EE412 Foundation of Big Data Analytics, Fall 2022
HW1

Due date: 10/05/2023 (11:59pm)

Submission instructions: Use KAIST KLMS to submit your homeworks. Your submission should be one gzipped tar file whose name is `YourStudentID_hw1.tar.gz`. For example, if your student ID is 20220000, and it is for homework #1, please name the file as `20220000_hw1.tar.gz`. You can also use these extensions: tar, gz, zip, tar.zip. Do not use other options not mentioned here.

Your zip file should contain total **six** files; one PDF file for writeup answers (`hw1.pdf`), four python files (`hw1_1.py`, `hw1_2_b.py`, `hw1_2_c.py`, and `hw1_3.py`), and the Ethics Oath pdf file. Before zipping your files, please make a directory named `YourStudentID_hw1` and put your files in the directory. Then, please compress the directory to make a zipped file. **If you violate any of the file name format or extension format, we will deduct 1 point of total score per mistake.**

Submitting writeup: Prepare answers to the homework questions into a single PDF file. You can use the following template. Please write as succinctly as possible.

Submitting code: Each problem is accompanied by a programming part. Put all the code for each question into a single file. We will only allow you to import **pyspark, numpy, re, sys, math, csv, and os** in your code. **You are not allowed to use any other libraries**. Good coding style (including comments) will be one criterion for grading. Please make sure your code is well structured and has descriptive comments.

Ethics Oath: For every homework submission, please fill out and submit the **PDF** version of this document that pledges your honor that you did not violate any ethics rules required by this course and KAIST. You can either scan a printed version into a PDF file or make the Word document into a PDF file after filling it out. Please sign on the document and submit it along with your other files.

Discussions with other people are permitted and encouraged. However, when the time comes to write your solution, such discussions (except with course staff members) are no longer appropriate: you must write down your own solutions independently. If you received any help, you must specify on the top of your written homework any individuals from whom you received help, and the nature of the help that you received. *Do not, under any circumstances, copy another person's solution.* We check all submissions for plagiarism and take any violations seriously.

1 MapReduce and Spark (30 pts)

We will find potential friends in a social network using Spark.

Social networks like Facebook, Twitter, and LiveJournal have users who are connected as friends in the form of a graph. In this problem, you will use a real dataset containing a LiveJournal friends graph to discover potential friends who have many mutual friends.

Download `hw1q1.zip` file from KLMS and use `soc-LiveJournal1Adj.txt`[1] as the dataset. Each line of the dataset is in the following format:

```
<User><TAB><Friends>
```

where `<User>` is an integer ID corresponding to a user, `<TAB>` is the tab character, and `<Friends>` is a comma-separated list of IDs of friends.

We are interested in finding pairs of users who are 1) not friends with each other, but 2) have many common friends. For example, if A is a friend of B, B is a friend of C, but A is not a friend of C, then we are interested in the pair (A, C), which has one common friend B.

The output should be in this format:

```
<User><TAB><User><TAB><Count>
```

where the first user ID is always smaller than the second user ID, and `<Count>` contains the number of mutual friends between the two users. The lines must be sorted by their counts in descending order. In case there are ties in counts, the lines are sorted by the first user ID integer in ascending order and then the second user ID integer in ascending order.

Here are some hints:

- The Spark Programming Guide may be useful: https://spark.apache.org/docs/latest/rdd-programming-guide.html.
- Before running on the entire dataset, debug your code on a small test dataset.
- If you run out of memory running Spark, try adding the flag --driver-memory 8G (on a Spark cluster, --executor-memory 8G). Also consider using a machine in the Haedong lounge.
- Although this problem requires thought, the code itself does not have to be long.

Please **use command-line** arguments to obtain the file path of the dataset. (Do not fix the path in your code.) For example, run:

```
bin/spark-submit hw1_1.py YourPathTo/soc-LiveJournal1Adj.txt
```

Please submit the following results:

[1]This dataset is from the Stanford Large Network Dataset Collection

- In your solution PDF file (hw1.pdf), please write (1) the **explanation** about your algorithm and (2) the program's **elapsed time** (i.e., the actual time taken from the start of your program to the end). You should give an explanation of the design choices in your Spark program. If your design choice and explanation are convincing, you can get full credit.
- Your source code in one file (with file name `hw1_1.py`).
  Your code should **print** the top-10 user pairs with their counts (total 10 lines). The printed results must be sorted by their counts in descending order, as following format for each line:
  `<User><TAB><User><TAB><Count>`
  You should use the `print` built-in function in Python to produce outputs. See the description above for more detailed information about output format.

