Homework 4

CSCI 4907/6444: Big Data and Analytics

Fall 2023

The deadline for homework assignment 4 is **12/2/2023 11:59 pm**. The total number of points for this assignment is 10. Make sure that you upload a *single* zip file for all of the questions in the assignment on blackboard. Create two folders `q1` and `q2` and put all the related content (code, output files, writeups, etc.) in the corresponding folder. Do not include the input data in your submissions. Name you folder `lastname_firstname_hw4` and compress it to create a zip file. Upload your zip file to blackboard. **Do NOT clear your Jupyter notebook cell outputs that contain your results.**

Academic Integrity

Honor Code: Students may discuss the solutions at a high level with one another. Note that each student must write down their solutions/code independently to show they understand the solution well enough in order to reconstruct it by themselves. Students **should clearly mention the names of all the other students who were part of their discussion group and specify each person's contribution.**

Important: Using code or solutions obtained from the web is considered an honor code violation. You may use printed or online sources for understanding how to use Python functions but you may not search for code or theoretical solutions. We check all the submissions for plagiarism. Please include the following text excerpt in your submission zip file (in the main folder, next to folders `q1` and `q2`):

```
I [insert your name here], affirm that this is my own work,
I attributed where I used the work of others, I did not fa-
cilitate academic dishonesty for myself or others, and I used
only authorized resources for this assignment, per the GW Code
of Academic Integrity.  If I failed to comply with this sta-
tement, I understand consequences will follow my actions.  Con-
sequences may range from receiving a zero on this assignment
to expulsion from the university and may include a transcript
notation.
```

1 Learning Embeddings (3 points)

In this problem, we want to explore learning embeddings with Singular Value Decomposition. We have a corpus of 10 words and a set of 15 documents from a Conservation Zoo. We can represent the documents and the corpus of words in matrix form as:

	zoo	kangaroo	monkey	alligator	tiger	camel	eagle	lemur	dragon	pizza
doc1	51	92	14	71	60	20	82	86	74	74
doc2	87	99	23	2	21	52	1	87	29	37
doc3	1	63	59	20	32	75	57	21	88	48
doc4	90	58	41	91	59	79	14	61	61	46
doc5	61	50	54	63	2	50	6	20	72	38
doc6	17	3	88	59	13	8	89	52	1	83
doc7	91	59	70	43	7	46	34	77	80	35
doc8	49	3	1	5	53	3	53	92	62	17
doc9	89	43	33	73	61	99	13	94	47	14
doc10	71	77	86	61	39	84	79	81	52	23
doc11	25	88	59	40	28	14	44	64	88	70
doc12	8	87	0	7	87	62	10	80	7	34
doc13	34	32	4	40	27	6	72	71	11	33
doc14	32	47	22	61	87	36	98	43	85	90
doc15	34	64	98	46	77	2	0	4	89	13

You will first decompose the matrix above. You are allowed to use any SVD solver. We recommend using numpy.linalg.svd(). Feel free to use the below code to recreate the matrix above in memory:

```
import numpy as np
np.random.seed(42)
x = np.random.randint(0,100, size=(15,10))
```

(a) [1.5 points]

Task: Using SVD and r = 9, compute the embedding for all 15 documents. Round to the nearest 2nd decimal.

(b) [1.5 points]

Task: Using SVD and r = 9, compute the embedding for all 10 words. Round to the nearest 2nd decimal.

What to submit

Submit the following:

1. Values from [part (a)]
2. Values from [part (b)]
3. Submit your code for both parts a and b.

2 Data Streams (7 points)

In this problem, we study an approach to approximating the frequency of occurrences of different items in a data stream. Assume S = ⟨a1, a2, . . . , at⟩ is a data stream of items from the set {1,2,...,n}. Assume for any $1 \leq i \leq n$, F[i] is the number of times i has appeared in S. We would like to have good approximations of the values F[i] ($1 \leq i \leq n$) at all times.

A simple way to do this is to just keep the counts for each item $1 \leq i \leq n$ separately. However, this will require O(n) space, and in many applications (e.g., think online advertising and counts of user's clicks on ads) this can be prohibitively large. We see in this problem that it is possible to approximate these counts using a much smaller amount of space. To do so, we consider the algorithm explained below.

