Algorithmic Methods of Data Mining - Assignment 2

Maksad Donayorov

October 19, 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. The algorithm is pretty straight forward:

   i. for sets A and B that represent two documents

   ii. calculated set C, such that $C = A \cup B$

   iii. shuffle set C and sample from it minimum element in permutation

   iv. repeat this in a cycle

   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.

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:

```
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:

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

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.

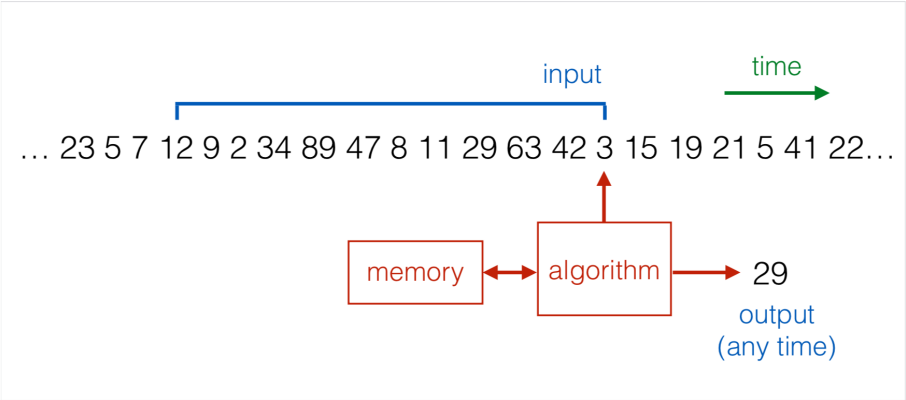

Figure 1: Representation of sliding window: *"CS-E4600 - slide set 7"*.

Using the same logic and intuition we can argue that the *priority sampling for sliding window* 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:*