2 Frequent Itemsets (35 pts)

(a) [10 pts] Solve the following problems which are based on the exercises in the MMDS textbook.

- Exercise 6.2.7
  Suppose we have market baskets that satisfy the following assumptions:
  (1) The support threshold is 10,000.
  (2) There are one million kinds of items, represented by the integers 0, 1, ..., 999999.
  (3) There are N frequent items, that is, items that occur 10,000 times or more.
  (4) There are one million pairs that occur 10,000 times or more.
  (5) There are $2M$ pairs that occur exactly once. Of these pairs, M consist of two frequent items; the other M each have at least one nonfrequent item.
  (6) No other pairs occur at all.
  (7) Integers are always represented by 4 bytes.

  Suppose we run the A-Priori Algorithm and can choose on the second pass between the triangular-matrix method for counting candidate pairs and a hash table of item-item-count triples. Neglect in the first case the space needed to translate between original item numbers and numbers for the frequent items, and in the second case neglect the space needed for the hash table. As a function of N and M, what is the minimum number of bytes of main memory needed to execute the A-Priori Algorithm on this data?

(b) [15 pts] Find frequent itemsets using the A-Priori algorithm.

Suppose you are an online retailer like Amazon and want to improve the shopping experience by analyzing customer behavior. In this problem, implement the A-Priori algorithm described in Chapter 6.2.5 to find frequent items and item pairs in an online browsing dataset.

Download `browsing.txt`[2] file from KLMS and use it as the dataset. Each line

[2]This dataset is from the Stanford Large Network Dataset Collection

of the dataset represents a browsing session of a customer where item id strings are delimited by spaces.

To store counts of pairs, please use the triangular method (and not the triples method). Also use a support threshold of 100.

Your code (`hw1_2_b.py`) should `print` total 12 lines: the number of frequent items, the number of frequent pairs, and the top-10 most frequent pairs with their counts and confidences, which must be sorted by their counts in descending order, as following format:

```
<Number of frequent items>
<Number of frequent pairs>
<Item 1><TAB><Item 2><TAB><Count><TAB><conf(1,2)><TAB><conf(2,1)>
...
<Item 1><TAB><Item 2><TAB><Count><TAB><conf(1,2)><TAB><conf(2,1)>
```

where the order between the first user and the second user is not important. You don't need to consider the case of 10th and 11th are tie. `<conf(A,B)>` is confidence of `<Item A>` $\rightarrow$ `<Item B>`.

This problem does not require Spark programming, and your code does not have to be long. (**Do not use Spark**)

Please **use command-line** arguments to obtain the file path of the dataset. (Do not fix the path in your code.) For example, run:

```
    python hw1_2_b.py YourPathTo/browsing.txt
```

Please submit the following results:

- The program's elapsed time (i.e., the actual time taken from the start of your program to the end) on the dataset. Please write it in your solution pdf file (hw1.pdf).
- Your source code in one file (with file name `hw1_2_b.py`).
  Your code should `print` total 12 lines: the number of frequent items, the number of frequent pairs, and the top-10 most frequent pairs with their counts and confidences, which must be sorted by their counts in descending order. Please use the `print` built-in function in Python to produce outputs. See the description above for more detailed information about output format.

(c) [10 pts] Following the problem 2-(b), find frequent triples using the A-Priori algorithm.

You can use any method to store counts of triples. Also use a support threshold of 100.

Your code (`hw1_2_c.py`) should `print` total 11 lines: the number of frequent triples, and the top-10 most frequent triples with their counts and confidences, which must be sorted by their counts in descending order, as following format:

```
<Number of frequent triples>
<Item 1><TAB><Item 2><TAB><Item 3><TAB><Count><TAB><Confidences>
...
<Item 1><TAB><Item 2><TAB><Item 3><TAB><Count><TAB><Confidences>
```

where `<Confidences>` is:

```
<conf((1,2),3)><TAB><conf((1,3),2)><TAB><conf((2,3),1>
```

The order between the three users is not important. You don't need to consider the case of 10th and 11th are tie. `<conf((A,B),C)>` is confidence of (`<Item A>`,`<Item B>`) $\rightarrow$ `<Item C>`.

This problem does not require Spark programming, and your code does not have to be long. (**Do not use Spark**)

Please **use command-line** arguments to obtain the file path of the dataset. (Do not fix the path in your code.) For example, run:

```
    python hw1_2_c.py YourPathTo/browsing.txt
```

Please submit the following results:

- The program's elapsed time (i.e., the actual time taken from the start of your program to the end) on the dataset. Please write it in your solution pdf file (hw1.pdf).
- Your source code in one file (with file name `hw1_2_c.py`).
  Your code should `print` total 11 lines: the number of frequent triples, and the top-10 most frequent triples with their counts and confidences, which must be sorted by their counts in descending order. Please use the `print` built-in function in Python to produce outputs. See the description above for more detailed information about output format.

