Big data computing - Homework 3

Giulia Muscarà 1743261

January 10, 2021

Assignment 1

Consider Locality Sensitive Hashing for the estimation of Jaccard similarity between sets. It is assumed the use of several independent permutations (hash functions) to produce the signature of each set. These hash functions are defined as $h_\pi(C) = min_\pi \pi(C)$, where $min_\pi \pi(C)$ is the index of the first row in the permuted order π in which column C has value 1. It is also assumed that the banding technique with b bands of r rows each is used and candidate column pairs are those that hash to the same bucket for at least one band. The goal is to compute the probability that the signatures of two sets $C1$ and $C2$ and with Jaccard similarity $t = sim(C1, C2) = \frac{|C1 \cap C2|}{|C1 \cup C2|}$ are identical in at least one band.
To start with, it can be proved that the probability that the minhash signatures of C1, C2 agree in one row of a band is $s = Pr[h_\pi(C1) = h_\pi(C2)] = sim(C1, C2)$, according to the Min-Hash property [1]. Assume π is a random permutation, X is a set of shingles and y $\in$ X is a shingle. It is equally likely that any y $\in$ X is mapped to the minimum element, so $Pr[\pi(y) = min(\pi(X))] = \frac{1}{|X|}$. Let y be such that $\pi(y) = min(\pi(C1 \cup C2))$. Then either $\pi(y) = min(\pi(C1))$ if $y \in C1$, or $\pi(y) = min(\pi(C2))$ if $y \in C2$.
So the probability that both are true is the probability that $y \in C1 \cap C2$.
As a consequence, $Pr[min(\pi(C1)) = min(\pi(C2))] = Pr[h_\pi(C1) = h_\pi(C2)] = \frac{|C1 \cap C2|}{|C1 \cup C2|} = sim(C1, C2)$.
After having defined s, it follows that the probability the signatures agree in all rows of one particular band is s^r, as in one band there are r rows and this probability for each row is independent from the others.
The probability that the signatures disagree in at least one row of a particular band is $1 - s^r$, as it is the complement of the previous probability.
The probability that the signatures disagree in at least one row of each of the bands in the signature matrix is $(1 - s^r)^b$, as the matrix is divided into b bands and this probability for each band is independent from the others.
Hence, the probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - (1 - s^r)^b$.

Assignment 2

It is assumed that a set of points S in R^d is represented using locality sensitive hashing. LSH reduces the set of d-dimensional points to a signature matrix M of dimension $m \times n$, where n is the number of points in S and m is the length of each signature. For cosine similarity, the matrix M is obtained by exploiting random vectors in R^d. For each point $x \in S$, m random vectors are chosen. The dot product between the point and each random vector is computed and one value of the signature of the point is ouput at a time, returning 1 if the result is positive or -1 otherwise. The result will be positive if the angle between the point and the random vector is acute. Otherwise, if the angle between them is obtuse it will be negative. These hash functions are chosen from the locality-sensitive family H for cosine distance and each hash function of the family is built from a randomly chosen vector. As a consequence, the signature of each point will be made of 1 and -1 values.
After computing the signature matrix, the banding technique is then applied to M, by dividing it into b bands, each one made of r rows such that $m = b \times r$. For each band b, a different hash table is built.
A hash function h is then chosen to map the partial signature of each point $x \in S$ of each band b to a bucket of the hash table built for b. In this way, all the hash tables are populated.
Given a new point $p \in R^d$, its signature can be computed as well by exploiting random vectors. Then, the approximated k nearest neighbors can be retrieved by taking the most similar k candidates that hashed to the same buckets of p, exploiting the hash tables and computing their cosine distances. This procedure is showed in the following pseudocode.

```
new_signature[m] = empty list
for v in random_vectors do                ▷ random_vectors is the list of m random vectors chosen in R^d
    if dot_product(v, p) ≥ 0 then
        new_signature[v] = 1
    else
        new_signature[v] = −1
    end if
end for
```

After computing the signature of p, its partial signatures can be hashed to the existing hash tables.

candidates = empty set of candidates which are in the same buckets where p was hashed to
for band in b **do**
bucket_index = h(new_signature[band])
bucket = set of points in the bucket with index bucket_index in the corresponding hash table
candidates.union(bucket)
bucket.add(p)
end for
candidates = list(candidates)
candidates = candidates.map(x, (x, cosine_similarity(c, p))).orderByDescending(x → x[1])
return candidates[:k]

All the points hashed to the same buckets as p can be considered similar to it beacuse of the LSH algorithm. For each candidate, the cosine similarity between it and p is computed so as to identify the k neighbours with highest similarity to p. This is why the algorithm is linear with respect to the number of similar points or sublinear to the number of total points in S.

