CS-E4600
Algorithmic Methods of Data Mining
Answers to Homework 2

Adam Ilyas 72581

November 7, 2018

Lukas Haug, Bianca Lachennmann , Francsico Caldas, Zeyneb Erdogan

Problem 1

A bag is an unordered list of tuples (x, n) where x is elements in the document and n is the frequency of that element

Useful Operations

Bag size Bag size A: $||A|| = \sum_{(x,n)\in A} n$

Bag union $A \cup B = \{(x, \max\{n, m\})$ where $(x, n) \in A$ and $(x, m) \in B\}$.

Bag Intersection $A \cap B = \{(x, \min\{n, m\})$ where $(x, n) \in A$ and $(x, m) \in B\}$.

Jaccard Coefficient between bags $J(A, B) = \frac{Intersect}{Union} = \frac{||A\cap B||}{||A\cup B||}$

Question 1.1 Argue that the extension of the Jaccard coefficient to bags, as defined above, is meaningful and well-motivated.

Ans. The jaccard coefficient $J(A, B)$ is a measure of similarity

$$0 \leq J(A, B) \leq 1$$

If the 2 documents contains the same elements with the same frequency, then the $J(A, B) = 1$ because the $||A \cup B|| = ||A \cap B||$

However, if the documents do not contain any elements in common, then $J(A, B) = 0$ since an intersection will not exist $||A \cap B|| = 0$, because we are taking the minimum frequency of the element $\min(n, m)$.

The more words that the 2 bags have in common, the higher the value of the intersect $||A \cap B||$. Hence, the value of the coefficient will increase. Hence this is a good measure for similarity where $J(A, B)$ cannot be more than 1 or less than 0, and is a value in between.

Question 1.2 Provide a locality-sensitive hashing (LSH) scheme for the Jaccard coefficient to bags. In other words, design a family of hash functions $\mathbb{F}$ such that

$$Pr[f(A) = f(B)] = J(A, B)$$

when f is drawn uniformly at random from $\mathbb{F}$.

Ans We represent each occurence of an element in the bag as a distinct element. For example, we convert a bag

$$A : \{(a, 2), (b, 3), (c, 1)\} \to \{a_1, a_2, b_1, b_2, b_3, c_1\}$$

$$B : \{(a, 3), (b, 1), (d, 2)\} \to \{a_1, a_2, a_3, b_1, d_1, d_2\}$$

To compare 2 bags in the manner, we convert both of them the form as above and express in a matrix which is just column A and B of the following table:

row number	x_i	A	B
0	a_1	1	1
1	a_2	1	1
2	a_3	0	1
3	b_1	1	1
4	b_2	1	0
5	b_3	1	0
6	c_1	1	0
7	d_1	0	1
8	d_2	0	1

Our hash function (h) will 'shuffle' the rows number by mapping the row number to another integer.

For each set (A and B), It will then iterate through the rows, in the increasing order of the new row number, until it reachs the first (1). It will then return the new row number contains the first (1)

We assume no collisions, then the $Pr(h(A) == h(B))$ is simply the (number of rows with both '1's) over (the total number of rows). This value is the intersection $A \cap B$, where both A and B has this value, divided by the union (total number of distinct item). This probability is equal to the jaccard coefficient.

Number of rows which both elements are 1: $||A \cap B||$,
Total number of rows: $||A \cup B||$

We denote the hash function we defined above as h. Thus

$$\Pr\left[h(A) = h(B)\right] = \frac{||A \cap B||}{||A \cup B||}$$

which is the Jaccard coefficient

Question 1.3. Discuss how exactly you will implement the locality-sensitive hashing scheme you designed. Provide pseudocode for finding similar documents in a document collection given a query document, where documents are represented as bags of words. Your pseudocode should describe both the preprocessing and the querying part of the similarity-search algorithm.

