 1, P(R_2) = \frac{1}{2}, P(R_3) = \frac{1}{3}, ...P(R_i) = \frac{1}{i}$

The probability that the first element is Max is 1 (obvious as Max is initially set to $-\infty$) and the probability of the second element being greater than the first (given random uniformity among the vector elements) is 1/2 and so on..
From this we can now say that

$E[R] = \sum_{i=1}^{n} \frac{1}{m} \approx \log$ n

The problem of 'priority sampling with sliding window' can be understood as finding the minimum element of window moving from right to left. This is **similar to the previous algorithm** just the fact that instead of maximum we are finding the minimum from last to first

Also the number of elements we store in our final set for priority sampling is equal to the number of times we find an element i which is the smallest compared to all others in the portion of window. (We store the elements here instead of replacing)
There-fore the space that will be taken up for the priority sampling technique is space for storing $\mathcal{O}(log\ w)$ numbers. As n is the size of the universe from which

numbers are drawn from we need $\mathcal{O}(log\ n)$ space to store any number and we need to store $\mathcal{O}(log\ w)$ numbers.

Hence the total space required by the priority sampling method is $\mathcal{O}(log\ w\ log\ n)$. Used the definitions in the slides to explain the logic.

3 Reservoir

3.1 Explaining uniformity

It means that along the data stream, at any time t, any of the "seen" values can be contained in our resulting vector with the same probability (because uniform). The only difference is that, since we have a vector of samples, the probability of one particular value persisting is higher by a factor of k(basically the dimension)

3.2 Probability

The probability p should be equal to $\frac{k}{t}$ for the $k - reservoir$ algorithm.
($1/t$ for 1 value in the vector. k/t for k values in the vector)

3.3

Divide the problem in two cases:
first case is the probability of a given sample unit being in the final output, taken after the first k elements (which will be taken with probability 1 at the beginning – the initialization) and Second case of the probability for these k elements.

CASE 1 Pr[x_i being in the final output]=P[$x_i \in R$]. With R being the final sample at time t $\geq i \geq$k

$$\begin{aligned} Pr[x_i \in R] &= \frac{k}{i} \cdot \left(1 - \frac{1}{i+1}\right) \cdot \dots \cdot \left(1 - \frac{1}{t-1}\right) \cdot \left(1 - \frac{1}{t}\right) \\ &= \frac{k}{i} \cdot \frac{i}{i+1} \cdot \frac{i+1}{i+2} \cdot \dots \cdot \frac{t-1}{t} \\ &= \frac{k}{t} \end{aligned}$$

CASE 2 We again analyze for times steps (all elements should have the probability of acceptance k/t. All the first k elements have probability 1 to be picked so now, let's see the probability of them not being replaced:

$$\begin{aligned} Pr[x_i \in R] &= 1 \cdot \left(1 - \frac{1}{k+1}\right) \cdot \left(1 - \frac{1}{k+2}\right) \dots \left(1 - \frac{1}{t-1}\right) \cdot \left(1 - \frac{1}{t}\right) \\ &= 1 \cdot \frac{k}{k+1} \cdot \frac{k+1}{k+2} \dots \frac{t-2}{t-1} \cdot \frac{t-1}{t} \\ &= \frac{k}{t} \end{aligned}$$

4 Problem 4

4.1 Application of the Algorithm

A large data set which has a small amount of good data,for example lets take binomial data(Since we are studying this is Bayesian Data Analysis), predominantly this is very useful in the Health-care field, like drug testing and Treating diseases in people

4.2 Intuition

The Main intuition behind Monte Carlo is that it is a method of computation that uses a large number of random samples to obtain results. By taking a large enough sample size N or performing the sample multiple times, we get better and better approximation of the actual ratio ρ, implying we dont have to look at the entire document

4.3 English Explanation

It basically defines the Probability of N satisfying the desired precision(ϵ approx) more precisely (1-ϵ) and (1+ϵ)

4.4 Proof

Lets first take the L.H.S

$$N \geq \frac{4}{\epsilon^2 \rho} ln \frac{2}{\delta} \tag{2}$$

$$\frac{\epsilon^2 \rho N}{4} \geq \ln \frac{2}{\delta} \tag{3}$$

$$\frac{-\epsilon^2 \rho N}{4} \leq -ln \frac{2}{\delta} \tag{4}$$

$$\frac{-\epsilon^2 \rho N}{4} \leq \ln \frac{\delta}{2} \tag{5}$$

$$\exp\left(\frac{-\epsilon^2 \rho N}{4}\right) \leq \exp\left(\ln \frac{\delta}{2}\right) \tag{6}$$

$$\exp\left(\frac{-\epsilon^2 \rho N}{4}\right) \leq \frac{\delta}{2} \tag{7}$$

$$\delta \geq 2 \cdot \exp\left(\frac{-\epsilon^2 \rho N}{4}\right) \tag{8}$$

The general form of Chernouff Bound is given by:

$$Pr[X > (1+\epsilon)\mu] \leq \exp\left(\frac{-\epsilon^2 \mu}{3}\right)$$

$$Pr[X < (1-\epsilon)\mu] \leq \exp\left(\frac{-\epsilon^2 \mu}{2}\right)$$

And we will re-write it as:

$$\begin{aligned} Pr[|Y| > (1+\epsilon)\rho N] &\leq \exp\left(\frac{-\epsilon^2\rho N}{3}\right) \\ Pr[|Y| < (1-\epsilon)\rho N] &\leq \exp\left(\frac{-\epsilon^2\rho N}{2}\right) \end{aligned} \tag{9}$$

We want the number $|Y|$ of good elements in our sample to be ϵ bounded to the actual value of good elements that would have been given us by the true ρ ratio in $|G|/(|U|)$.

we have used the fact that if
$Pr[V < a] < p_1$ and $Pr[V > b] < p_2, then Pr[V(a,b)] > 1 - p_1 - p_2$, since the intervals are exclusive, Now lets simplify equations (9) and combine them,

$$Pr[|Y| \in [(1-\epsilon)\rho N, (1+\epsilon)\rho N]] > 1 - 2\exp(\frac{-\epsilon\rho N}{4})$$

which can be written as,

$$2\exp(\frac{-\epsilon\rho N)}{4} \leq \delta$$

And the above equation is true for (8), Hence proved

4.5 Discuss Bound (2)

The general consensus is that more samples we take, the better the overall approximation of $|G|$ will be. From this equation, it's clearly seen that when $\delta, \epsilon \rightarrow$0,N$\rightarrow \infty$ in this bound when number of sample draws – considered here as experiments – goes to ∞, the expectation becomes the actual true value, but this is not possible as we have a finite set of values. But this just exemplifies our reason to use Monte-Carlo in the first place.
Also interesting to note is, For a fixed value of the error ϵ and δ we see that the number of samples N we need to take to achieve the same (ϵ, δ) approximation depends on ρ. This means that if ρ is very small then we need to a large sample to make sure that our approximation is correct. Similarly if the ratio ρ is quite high then we can take relatively smaller samples and still have the same approximation errors

5 Acknowledgement

- Wikipedia
- https://medium.com/engineering-brainly/locality-sensitive-hashing-explained-304eb39291e4
- Ananth Mahadevan
- Alexandru