CSCI 5523 Introduction to Data Mining (Spring 2023)

Assignment 2 (10 points)

Deadline: Feb 27th 11:59 PM CDT (Gradescope)

1. Overview of the Assignment

In this assignment, you will implement the **SON algorithm** using the Apache Spark Framework. You will develop a program to find frequent itemsets in two datasets, one simulated dataset and one real-world dataset generated from Yelp dataset. The goal of this assignment is to apply the algorithms you have learned in class on large datasets more efficiently in a distributed environment.

2. Requirements

2.1 Programming Requirements

a. You must use **Python** to implement all tasks. You can only use standard python libraries (i.e., external libraries like numpy or pandas are not allowed).

b. **You are required to only use Spark RDD**, i.e. no point if using Spark DataFrame or DataSet.

2.2 Submission Platform

We will use a submission platform to automatically run and grade your submission. We highly recommend that you first test your scripts on your local machine before submitting your solutions. **We will use Gradescope to grade your submissions. The Gradescope submission site will be available on 02/20/2023**.

2.3 Programming Environment

Python 3.9.12, and Spark 3.2.1

2.4 Write your own code

Do not share code with other students!!

For this assignment to be an effective learning experience, **you must write your own code**! We emphasize this point because you will be able to find Python implementations of some of the required functions on the web. Please do not look for or at any such code!

TAs will combine all the code we can find from the web (e.g., Github) as well as other students' code from this and other (previous) sections for plagiarism detection. We will report all detected plagiarism to the university.

3. Datasets

In this assignment, you will implement the algorithms using simulated and real-world datasets. In Task 1, you will build and test your program with two small simulated CSV files (small1.csv and small2.csv).

In Task 2, you need to first generate a subset using business.json and review.json from the Yelp dataset (see Section 4.2.1). The dataset should have the same structure as the simulated datasets. You will test your program with this real-world dataset **locally** (you do not need to submit the dataset). **TAs will use different sampled subsets of the Yelp datasets for grading.**

Figure 1 shows the file structure, the first column is "user_id" and the second column is "business_id".

user_id	business_id
1	100
1	98
1	101
1	102
2	101
2	99

Figure 1: Input Data Format

4. Tasks

In this assignment, you will implement the **SON algorithm** to solve all tasks (Task 1 and Task 2) on top of Apache Spark Framework. You need to find **all the possible combinations of the frequent itemsets** in any given input file within the required time. You can refer to Chapter 6 from the Mining of Massive Datasets book and concentrate on section 6.4 – Limited-Pass Algorithms. (Hint: you can choose either A-Priori, MultiHash, or PCY algorithm to process each chunk of the data)

4.0 Submission

You need to submit the following files on Gradescope:

a. (Required) Two Python scripts (all lowercase): **task1.py, task2.py**

b. (Optional) Other Python scripts containing functions

4.1 Task 1: Simulated data (6 pts)

There are two CSV files (small1.csv and small2.csv) provided in the data folder. The small1.csv is a sample file that you can use to debug your code. **TAs will grade your code with small2.csv.**

4.1.1 Task Description

You need to create two kinds of market-basket models and build the algorithm for the two cases. More specifically, you will calculate the combinations of **frequent businesses (Case 1) and frequent users (Case 2)**, including singletons, pairs, triples, etc., that are qualified as frequent given a support threshold.

Case 1 (3 pts):

You will create a basket for each user containing the business ids reviewed by this user. If a business was reviewed more than once by a reviewer, we consider this business was rated only once, i.e., the business ids within each basket are unique. The generated baskets will be like:

user1: [business11, business12, business13, ...]

user2: [business21, business22, business23, ...]

user3: [business31, business32, business33, ...]

Case 2 (3 pts):

You will create a basket for each business containing the user ids who reviewed this business. Similar to case 1, the user ids within each basket are unique. The generated will be like:

business1: [user11, user12, user13, ...]

business2: [user21, user22, user23, ...]

business3: [user31, user32, user33, ...]