Preprocessing We want to convert our bags to sets:
bagUnion $A \cup B = \{(x, \max\{n, m\})$ where $(x, n) \in A$ and $(x, m) \in B\}$
unionSet $\{x_i$ for i in [1, 2 .. m] for (x,m) in bagUnion$\}$
setA $\{x_i$ for i in [1, 2 .. m] for (x,m) in bag A$\}$
setB $\{x_i$ for i in [1, 2 .. m] for (x,m) in bag B$\}$

```
1: unionSize ← size of unionSet
2: Initialize 2 arrays of zeroes $S_A, S_B$ with size unionSize
3: for index from [1, 2 .. unionSize] do
4:     if unionSet[index] in setA then
5:         $S_A[index] \leftarrow 1$
6:     if unionSet[index] in setB then
7:         $S_B[index] \leftarrow 1$
```

Previously, our h_i function will 'shuffle' the row numbers then return the minimum row number that has (1).

However in our algorithm, it will do the same but the hash function f will only shuffle the row number. The selection of the minimum row number will

be accounted for in our Algorithm.
We have a *minhash* function family of f_i where each f 'shuffles' the rows differently by having a different map for the row number to another integer. We can choose n hash functions $f1, f2, \dots f_n$ which can be implemented to get a *minhash* signature matrix.
First we initialize our signature matrix of n rows by 2 cols (for both our 2 sets)

$$\text{M} = \begin{array}{c|c|c} \text{f} & S_A & S_B \\ \hline f_1 & \infty & \infty \\ f_2 & \infty & \infty \\ \vdots & \vdots & \vdots \\ f_n & \infty & \infty \end{array}$$

We recall our initial characteristic matrix and pre compute the new row number based on the hashing functions

row number	x_i	A	B	f_1	f_2	$\cdots$
0	a_1	1	1	3	7	$\cdots$
1	a_2	1	1	4	8	$\cdots$
2	a_3	0	1	5	0	$\cdots$
3	b_1	1	1	6	1	$\cdots$
4	b_2	1	0	7	2	$\cdots$
5	b_3	1	0	8	3	$\cdots$
6	c_1	1	0	0	4	$\cdots$
7	d_1	0	1	1	5	$\cdots$
8	d_2	0	1	2	6	$\cdots$

```
Input $S_A, S_B$
 1: for row from [1, 2 .. unionSize] do
 2:     if $S_A[row] == 1$ then
 3:         for i from 1 to N do
 4:             $M[row, 0] \leftarrow \min(M[row, 0], f_i(row))$
 5:     if $S_B[row] == 1$ then
 6:         for i from 1 to n do
 7:             $M[row, 1] \leftarrow \min(M[row, 1], f_i(row))$
 8: initalize int $same \leftarrow 0$;
 9: for i from 1 to n do
10:     if $S_A[i] == S_B[i]$ then
11:         $same \leftarrow same + 1$
12: return (same/n)
```

The intuition is that we shuffle the characteristic matrix n times. And for each time, we get the first row number that has (1). We then end up with a n by 1 vector for each set, each element is the first row number that has (1).

To compare the similarity between A and B, we just take the number of rows with the same value for S_A and S_B and divide this number with n our similarity measure, which is equivilent to the jaccard coefficient.

Problem 2

Consider an array X of m numbers, where each number $X[i]$ is drawn independently and uniform at random from the interval (0, 1). Consider a simple linear algorithm to compute the maximum number in X : start with $max = \infty$ and scan the array; each time you encounter a number $X[i]$ that is larger than the current value of max update its value with $max = X[i]$. Show that the update step (max $= X[i]$) will be executed $\mathcal{O}(\log m)$ times, on expectaton.

Ans. For the current max to be updated to $X[i]$, $X[i]$ has to be bigger than the previous $i-1$ number ($X[i]$ to $X[i-1]$)

If we have i elements, the probaility that the the i^{th} element is the biggest value is $\frac{1}{i}$ since all the values are uniformly distributed. Hence the probability of the max value being updated to the i^{th} value is also $\frac{1}{i}$

$$\begin{aligned}
\text{Pr(max update: max } \leftarrow X[1]) &= 1 \\
\text{Pr(max update: max } \leftarrow X[2]) &= \frac{1}{2} \\
\text{Pr(max update: max } \leftarrow X[3]) &= \frac{1}{3} \\
&\vdots \\
\text{Pr(max update: max } \leftarrow X[m]) &= \frac{1}{m} \\
\text{Expected max updates: } \sum_{i=1}^{m} \frac{1}{i} &\approx \log m
\end{aligned}$$

Use this result to argue that the priority sampling for sliding window discussed in class requires space $\mathcal{O}(\log w \log n)$. Recall that w is the length of the sliding window, and n the size of the universe where numbers are drawn from.

