Algorithmic Methods of Data Mining - Assignment 2

Maksad Donayorov

October 20, 2018

1. **Problem:** *Bag of words:*

   1.1. **Question:** *Jaccard coefficient to bags is meaningful and well-motivated:*
   Jaccard coefficient is used for comparing the similarity and diversity of sets. As we are using a finite set to represent "bag of words" the extention of Jaccard index to bag is meaningful and well-motivated.

   For example: assume that we have two documents which are not similar and represented by sets: $A = \{(x,2),(y,3)\}$ and $B = \{(x,1),(z,3)\}$. The Jaccard coefficient for these documents will be:

$$J(A,B) = \frac{\|A \cap B\|}{\|A \cup B\|} = \frac{\|\{x,x,y,y,y\} \cap \{x,z,z,z\}\|}{\|\{x,x,y,y,y\} \cup \{x,z,z,z\}\|} = \frac{\|\{x\}\|}{\|\{x,x,y,y,y,z,z,z\}\|} = \frac{1}{8} = 0.125$$

   As we can see the Jaccard index does prove that these two documents are not similar. On the contrary, if we take two exactly the same bags: $A = \{(x,2),(y,1)\}$ and $B = \{(x,2),(y,1)\}$. The the Jaccard index will be:

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

   And again we can prove by Jaccard index that these two documents are exactly the same. Consequently, we can say that the extension of the Jaccard coefficient to bags, is meaningful and well-motivated.

   1.2. **Question:** *Design a family of hash functions $\mathcal{F}$:*
   As mentioned in the part 1.1 the Jaccard index measures how similar two sets are. Therefore, we can use **min-hash** locality-sensitive hashing scheme for Jaccard coefficient to bags.

   Now coming back to the prove that min-hash function does satisfies this $Pr[f(A) = f(B)] = J(A,B)$ equality. As always, it is the best to have a good intuition and for that we can look at an example. Let's assume we have two bags:

$$A = \{(x,2),(y,2),(z,1)\}$$
$$B = \{(x,2),(z,1)\}$$

Let's now take the union of A and B.

$$C = A \cup B = \{(x,2),(y,2),(z,1)\} = \{x_0, x_1, y_2, y_3, z_4\}$$

This union shows us that all the elements that A and B have and we can use this set in our "hash table". But before that let's think of some hash function. Based on the book a good hash function is the one that has a modulo of a prime number that is bigger than the number of elements in the union set C. Since we have 5 elements in C we can choose 7 for our modulo and create some hash functions:

$$(i+1)\ mod\ 7$$
$$2(i+1)\ mod\ 7$$
$$(i+13)\ mod\ 7$$

Following these hash functions we can generate a hash table and populate it with the value of hash functions where i is the value of index:

i	A	B	$(i+1)\ mod\ 7$	$2(i+1)\ mod\ 7$	$(i+13)\ mod\ 7$
x_0	1	1	1	2	5
x_1	1	1	2	3	0
y_2	1	0	3	6	1
y_3	1	0	4	1	2
z_4	1	1	5	3	3
			h_1	h_2	h_3

The next step would be creating a signature matrix based the hash table:

$i=0$	A	B
h_1	1	1
h_2	2	2
h_3	5	5

$i=1$	A	B
h_1	1	1
h_2	2	2
h_3	0	0

$i=2$	A	B
h_1	1	1
h_2	2	2
h_3	0	0

$i=3$	A	B
h_1	1	1
h_2	1	2
h_3	0	0

$i=4$	A	B
h_1	1	1
h_2	1	2
h_3	0	0

We are interested in the signature matrix when $i = 4$, as that's the last index and it will give us the similarity of A and B.

$i=4$	A	B	similarity
h_1	1	1	1
h_2	1	2	0
h_3	0	0	1

The *similarity* column shows us that 2 out 3 hash function found A to be similar to B. Which is:

$$minHash(A,B) = \frac{2}{3} = 0.666\bar{6}$$

To compare the correctness of our calculations we can have a look at Jaccard coefficient for A and B.

$$J(A,B) = \frac{\|A \cap B\|}{\|A \cup B\|} = \frac{\|\{x,x,z\}\|}{\|\{x,x,y,y,z\}\|} = \frac{3}{5} = 0.6$$

As you can see $J(A,B) \approx minHash(A,B) \approx 0.6$. By defining more hash functions we can decrease this approximation. Consequently, we proved that the equality holds.