3 Finding Similar Items (35 pts)

(a) [10 pts] Solve the following exercises in the MMDS textbook.

- Exercise 3.6.1
  What is the effect on probability of starting with the family of minhash functions and applying:

  (a) A 2-way AND construction followed by a 3-way OR construction.
  (b) A 3-way OR construction followed by a 2-way AND construction.
  (c) A 2-way AND construction followed by a 2-way OR construction, followed by a 2-way AND construction.
  (d) A 2-way OR construction followed by a 2-way AND construction, followed by a 2-way OR construction followed by a 2-way AND construction.

(b) [25 pts] Find similar documents using minhash-based LSH.

Suppose you are looking for very similar articles within a large article set. In this problem, implement the minhash-based LSH algorithm in Chapter 3.4.3 to efficiently find articles that have high Jaccard similarities.

Before you solve this problem, you should understand the concept of minhash-based LSH. Please read carefully the textbook and the EE412 lecture notes.

Download `articles.txt`[3] file from KLMS and use it as the dataset. Each line is in the following format:

```
<ARTICLE ID><SPACE><TEXT>
```

where `<ARTICLE ID>` is the article ID, `<SPACE>` is a single space, and `<TEXT>` is the text of the article.

When extracting k-shingles,

- Consider a shingle unit as an alphabetic character
- Treat the white space as a character
- Ignore non-alphabet characters except the white space
- Convert all characters to lower case
- Extract 3-shingles (i.e., set k=3)

When generating random hash functions, use the hash function $(ax + b)\%c$ as in the textbook. Let n be the number of rows as in Figure 3.4. Then set c to be the smallest prime number larger than or equal to n. Then set a to be a random integer between $[0, c-1]$ and b to be another random integer between $[0, c-1]$.

Set b and r so that the threshold is about 0.9. You can set $b = 6$ and $r = 20$,

[3]This dataset is from the University of Edinburgh

unless you prefer a different setting.

Your code (`hw1_3.py`) should `print` the candidate article ID pairs whose signature components agree by at least 0.9, as following format for each line:

```
<ARTICLE ID><TAB><ARTICLE ID>
```

where the order between the first user and the second user is not important. You can get full credit if your output contains all correct pairs regardless of the order between lines.

This problem does not require Spark programming. (**Do not use Spark**)

Please **use command-line** arguments to obtain the file path of the dataset. (Do not fix the path in your code.) For example, run:

```
python hw1_3.py YourPathTo/articles.txt
```

Please submit the following results:

- The program's elapsed time (i.e., the actual time taken from the start of your program to the end) on the dataset. Please write it in your solution pdf file (hw1.pdf).
- Your source code in one file (with file name `hw1_3.py`).
  Your code should **print** the candidate article ID pairs. Please use the `print` built-in function in Python to produce outputs. See the description above for more detailed information about output format.

Answers to Frequently Asked Questions

1) About the program's **elapsed time**: If you well-designed the codes, the programs' elapsed times (for all problems in HW1) should be less than 10 minutes. However, TAs will give 30 minutes when evaluating students' codes. If your code runs more than 30 minutes, TAs will not give the credits.
2) About **design choices**: You have to decide on your own which design choices including data structures (e.g., list, array, and dictionary) are good for each program. There can be multiple correct choices, and TAs will not answer which design choices are optimal or acceptable.
3) About using **external libraries**: We will only allow you to import **pyspark, numpy, re, sys, math, csv, and os** in your code. You are not allowed to use any other libraries. (You can use **time** library to check the program's elapsed time.)
4) In hw1_1, you can implement a function via def or lambda.
5) In hw1_1, if you still have memory error with --driver-memory 8G command, you can use Haedong server or try to remove useless(not related to result) "print" statement in your code.

6) In hw1_3, you should not fix the 'c' value. Your program should find the 'c' value for the algorithm. ('n' is total number of shingles. So, you should compute number of shingles and then you can find prime number 'c'.)
7) In hw1_3, after finding the candidates, you have to measure the actual Jaccard similarity.
8) Please use the `print` **built-in function in Python** to produce outputs.
9) Also, note that `<TAB>` means `\t` in the Python print function.
10) In this homework, you do not need to save the output files.
11) When you need to test your code: Our recommendation is to create a 'small' test file that you can check the answers on your own.