Ans. We need $\log w$ space to maintain the number of minimal items in the sliding window.

Encodings are the random values $v \in (0, 1)$ which we assign to the elements of the window. We then maintain all the elements in the sliding window in which their assigned value v is smaller than all the subsequent values.

The space we need for these encodings is $(\log(n))$ space. Also, we need to find all minimum encodings for every instance of a sliding window. For every window, we need $\mathcal{O}(log(w))$ space to find the minimum probability.

Thus, we need $\mathcal{O}(log(n)log(w))$ space for this Algorithm.

Problem 3

In class we discussed the reservoir algorithm for sampling 1 element in a data stream. The algorithm has the property that it obtains a uniform sample. Uniform here means the following: Consider the algorithm at any time t. Up to that point the algorithm has seen t items in total. Any of these t items has equal probability to be the sampled item.

In this problem we want to make a simple extension of the reservoir algorithm to sample k items. Again we want to obtain a uniform sample. We are thinking that the following algorithm is a meaningful extension of the 1-sample reservoir algorithm.

Algorithm: k-RESERVOIR

```
1: Initialize array S[1...k]
2: $t \leftarrow 1$
3: while (1) do
4:     read input $X_i$
5:     if $t \leq k$ then
6:         $S[t] \leftarrow x_t$
7:     else
8:         with probability p
9:         $j \leftarrow random[1..k]$
10:        $S[j] \leftarrow x_t$
11:    $t \leftarrow t + 1$
```

Question 3.1. Write in English what does it mean that a k-sample in a data stream is uniform.

Ans Each element of the data stream has an equal probabulity to be chosen to be an element of the sample, and we want to choose up to k-items from the data steam to be our sample

Question 3.2. What should the value of the probability p be for the k-RESERVOIR algorithm to give uniform samples?

Ans Probability $p = \frac{k}{t}$

Question 3.3. Prove that for the value of p that you specified in 3.2 the k-RESERVOIR algorithm gives indeed uniform samples.

Consider 2 cases:
Case 1: $t \leq k$
When there are k items or fewer in S, each element of S is kept with probability 1
Case 2: $t > k$
When $t = k+1$, each of the old and new items have a probability of $\frac{k}{k+1}$ of being in the sample
The new item is chosen with the probability: $\frac{k}{k+1}$ because we set the probability of keep a new item as $\frac{k}{t}$

Probability that any one of previous elements to remain in the reservoir is:

$$\Pr(\text{choose other (k-1) elements}) \times \Pr((k+1)^{th} \text{ element is selected}) +$$

$$\Pr((k+1)^{th}\text{element is not selected}) =$$

$$\frac{k-1}{k} \times \frac{k}{k+1} + \frac{1}{k+1} = \frac{k}{k+1}$$

When $t = k+2$, the $(k+2)^{th}$ item has prob. of $\frac{k}{k+2}$ of being kept into the reservoir.

Recall that the previous $k+1$ elements each has the prob. of $\frac{k}{k+1}$ of "surviving $t = k+1$". The probability that each of them are kept in this round $(t = k+2)$ is

$$\Pr(\text{other items is replaced})\Pr(\text{new item is kept}) + \Pr(\text{new item is not kept}) =$$

$$\frac{k-1}{k}\frac{k}{k+2} + \frac{2}{k+2} = \frac{k+1}{k+2}$$

Thus, the probability that each of the $k+1$ element be kept in $t = k+2$ is:

$$\Pr(\text{kept in previous rounds})\Pr(\text{kept in current round})$$

$$\frac{k}{k+1}\frac{k+1}{k+2} = \frac{k}{k+2}$$

Thus, for a general $t = i$, the probability that each of the previous elements are kept up to and during $t = i$ is:

$$\Pr(t = k) \cdot \Pr(t = k+1) \cdot \Pr(t = k+2) \ldots \Pr(t = i-1) \cdot \Pr(t = i)$$

$$1 \cdot \frac{k}{k+1} \cdot \frac{k+1}{k+2} \cdots \frac{i-2}{i-1} \cdot \frac{i-1}{i} = \frac{k}{i}$$

Hence, for the t^{th} element, where $t \geq k$, we set $p = \frac{k}{t}$ to obtain a uniform k-sample

Problem 4

We consider a set of items U. The size of U is potentially very large. A subset G of U has a property of interest. We say that G are the good items of U. We want to estimate the number of good items $|G|$.