1.3. **Question:** *Provide pseudo-code $\mathcal{F}$:*
The algorithm is pretty straight forward:

i. for sets A, B and etc that represent documents
ii. calculated set C, such that $C = A \cup B \cup ...$
iii. shuffle set C
iv. sample from C the minimum element in permutation
v. populate signature matrix
vi. compute similarity of documents by the last signature matrix's values

```
sample = generate a sample from Union set
Row = good items of sample

for row in Row:
  for hash_func in hash_functions:
    compute hash_func(row)
    for column in S:
      if hash_func(row) < signature(S,i):
        sig_matrix(i) = hash_func(row)

// query
quer = new_doc
for row in length(quer):
  for hash in hash_functions:
    hash = compute hash_func(row)
    if hash < signature(S,i):
      sig_matrix(i) = hash

similarity = compare(new_doc, sig_matrix)
```

2. **Problem:** $\mathcal{O}(\log w \ \log n)$:
Before analyzing the complexity and responding to the question it could be nice to see an algorithm that finds the *max* number in an array. Such algorithm can be easily written in python, that gives us a clear picture of how the computation is done:

```python
def get_max(arr):
  max_num = -1
  for el in arr:
    if max_num < el:
      max_num = el
  return max_num
```

The best case for finding the *max* element is when the first element is bigger than all of the other elements of X, which has complexity of $\mathcal{O}(1)$. And the worst case is when the last element is the largest and all the consecutive numbers proceed from small to large. This case has a complexity of $\mathcal{O}(n)$. The tricky part is the average case, where we have to compute the complexity for *the number of times that max is assigned to an element.*

Assuming that $X[1, 2, 3...m]$ is drawn independently and uniformly at random from the interval $(0, 1)$, the expected value will be:

$$E[x] = \sum_{i=1}^{m} Pr(X_i)$$

Where $Pr(X_i)$ is the probability of the *i-th* element being the *max* element in $X[1, 2, 3...m]$. To have a better intuition of what could be the probability of *i-th* element we can have a look at different size of m:

$$\begin{aligned} Pr(x_1) &= 1 \\ Pr(x_2) &= \frac{1}{2} \\ Pr(x_3) &= \frac{1}{3} \\ &... \\ Pr(x_m) &= \frac{1}{m} \end{aligned}$$

Looking at this we can infer that the expected value can be calculated as:

$$E[x] = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{m} = \sum_{i=1}^{m} \frac{1}{i} \approx \log m$$

Consequently, we can conclude that $max = X[i]$ will be executed $\mathcal{O}(\log m)$ times, on expectation.

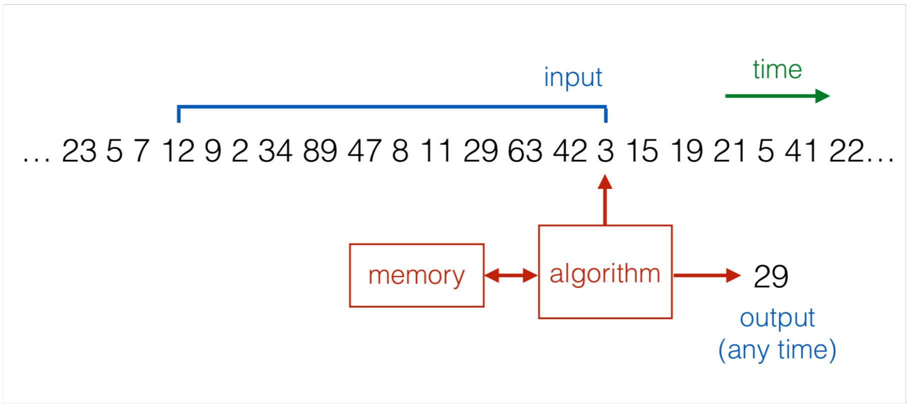

Figure 1: Representation of sliding window: *"CS-E4600 - slide set 7"*.