Assignment 3

1. Assume that $x \in R^d$ and $s \in R^d$, two d-dimensional vectors, and that $F(x) = s^T x$. While x can be any vector, it is assumed that s is a random variable vector with entries drawn independently from the others from {1, -1} with uniform probability. This means that the probabilities that entries in s assume one of the two possible values, denoted $P(s_i = 1)$ and $P(s_i = -1)$, $\forall i$ in $\{1...d\}$, are both equal to $\frac{1}{2}$. As a consequence, the expected value of s is $E[s] = \frac{1}{2} * 1 + \frac{1}{2} * -1 = 0$.
Under these assumptions, $E[F(x)^2] = ||x||^2$ and it can be proved as follows.

2. $E[F(x)^2] = E[(s^T x)^2] = E[(\sum_{i=1}^d s_i x_i)^2] = E[\sum_{i=1}^d \sum_{j=1}^d x_i x_j s_i s_j] = E[\sum_{i=1}^d x_i^2 + \sum_{i=1}^d \sum_{j \neq i}^d x_i x_j s_i s_j]$
Indeed, when $i = j$ $x_i * x_j = x_i^2$ and $s_i * s_j = s_i^2 = 1$.
The expectation of the first sum on x is the sum itself, because x is not a random variable by definition. In the second part of the expression, instead, for linearity of expectation the expectation sign can be moved inside of the sum operators and also $x_i x_j$ can be moved outside the expectation as they are not random variables.

$E[F(x)^2] = \sum_{i=1}^d x_i^2 + E[\sum_{i=1}^d \sum_{j \neq i}^d x_i x_j s_i s_j] =$
$= \sum_{i=1}^d x_i^2 + \sum_{i=1}^d \sum_{j \neq i}^d E[x_i x_j s_i s_j] =$
$= \sum_{i=1}^d x_i^2 + \sum_{i=1}^d \sum_{j \neq i}^d x_i x_j E[s_i s_j] =$
$= \sum_{i=1}^d x_i^2 + \sum_{i=1}^d \sum_{j \neq i}^d x_i x_j E[s_i] E[s_j] =$
$= \sum_{i=1}^d x_i^2 = ||x||^2$

Indeed, $\sum_{i=1}^d \sum_{j \neq i}^d x_i x_j E[s_i] E[s_j] = 0$ under the assumption that s_i and s_j are independent and that $E[s_i] = E[s_j] = 0$, which was previously proved. Finally, $\sum_{i=1}^d x_i^2$ equals the euclidean norm of x squared.

Assignment 4

Reservoir sampling algorithm allows to choose a sample of s items from a data stream of unknown size n at any given moment. The size of the sample is fixed a priori and whenever the sample may be extracted, it should maintain the desired size over the part of the stream seen so far.
It is assumed that sample S is required to be of size s and that at time n, n items were observed. After observing n items, each is in the sample S with equal probability $\frac{s}{n}$. By induction this claim can be proved to be true whenever the stream produces a new item as well.
The base step for induction is trivially true for the first s data points. Indeed, it can be assumed that after observing the first s elements, they are all in the sample. When $n = s+1$ and the $s+1-th$ item is produced, it could be either discarded or inserted in the sample at the expense of another item already in S. The probability that the new item is inserted in S is $\frac{s}{s+1} = \frac{s}{n}$. As a consequence, the probability for every item x already in S to stay in the sample is:

$P(x \in S \text{ at time n}) = 1 - P(x \text{ is discarded}) = 1 - \frac{s}{s+1} * \frac{1}{s} = 1 - \frac{1}{s+1} = \frac{s}{s+1} = \frac{s}{n}$

The probability that x is discarded in favour of the new item is given by the product of the probability that the new item is taken and the probability of being chosen among the other s elements, as they are independent events. For the induction step, it can be assumed that the following claim is true also after observing n elements.

$\forall x \in 1, ..., n \; P(x \in S^{(n)}) = \frac{s}{n}$

This claim holds for $n + 1$ items observed as well: the sample S contains each element seen so far with probability $s/(n + 1)$, which can be proved as follows.