Strategy The algorithm has two parameters $\delta, \epsilon > 0$. It picks $\lceil \log \frac{1}{\delta} \rceil$ independent hash functions:

$$\forall j \in \left[1; \lceil \log \frac{1}{\delta} \rceil\right], \quad h_j : \{1, 2, .., n\} \rightarrow \{1, 2, .., \lceil \frac{e}{\epsilon} \rceil\},$$

where log denotes natural logarithm. Also, it associates a count $c_{j,x}$ to any $1 \leq j \leq \lceil \log \frac{1}{\delta} \rceil$ and $1 \leq x \leq \lceil \frac{e}{\epsilon} \rceil$. In the beginning of the stream, all these counts are initialized to 0. Then, upon arrival of each a_k ($1 \leq k \leq t$), each of the counts $c_{j,h_j(a_k)} (1 \leq j \leq \lceil \log \frac{1}{\delta} \rceil)$ is incremented by 1.

For any $1 \leq i \leq n$, we define $\tilde{F}[i] = \min_j \{c_{j,h_j(i)}\}$. We will see that $\tilde{F}[i]$ provides a good approximation to F[i].

Memory cost Note that this algorithm only uses $O(\frac{1}{\epsilon} log \frac{1}{\delta})$ space.

Property Note this important property of the algorithm. For any $1 \leq i \leq n$:

$$\tilde{F}[i] \geq F[i]$$

Warning This implementation question requires substantial computation time. Python implementation reported to take 15min - 1 hour. Therefore, we advise you to start early.

Dataset The dataset in q4/data contains the following files:

1. `words_stream.txt` Each line of this file is a number, corresponding to the ID of a word in the stream.
2. `counts.txt` Each line is a pair of numbers separated by a tab. The first number is an ID of a word and the second number is its associated exact frequency count in the stream.
3. `words_stream_tiny.txt` and `counts_tiny.txt` are smaller versions of the dataset above that you can use for debugging your implementation.

4. `hash_params.txt` Each line is a pair of numbers separated by a tab, corresponding to parameters a and b which you may use to define your own hash functions (See explanation below).

Instructions Implement the algorithm and run it on the dataset with parameters $\delta = e^{-5}$, $\epsilon = e \times 10^{-4}$. (Note: with this choice of δ you will be using 5 hash functions - the 5 pairs (a, b) that you'll need for the hash functions are in `hash params.txt`). Then for each distinct word i in the dataset, compute the relative error $Er[i] = \frac{\tilde{F}[i]-F[i]}{F[i]}$ and plot these values as a function of the exact word frequency $\frac{F[i]}{t}$. **(You do not have to implement the algorithm in Spark.)**

The plot should use a logarithm scale both for the x and the y axes, and there should be ticks to allow reading the powers of 10 (e.g. 10^{-1}, 10^0, 10^1 etc...). The plot should have a title, as well as the x and y axes. The exact frequencies F[i] should be read from the counts file. Note that words of low frequency can have a very large relative error. That is not a bug in your implementation, but just a consequence of the way we defined our algorithm.

Answer the following question by reading values from your plot: What is an approximate condition on a word frequency in the document to have a relative error below $1 = 10^0$?

Hash functions You may use the following hash function (see example pseudocode), with $p = 123457$, a and b values provided in the hash params file and `n_buckets` (which is equivalent to $\lceil \frac{e}{\epsilon} \rceil$) chosen according to the specification of the algorithm. In the provided file, each line gives you a, b values to create one hash function.

```
# Returns hash(x) for hash function given by parameters a, b, p and n_buckets

def hash_fun(a, b, p, n_buckets, x)
{
    y = x [modulo] p
    hash_val = (a*y + b) [modulo] p
    return hash_val [modulo] n_buckets
}
```

Note: This hash function implementation produces outputs of value from 0 to `(n_buckets - 1)`, which is different from our specification in the **Strategy** part. You can either keep the range as $\{0, ..., \texttt{n\_buckets} - 1\}$, or add 1 to the hash result so the value range becomes $\{1, ..., \texttt{n\_buckets}\}$, as long as you stay consistent within your implementation.

What to submit

1. Log-log plot of the relative error as a function of the frequency. Answer for which word frequencies is the relative error below 1.
2. Upload the code. Make sure you write comments and elaborate on the process. Remember not to clear the cell outputs from your jupyter notebook.