Using the same logic and intuition we can argue that the *priority sampling for sliding window* (shown in figure 1), will use $\mathcal{O}(\log w \log n)$, where $\log w$ is the length of the sliding window and $\log n$ is the space for numbers to be stored. The arguments for our prove are the criteria that:

- in any given window each item has equal chance to be selected as a random sample
- each removed element has a larger element proceeding it
- at any given point we expect the space efficiency $\mathcal{O}(w)$ with maximal elements

- and finally, maintaining list of maximal elements requires $\mathcal{O}(\log w)$ time

3. **Problem:** *Reservoir algorithm for sampling 1 element in a data stream:*

   3.1. **Question**: *explain k-sample in a data stream is uniform:*
   The meaning of $k-sample$ in a data stream being uniform is: for randomly choosing a sample of k items from a data stream, at any given time each element of the data stream have equal probability of being sampled.

   3.2. **Question**: *value of the probability p:*
   The value of the probability will be the length of $k-sample$ divided by the value of how many elements at current time we have seen, t. Which is $\frac{k}{t}$.

   3.3. **Question**: *prove that value of p gives uniform samples:*
   Let's suppose that our $k-sample$ has 5 elements. When sixth element arrives $i=6$ and $t=[1,2...6]$, each element is kept with the probability $\frac{5}{6}$, which is:

$$(1)(\frac{1}{6}+(\frac{5}{6})(\frac{4}{5}))=\frac{1}{6}+\frac{4}{6}=\frac{5}{6}$$

   when the seventh element arrives $i=7$ and $t=[1,2...7]$, the seventh element is kept with the probability 5/7 and each of the previous 6 elements are also kept with the same probability. Following that logic, we can prove by induction that when there are n elements, each one is kept with the probability $5/n$. Consequently, our general notation will be:

$$P(k_{sample}|t_{elements\ seen})=\frac{k}{t}$$

4. **Problem:** *Resort to sampling of good items:*

   4.1. **Question**: *a real-world application for the algorithm:*
   Imagine an insurance company that operates in the US, Europe and Asia and has about 2 billion customers. By the end of every month it wants to check an estimated number of sick leaves reported by each customer for the whole period of 30 days. A traditional way would be run through all 2 billion people and check if they have reported sick leave or not for a period of 30 days. That could be computationally expensive and time consuming. Instead they could use this algorithm to take a subset of customers ($X \subseteq U$) for each day and compute how many sick leaves were reported for i-th day, then sum all the 30 days to get an estimated number of sick leaves reported for the past one month.

   4.2. **Question**: *explain the intuition of Monte Carlo algorithm:*
   Referring to the example brought in 4.1 we can explain the Monte Carlo algorithm with some assumed data. So, we have a set of 2 billion people and in the first step the Monte Carlo algorithm takes a random subset $X \subseteq U$, of size $|X| = N$:

$$U = \{all\ 2\ bilion\ customers\}$$
$$X = \{a\ random\ subset\ of\ U\ with\ both\ good\ and\ bad\ items\}$$
$$N = length\ of\ X,\ let's\ say\ it\ is\ 20,000$$

Next it takes only good items from the set X:

$$Y = \{subset\ of\ X\ who\ reported\ sick\ leave,\ "good\ items"\} = \|Y\| = eg : 900$$

And only after that it calculates the the estimated "good items":

$$Z = \frac{|U||Y|}{N} = \frac{900 * 2,000,000,000}{20000}$$
$$= 90,000,000\ sick\ leaves\ reported\ per\ one\ day$$

For 30 days we can do the same calculations 30 times and sum all up.
As you can see the intuition is pretty straight forward once you have a real world use case for the algorithm.

4.3. **Question**: *explain the equation in English:*
The equation:

$$Pr[(1-\epsilon)|G| \leq Z \leq (1+\epsilon)|G|] \geq 1-\delta$$

has several "segments" and by explaining each part we can clarify the general picture, which allows us to explicitly state what the equation stands for:

$|G|$ - is the number good items of a universe U
Z - estimate or approximation of $|G|$
ϵ - an error
δ - a probability that Z does not lie within an interval

We then move on to the bigger parts:

$(1-\epsilon)|G|$ - is a lower bound for Z
$(1+\epsilon)|G|$ - is an upper bound for Z

Putting all these together, the equation implies that the probability that Z lies between two bounds is greater than or equal to $1-\delta$.