$\forall x \in 1, ..., n \;\; P(x \in S^{(n+1)}) = P(x \in S^{(n+1)} \wedge x \in S^{(n)}) = P(x \in S^{(n+1)} | x \in S^{(n)}) * P(x \in S^{(n)}) =$
$= [(1 - \frac{s}{n+1}) + (\frac{s}{n+1})(\frac{s-1}{s})] * \frac{s}{n} = \frac{n}{n+1} * \frac{s}{n} = \frac{s}{n+1}$

Indeed, x can stay in the new sample for two mutually exclusive reasons. In the first case the new item is discarded with probability $1 - \frac{s}{n+1}$. Otherwise, the $n+1-ith$ item is accepted and x is kept all the same, with probability $\frac{s}{n+1} * (1 - \frac{1}{s})$.
The reservoir sampling algorithm follows, basing on the proved property, where D is the data stream, S the sample composed of s items and n is the number of items observed at a given point in time. To start with, the sample is initialized with the first s elements of the stream. Then for every new item in the stream in position i, the index of an item among all those observed so far is chosen uniformly at random through the function $random(1, i)$, returning a random integer in the range between 1 and i. If the integer is less then or equal to s, the element in position i in the stream will take the place of the element in the sample that is in the position returned by the function. Otherwise, if i is greater than s no old item will be discarded and the sample will stay the same.

```
for i = 1 to s do
    S[i] = D[i]
end for
for i = s + 1 to n do
    item_to_discard = random(1, i)
    if item_to_discard ≤ s then
        S[item_to_discard] = D[i]
    end if
end for
```

Assignment 5

It is assumed that S is a stream of items from a discrete universe of size n and that the same item can appear multiple times in S. Consider a hash function chosen from the min-wise independent family. The family of functions H is min-wise independent if for any set $X \subseteq [n]$ and any $x \in X$, when h is chosen at random in H, $Pr(min\{h(X)\} = h(x)) = \frac{1}{|X|}$ [2].
This means that if h is a min-wise independent hash function, the probability that any of the elements in S is mapped to the minimum hash value is uniform. Also, even if there may be duplicates in S, h will hash equal elements in the stream to the same hash value. Hence, if d distinct elements were observed so far in S, exactly d different hash values would be produced.
At first, when there is only one element in the stream, its hash value will be the minimum one with probability 1, being the only hash value generated so far. When a second element arrives, it could either be a duplicate of the first one or a new element. If it is a duplicate, it will be hashed to the same hash value of the first element, so there will be only one hash value elegible to become the minimum one. As a consequence, it will still become the minimum with probability 1. Otherwise, if the second element in the stream is not a duplicate but a new digit, it will be hashed to a different value with respect to the first element. In this case, two different hash values will be eligible to become the minimum and the probability that either of them is chosen is uniform, so $\frac{1}{2}$. The same reasoning holds when other elements arrive. Then, the probability for any of these hash values to be the minimum among the others would be $\frac{1}{d}$, given that the set of all the generated hash values has not size $|S|$, but d.
Storing only the element that hashes to the minimum value into the sample and updating it whenever it is

needed, the condition that each of the distinct items observed so far has the same probability of being the sampled item is satisfied. The pseudocode of the algorithm follows.

```
min_hash = h(S[0])        ▷ min_hash is initialized with the hash of the first element in the stream
sample = S[0]             ▷ sample is initialized with the first element in the stream
for s in S do
    hash_val = h(s)
    if hash_val < min_hash then
        min_hash = hash_val
        sample = s
    end if
end for
```

For example, if S is $2, 1, 2, 4, 1, 9$, hashing each number we would get only four different hash values because 2 and its duplicate will hash to the same value, just as 1 and its duplicate. For the definition of min-wise independent hash function, for the set $X = \{2, 1, 4, 9\} \subseteq [n]$ and for any $x \in X$, $Pr(min\{h(X)\} = h(x)) = \frac{1}{|X|}$, where $|X| = 4$, the number of distinct elements observed in the stream so far. If another element arrives, for example 8, its hash value will be different from the others, so the probability will become $\frac{1}{|X|}$, where $|X| = 5$.

References

[1] Leskovec, J., *Mining Massive Datasets*, Stanford University, 2011.

[2] Broder, A., et al., *Min-wise independent permutations*, Journal of Computer and System Sciences, 2000.