Big Data Analytics: HOMEWORK 1
Professor: Mohamed Ndaoud
DUE ON: FRIDAY, NOVEMBER 11.

> The homework can be done in groups of 4 students. Please type your solutions (LaTeX and code) on a Jupyter Notebook that you will upload on Moodle. Make sure to write the names of the 4 students of your group in the top of your notebook. It is enough that one students uploads the work of the whole group.
> In case you struggle writing Latex on a Jupyter notebook, you can return a (single) pdf file that includes both the executed code and your solutions to the theoretical questions.

Problem 1, 30 points: (On the Jaccard distance)

Necessary condition:

We wish to show that for the Jaccard similarity $\mathrm{sim}(x,y)$ to possess a locality-sensitive hashing scheme, the function $d(x,y) = 1 - \mathrm{sim}(x,y)$ must satisfy the triangle inequality:

$$d(x,y) + d(y,z) \geq d(x,z) \tag{1}$$

for all x, y and z.

We begin with a probabilistic interpretation of the function $d(x,y)$:

$$\begin{aligned} d(x,y) &= 1 - \mathrm{sim}(x,y) \\ &= 1 - \mathbb{P}\left[h(x) = h(y)\right] \\ &= \mathbb{P}\left[h(x) \neq h(y)\right]. \end{aligned} \tag{2}$$

We thus see that Eq. 1 is equivalent to the following statements:

$$\mathbb{P}[h(x) \neq h(y)] + \mathbb{P}[h(y) \neq h(z)] \geq \mathbb{P}[h(x) \neq h(z)] \tag{3}$$

$$\mathbb{P}(A) + \mathbb{P}(B) \geq \mathbb{P}(C) \tag{4}$$

where we have defined the events A, B and C as:

$$A = [h(x) \neq h(y)], \qquad B = [h(y) \neq h(z)], \qquad C = [h(x) \neq h(z)]. \tag{5}$$

Let us enumerate the total probability using the events A, B and C:

A	B	C	$\mathbb{P}$
0	0	0	$p_0 = \mathbb{P}(\bar{A} \cap \bar{B} \cap \bar{C})$
0	0	1	$p_1 = \mathbb{P}(\bar{A} \cap \bar{B} \cap C)$
0	1	0	$p_2 = \mathbb{P}(\bar{A} \cap B \cap \bar{C})$
0	1	1	$p_3 = \mathbb{P}(\bar{A} \cap B \cap C)$
1	0	0	$p_4 = \mathbb{P}(A \cap \bar{B} \cap \bar{C})$
1	0	1	$p_5 = \mathbb{P}(A \cap \bar{B} \cap C)$
1	1	0	$p_6 = \mathbb{P}(A \cap B \cap \bar{C})$
1	1	1	$p_7 = \mathbb{P}(A \cap B \cap C)$

1. Compute the following probabilities: p_1, p_2 and p_4.

2. Deduce $\mathbb{P}(A), \mathbb{P}(B)$ and $\mathbb{P}(C)$.

3. Conclude.

Solution:

1. Note that the probabilities p_1, p_2 and p_4 are identically zero, since their associated event (the appropriate intersection of A, B and C) is the null event. For instance, the event $\bar{A} \cap \bar{B} \cap C$ requires simultaneously $h(x) = h(y)$, $h(y) \neq h(z)$ and $h(x) = h(z)$ which is clearly impossible. It comes out that $p_1 = p_2 = p_4 = 0$.

2. From the table, we then find:

$$\begin{aligned} \mathbb{P}(A) &= p_4 + p_5 + p_6 + p_7 = p_5 + p_6 + p_7 \\ \mathbb{P}(B) &= p_2 + p_3 + p_6 + p_7 = p_3 + p_6 + p_7 \\ \mathbb{P}(C) &= p_1 + p_3 + p_5 + p_7 = p_3 + p_5 + p_7 \end{aligned}$$

3. Based on question 2 we now have that

$$\mathbb{P}(C) = p_3 + p_5 + p_7$$

and

$$\mathbb{P}(A) + \mathbb{P}(B) = p_3 + p_5 + 2p_6 + 2p_7.$$

The desired inequality of Eq. 4 immediately follows.