4.1.2 Input Format (Please make sure you use exactly the same input parameters names!)

Similar to HW1, you will use the "**argparse**" module to parse the following arguments:

1. Case number (**--c**): Integer that specifies the case: **1 for Case 1, and 2 for Case 2**.

2. Support (**--s**): Integer that defines the minimum count to qualify as a frequent itemset.

3. Input file path (**--input_file**): This is the path to the input file including path, file name and extension.

4. Output file path (**--output_file**): This is the path to the output file including path, file name and extension.

Execution example:

$ python task1.py --c <case number> --s <support> --input_file <input_file_path> --output_file <output_file_path>

Example: $ python task1.py --c 1 --s 10 --input_file ./data/ --output_file ./results/

4.1.3 Output Format

You must write the results, including runtime, the candidates of frequent itemsets, and the final frequent itemset, in **one** JSON file (.json).

(1) Runtime Duration

The total execution time (seconds) **from loading the file till finishing writing the output file**. You need to save the runtime with a "**Runtime**" tag: "Runtime": <time_in_seconds>, e.g., "Runtime": 100.00.

(2) Candidates

You need to output the candidates of frequent itemsets as a list after **the first pass** of the SON algorithm. The list contains lists of singletons, pairs, triples, etc, where the itemsets are sorted in the **lexicographical** order. Both user_id and business_id have the data type "string". You should use "**Candidates**" as the tag.

(3) Frequent Itemsets

You need to output the final frequent itemsets with the same format as Candidates. You should use "**Frequent Itemsets**" as the tag.

Here is an example of the output file format:

```
{
    "Candidates": [
        [["100"], ["101"], ["102"]],
        [["100", "101"], ["100", "102"], ["101", "102"]],
        [["100", "101", "102"]]
    ],
    "Frequent Itemsets": [
        [["100"], ["101"], ["102"]],
        [["100", "101"], ["100", "102"]]
    ],
    "Runtime": 183.28
}
```

4.2 Task 2: Yelp data (4 pts)

4.2.1 Task Description

In task2, you will explore the Yelp dataset to find the frequent business sets (**Case 1 in Task 1**). You will use the business.json and review.json provided in the data folder to generate the input user-business CSV file for local tests.

Data preprocessing (0 pts)

You will generate a sample dataset from business.json and review.json with following steps:

1. Filter the businesses in the state “Nevada” using business.json, i.e., “state”== “NV”.

2. Select (user_id, business_id) from review.json whose business_id is from Nevada and write them into a CSV file. The header should be “user_id, business_id”. Figure 3 shows an example of the output file.

user_id	business_id
hG7b0MtEbXx5QzbzE6C_VA	ujmEBvifdJM6h6RLv4wQIg
yXQM5uF2jS6es16SJzNHfg	NZnhc2sEQy3RmzKTZnqtwQ
nMeCE5-xsdleyxYuNZ_7rA	oxwGyA17NL6c5t1Etg5WgQ
FIk4lQQu1eTe2EpzQ4xhBA	8mIrX_LrOnAqWsB5JrOojQ

Figure 2: user_business file

You DO NOT need to submit the code or data for this step. The preprocessing code will NOT be graded for correctness. TAs will use a different filtering criteria to generate datasets for grading.

Apply SON algorithm (4 pts)

You will find the frequent business sets (**Case 1 Task 1**) with the large dataset you generated. You need to do the following steps:

1. Load the user_business CSV file and build Case 1 market-basket model

2. Filter qualified users who reviewed more than *k* businesses (*k* is the threshold)

3. Apply the SON algorithm to the market-basket model

4.2.2 Input format (Please make sure you use exactly the same input parameters names!)

Similar to HW1, you will use the "**argparse**" module to parse the following arguments:

1. Filter threshold (**--k**): Integer that is used to filter out qualified users

2. Support (**--s**): Integer that defines the minimum count to qualify as a frequent itemset.

3. Input file path (**--input_file**): This is the path to the input file including path, file name and extension. **This is the preprocessed CSV file you generated.**

4. Output file path (**--output_file**): This is the path to the output file including path, file name and extension.

Execution example:

$ python task2.py --k <threshold> --s <support> --input_file <input_file_path> --output_file <output_file_path>

Example: $ python task2.py --k 10 --s 10 --input_file ./data/ --output_file ./results/

4.2.3 Output Format

You must write the results, including runtime, the candidates of frequent itemsets, and the final frequent itemset, in **one** JSON file (.json). The format requirement is the same as 4.1.3.