4.4. **Question**: *use the Chernoff bound to show the Monte Carlo algorithm returns an estimate Z:*

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

Let's start by replacing the Z with the value $\frac{|U||Y|}{N}$:

$$Pr[(1-\epsilon)|G| \leq Z \leq (1+\epsilon)|G|] \geq 1-\delta =>$$
$$Pr[(1-\epsilon)|G| \leq \frac{|U||Y|}{N} \leq (1+\epsilon)|G|] \geq 1-\delta$$

As we know the value of ρ:

$$\rho = \frac{|G|}{|U|}\ which\ is\ |G| = \rho|U|$$

Replacing the value of $|G|$ in our equation we get:

$$Pr[(1-\epsilon)\rho|U| \leq \frac{|U||Y|}{N} \leq (1+\epsilon)\rho|U|] \geq 1-\delta$$

Multiplying this equality with $\frac{N}{|U|}$ we get:

$$Pr[(1-\epsilon)\rho|U| \times \frac{N}{|U|} \leq \frac{|U||Y|}{N} \times \frac{N}{|U|} \leq (1+\epsilon)\rho|U| \times \frac{N}{|U|}] \geq 1-\delta$$

We get:

$$Pr[(1-\epsilon)\rho N \leq |Y| \leq (1+\epsilon)\rho N] \geq 1-\delta \tag{1}$$

Before moving forward it is good time to state the Chernoff bound general notation:

$$Pr[X \geq (1+\delta)\mu] \leq e^{-\frac{\delta^2\mu}{3}}$$
$$Pr[X \leq (1-\delta)\mu] \leq e^{-\frac{\delta^2\mu}{2}}$$

Let's now assume that $X = Y$, $\mu = \rho N$ and $1-\delta$:

$$Pr[Y \geq (1+\epsilon)\rho N] \leq e^{-\frac{\epsilon^2\rho N}{3}}$$
$$Pr[Y \leq (1-\epsilon)\rho N] \leq e^{-\frac{\epsilon^2\rho N}{2}}$$

Now, let's come back to the equation (1) and combine it with the Chernoff bound:

$$Pr[(1-\epsilon)\rho N \leq |Y| \leq (1+\epsilon)\rho N] \geq 1 - e^{-\frac{\epsilon^2\rho N}{3}} - e^{-\frac{\epsilon^2\rho N}{2}} \tag{2}$$

Since

$$e^{-\frac{\epsilon^2\rho N}{3}} < e^{-\frac{\epsilon^2\rho N}{4}}$$
$$and$$
$$e^{-\frac{\epsilon^2\rho N}{2}} < e^{-\frac{\epsilon^2\rho N}{4}}$$

We can rewrite the equation in (2) like:

$$\begin{aligned}
1 - e^{-\frac{\epsilon^2\rho N}{4}} - e^{-\frac{\epsilon^2\rho N}{4}} &\leq 1-\delta \leq 1 \\
2e^{-\frac{\epsilon^2\rho N}{4}} &\leq \delta \\
\frac{2}{\delta} &\leq e^{-\frac{\epsilon^2\rho N}{4}} \\
taking\ ln\ from\ both\ sides\ results\ in: \\
ln\frac{2}{\delta} &\leq \frac{\epsilon^2\rho N}{4} \\
which\ is\ equivalent\ to:\ N &\geq \frac{4}{\epsilon^2\rho} ln\frac{2}{\delta}
\end{aligned}$$

4.5. **Question**: *discuss the bound:*
If we increase the value of N then the error rate decreases. If ϵ and δ are constant and ρ is **small** then N size should be big. On the other hand if ϵ and δ are constant and ρ is **big** then N size should be small. If ρ is constant then we have these conditions:

1. When ϵ is constant and δ changes:

ϵ	δ	$\frac{4}{\epsilon^2 \rho} ln \frac{2}{\delta}$
0.2	0.1	2995.73
0.2	0.05	3688.87
0.2	0.01	5298.31
0.2	0.001	7600.90

2. When δ is constant and ϵ changes:

ϵ	δ	$\frac{4}{\epsilon^2 \rho} ln \frac{2}{\delta}$
0.2	0.1	2995.7
0.3	0.1	1331.4
0.4	0.1	748.9
0.5	0.1	479.3