Problem 2, 60 points: (LSH for approximate near neighbor search - ANN)

The "trick" to this problem is simply to be careful with notation, and to recall the algebraic properties of the logarithm function. The basic facts are as follows:

- The dataset $\mathcal{A}$ is a set of n points in a metric space with distance measure d.
  - Define $z \in \mathcal{A}$ to be a specified "query point".
  - Assuming there exists a point $x \in \mathcal{A}$ such that $d(x, z) \leq \lambda$, our goal is to retrieve a point $x' \in \mathcal{A}$ with $d(x', z) \leq c\lambda$.
- The family $\mathcal{G}$, formed of hash functions of $\mathcal{H}$, is such that for any $g \in \mathcal{G}$
  - $\mathbb{P}(g(x) = g(z)) = \frac{1}{n}$ for any x such that $d(x, z) > c\lambda$.
  - $\mathbb{P}(g(x) = g(z)) = \frac{1}{n^\rho}$ for any x such that $d(x, z) \leq \lambda$ for some $\rho < 1$.
- Finally, we take $L = n^\rho$ random hash functions $g_1, g_2, \cdots, g_L$ of $\mathcal{G}$, and hash all the points of $\mathcal{A}$ using all g_i's.

An upper-bound on false positives:

Let

- $W_j = \{x \in \mathcal{A} \mid g_j(x) = g_j(z)\}$, *i.e.* the subset of $\mathcal{A}$ that map under g_j to the same bucket as $g_j(z)$.
- $T = \{x \in \mathcal{A} \mid d(x, z) > c\lambda\}$, *i.e.* the set of points that do not match our search criterion. Ideally, the elements of T should not share any hash bucket with z.

1. Show that

$$\mathbb{P}\left[\sum_{j=1}^{L} |T \cap W_j| > 3L\right] < \frac{1}{3} \tag{6}$$

i.e. the probability that we have more than $3L$ false positives in our LSH scheme (the elements of T that did in fact map to one of the buckets $g_i(z)$) is at most 1/3.

An upper-bound on false negatives:

Let $x^* \in \mathcal{A}$ be a point such that $d(x^*, z) \le \lambda$.

1. Show that

$$\mathbb{P}\left[g_j(x^*) \neq g_j(z)\ (\forall 1 \le j \le L)\right] < \frac{1}{e}. \tag{7}$$

(c, λ)-ANN has contant probability of success:

In the (c, λ)-ANN algorithm, we retrieve at most $3L$ data points from the buckets $g_j(z)$, $(1 \le j \le L)$ and report the closest one as a (c, λ)-ANN.

1. Estimate the probability of error of the algorithm and conclude.

Solution:

1. Consider the random variable

$$\mathbb{E}\left(\sum_{j=1}^{L} |T \cap W_j|\right) = L \cdot \mathbb{E}\left(|T \cap W_1|\right) \tag{8}$$

where we have used the symmetry of the problem with respect to the g_j's.

Now, let us examine the random variable $|T \cap W_1|$. Let $x \in T$ be fixed. Since $\mathbb{P}(g(x) = g(z)) = \frac{1}{n}$ for any x such that $d(x, z) > c\lambda$, the probability that $x \in W_1$ is at most $1/n$. It then follows that $|T \cap W_1|$ is a binomial distribution with a mean that is bounded as: $\mathbb{E}(|T \cap W_1|) = np < n \cdot 1/n = 1$. Using this result, Eq. 8 yields that $\mathbb{E}(\sum_{j=1}^{L} |T \cap W_j|) < L$.

We then apply the Markov inequality to get Eq. 6

$$\mathbb{P}\left[\sum_{j=1}^{L} |T \cap W_j| > 3L\right] < \mathbb{P}\left[\sum_{j=1}^{L} |T \cap W_j| > 3 \cdot \mathbb{E}\left(\sum_{j=1}^{L} |T \cap W_j|\right)\right] < \frac{1}{3}. \qquad \square$$

Observe that we could have used Hoeffding inequality to improve the previous bound.

2. Since $\mathbb{P}(g(x) = g(z)) = \frac{1}{n^\rho}$ for any x such that $d(x, z) \le \lambda$ for some $\rho < 1$, the probability that $g_j(x^*) = g_j(z)$ for any particular j is at least $\frac{1}{n^\rho}$. Equivalently, the probability that $g_j(x^*) \neq g_j(z)$ for any particular j is at most $1 - \frac{1}{n^\rho}$; and it follows that the probability that $g_j(x^*) \neq g_j(z)$ for all j is at most $\left(1 - \frac{1}{n^\rho}\right)^L$.

Using the definition of $L = n^\rho$, we have

$$\mathbb{P}\left[g_j(x^*) \neq g_j(z)\ (\forall 1 \le j \le L)\right] \le \left(1 - \frac{1}{L}\right)^L < \frac{1}{e}.$$

In the last line, we have used the limit definition of the exponential function (*i.e.* $e^x = \lim_{m\to\infty}\left(1+\frac{x}{m}\right)^m$) and the fact that the expansion for finite $m \geq 1$ is a lower bound for the series limit. □

3. Note that we assume that there is at least one point $x^* \in \mathcal{A}$ with $d(x^*, z) \leq \lambda$. Clearly, if there are more points that satisfy the distance criterion, the probability of success will be higher.