Here is an example of the output file format:


```
{
    "Candidates": [
        [["a0v7Si0DK4cIko7AQY4YXg"], ["vAjSFWWrwEcduuQ"], ["z9gfVrZESgokFRXwkASofA"]],
        [["a0v7Si0DK4cIko7AQY4YXg", "vAjSFWWrwEcduuQ"],
            ["a0v7Si0DK4cIko7AQY4YXg", "z9gfVrZESgokFRXwkASofA"],
            ["vAjSFWWrwEcduuQ", "z9gfVrZESgokFRXwkASofA"]],
        [["a0v7Si0DK4cIko7AQY4YXg", "vAjSFWWrwEcduuQ", "z9gfVrZESgokFRXwkASofA"]]
    ],
    "Frequent Itemsets": [
        [["a0v7Si0DK4cIko7AQY4YXg"], ["vAjSFWWrwEcduuQ"], ["z9gfVrZESgokFRXwkASofA"]],
        [["a0v7Si0DK4cIko7AQY4YXg", "vAjSFWWrwEcduuQ"], ["a0v7Si0DK4cIko7AQY4YXg", "z9gfVrZESgokFRXwkASofA"]]
    ],
    "Runtime": 187.28
}
```


5. Evaluation Metric

Please refer to the following metrics to determine whether your implementation is efficient enough. We will terminate your code on a test dataset if the running time is extremely long.

Input File	Case	Support	Runtime (sec)
small2.csv	1	4	<=200

small2.csv	2	9	<=200

Input File	Filter Threshold (k)	Support	Runtime (sec)
user_business.csv	5	5	<=200

6. Grading Criteria

(% penalty = % penalty of possible points you get)

1. You can use your free 5-day extension separately or together. During grading, TAs will count the number of late days **based on the submission time.** For example, if the due date is 2023/09/10 23:59:59 CDT and your submission is 2023/09/11 05:23:54 CDT, the number of late days would be 1. **TAs will record grace days by default (NO separate Emails)**. However, if you want to save it next time and treat your assignment as late submission (with some penalties), **please leave a comment in your submission.**
2. There will be no point if your submission cannot be executed on the grading platform.
3. There is no regrading. Once the grade is posted on Canvas, we will only regrade your assignments if there is a grading error. No exceptions.
4. Homework assignments are due at 11:59 pm CT on the due date and should be submitted on Canvas. Late submissions within 24 hours of the due date will receive a 30% penalty. Late submissions after 24 hours of the due date will receive a 70% penalty. Every student has FIVE free late days for homework assignments. You can use these free late days for any reason, separately or together, to avoid the late penalty. There will be no other extensions for any reason. **You cannot use the free late days after the last day of the class.**

7. Submission Instructions

1. You will submit your programming assignment via Gradescope. You can access it from Canvas. The Gradescope will be available **next Monday 02/20**.

2. On Gradescope, you will find a link for the submission of **TWO** Python files (lowercase):
   task1.py
   task2.py

3. You MUST follow the parameter naming convention (case sensitive) provided by the description of each task (see execution examples of each task). Failure to do this will result in failing to pass all test cases (even if your codes are correct). It is your responsibility to make sure that your filenames strictly follow the instructions provided above.

4. Once you submit your programming files, the grading scripts on Gradescope will take some time to run your files and run test cases. You should be able to see the final score for each task. **However, you should NOT use Gradescope as a debugging tool.** You should first debug your code on your local machine before submitting your code to Gradescope!

5. On Gradescope, there is a function named ``test_tasklog'' to display the stdout and stderr when running your code on Gradescope.

6. If you did not pass a test case for a particular task, this means that there is (are) an error (errors) that you have to fix. On the other hand, passing all test cases is a good sign, **but it does not mean that you will pass the test cases that the TA will use for grading**. Your final grade is ultimately determined by the test cases that are run after the due date (i.e., not the test cases that are available to students before the due date).

7. You are allowed to submit your codes to Gradescope as long as it is before the due date. If you submit your codes after the due dates, it will be recorded as a late submission (See Section 6 for information about grading criteria).