We assume that the fraction of good items is $\rho = |G|/|U|$.

We also assume that given an item of U we can easily check whether it is a good item. The challenge is that as U is very large we cannot enumerate all its items. So we are thinking to resort to sampling. We consider the following MONTE CARLO sampling algorithm, which returns a variable Z as an estimate of $|G|$.

Algorithm: MONTECARLO

```
1: Sample a random subset $X \subseteq U$, of size $|X| = N$
2: Let $Y \subseteq X$ be the items in $X$ that are good
3: Return $Z = \frac{|U||Y|}{N}$
```

Question 4.1. Describe a real-world application that you may consider applying the above algorithm.

When you have to check the quality of items in a factory but you do not want to check for every single item, the above Algorithm gives a good estimate of checking the overall quality of the items by sampling just a subset of all the items

Question 4.2. Explain the intuition of MONTE CARLO algorithm.

Based on the question, fraction of good items is $\rho = \frac{|G|}{|U|}$.

If we take a selected item is "good" is a success probability p,

$$\Pr(x \text{ is good}) = p, \quad \Pr(x \text{ is bad}) = 1 - p, \quad x \in U$$

Then each item $x \in U$ follows the distribution bernoulli(p).

Since X is selecting N items with bernoulli(p), $X \sim \mathit{Binomial}(N, p)$ with means $\mu = Np = N\frac{|G|}{|U|}$, $\mu\frac{|U|}{N} = |G|$

From this equation, we can obtain $Z = |Y|\frac{|U|}{N}$ where Z is a good approximation to $|G|$ because the $|Y|$ that we generate has $E[|Y|] \approx \mu$.

An important question is how large the sample size N should be so as to obtain a good approximation or $|G|$. We are interested in (ϵ, δ) approximations. In particular, given numbers $\epsilon, \delta > 0$, we say that Z is an (ϵ, δ)-approximation of $|G|$ if the following is true:

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

Question 4.3. Explain Equation (1) (the above equation) in English.

It shows the probability that the value generated Z is close to the real value $|G|$ with a relative deviation of value ϵ.This probability is at least $1-\delta$

When ϵ increases, so does the probability $(1-\delta)$ as it increases the range which Z can deviate by.

Question 4.4. Use the Chernoff bound to show that if

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

then the MONTECARLO algorithm returns an estimate Z that is an (ϵ, δ)-approximation of $|G|$.

Our chernoff bounds where $0 < \epsilon < 2e - 1$:

$$\Pr(Z \geq (1+\epsilon)|Q|) \leq e^{-\frac{\epsilon^2 \mu}{4}}, \quad \Pr(Z \leq (1-\epsilon)|Q|) \leq e^{-\frac{\epsilon^2 \mu}{4}}$$

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

We then know the probability in terms of e

$$e^{-}\frac{e^2\mu}{4} + e^{-}\frac{e^2\mu}{4} \leq \delta$$
$$2e^{-}\frac{e^2\mu}{4} \leq \delta$$
$$\ln(2e^{-}\frac{e^2\mu}{4}) \leq \ln(\delta)$$
$$-\frac{e^2\mu}{4} \leq \frac{\delta}{2}$$
$$\frac{e^2\mu}{4} \geq \frac{\delta}{2}$$

We can introduce N, ρ into the equation

$$\begin{aligned}\frac{e^2\mu}{4} &\geq \ln(\frac{\delta}{2}) \\ N\rho &\geq \frac{4}{\epsilon^2}\ln(\frac{\delta}{2}) \\ N &\geq \frac{4}{\epsilon^2\rho}\ln(\frac{\delta}{2})\end{aligned}$$

Question 4.5. Discuss the bound (2) in terms of the dependence of N with respect to the approximation parameters ϵ and δ and the problem parameters $|U|$ and ρ. What are the implications of the bound (2)?

Implications For both parameters $|U|$ and ρ:

> If we increase either, then it will lead to N (number of items in X) to be smaller
>
> If we decrease either, then it will lead to N to be greater

The parameters do not depend on N. N depends on these parameters actually.