   The algorithm can fail in two ways:

   - The element x^* is hashed to an incorrect bucket, *i.e.* $g_j(x^*) \neq g_j(z)$ for all j. This event was considered in the previous question. The probability of this "false negative" event is bounded at e^{-1}.
   - The element x^* is hashed to a correct bucket, but too many false positives (more than $3L$) from the set $\sum_{j=1}^{L} |T \cap W_j|$ leads x^* not to be considered in our first $3L$ candidates. This "false positives" scenario was considered in the first question.

   The corresponding probability of failure is

   $$\mathbb{P}_{\text{fail}} < e^{-1} + \left(1 - e^{-1}\right) \cdot \frac{1}{3} \approx 0.58,$$

   and hence the reported point is an actual (c, λ)-ANN with constant probability. □

Problem 3, 40 points: (A similarity-matching function)

The goal of this excercise is to implement your own similarity-matching function based on Jaccard similarity and minimum hash values.

- You will start by writing a simple similarity-matching function minhash(input_question, compare_question) that computes the similarity between two input strings. This function will take two input string parameters and returns its Jaccard similarity:

  1. Split text into elements: write a function shingles to convert the input text into elements of three characters.
  2. Calculate Jaccard distance: create a function that accepts as imputis two sets and compute its Jaccard distance.
  3. Test out the results by running the following code: print(shingles("This function works perfectly"))
  4. Use 1 and 3 to write the minhash function. Put the main code inside a try except call.

- Try out your function on some simple examples, e.g., "I have a cat", "I have a dog"

- Case sensitivity: this minhash function is case sensitive. To avoid it, lowercase before calling the minhash function.

- Take a small text of few strings. Slightly modify it and compute the similarities between two texts. Add further modifications to the text and compute the similarities again. What do you observe?

Problem 4, 70 points: (Searching via MinHash and Locality Sensitive Hashing)

In this problem the data contains pairs of questions with labels indicating if they are duplicates or not. The goal is to identify if the pair of questions are duplicates. You can find the train set "train.csv" on Moodle. More precisely, the goal is to compare only the questions that are "similar" in order to save computational resources. We will do it using MinHash and Locality Sensitive Hashing.

1. You may need the folowing packges:
   import numpy as np # linear algebra
   import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
   from tqdm import tqdm # make your loops show a smart progress meter
   import nltk # Natural Language Toolkit
   import datasketch # Probabilistic data structures for processing and searching very large datasets

2. Extract the data from "train.csv" to the DataFrame qa_pairs and take a look on it.

3. Create a random sample of questions from qa_pairs. For example you can use the following code:

   sents_pairs = pd.concat([qa_pairs[qa_pairs['is_duplicate'] == 0].sample(100, random_state=42), qa_pairs[qa_pairs['is_duplicate'] == 1].sample(100, random_state=42)]).reset_index(drop=True)
   sents_pairs = sents_pairs.sample(frac=1.)
   sents = pd.concat([sents_pairs['question1'], sents_pairs['question2']])

4. Represent the questions as single word tokens if they are not stop words:
   - Download 'stopwords' from nltk package
   - Create 'set_dict' dictionary which maps question id (eg 'm23') to set representation of question.
   - Loop through each question, convert them into shingles, and, if the shingle isn't a stop word, add it to a hashset which will be the value for the set_dict dictionary.
   - Do not froget to lowcase!
   - Additionally create 'norm_dict' dictionnary which maps question id (eg 'm23') to actual question (we may use it to evaluate the result).

5. Create minHash signatures:
   - Fix the number of permutations for the MinHash algorithm 'num_perm'.
   - Create 'min_dict' which maps question id (eg 'm23') to min hash signatures. You can use 'MinHash' from 'datasketch' package: `http://ekzhu.com/datasketch/minhash.html`
   - Loop through all the set representations of questions and calculate the signatures and store them in the 'min_dict' dictionary.

6. LSH can be used with MinHash to achieve sub-linear query cost. Create LSH index using 'MinHashLSH' from 'datasketch' package: `http://ekzhu.com/datasketch/lsh.html`

   - Set the Jaccard similarity threshold (e.g. =0.4) as a parameter in MinHashLSH.
   - Loop through the signatures or keys in the 'min_dict' dictionary and store them. Datasketch stores these in a dictionary format, where the key is a question and the values are all the questions deemed similar based on the threshold.

7. Giving the MinHash of the query set, retrieve the keys (m1, m2 etc.) that references sets with approximate Jaccard similarities using the following code:

   ```
   big_list = []
   for query in min_dict.keys():
   big_list.append(lsh.query(min_dict[query]))
   ```

8. Check some of the resulting pairs.