CSCI 5523 Introduction to Data Mining (Spring 2025)

Assignment 2 (15 points)

Deadline: March 17th 11:59 PM CDT

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 (Task 1) and one real-world dataset generated from Yelp dataset (Task 2). 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.**

2.3 Programming Environment

Python 3.9.12, and Spark 3.2.1 or Spark 3.5.4

2.4 Write your own code

Do not use large language models (e.g., ChatGPT, Copilot) or 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), other students' code from this and other (previous) sections, and code generated from AI tools 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 simulated CSV files (small1.csv and small2.csv).

In Task 2, you will first generate a subset using business.json and review.json from the Yelp dataset (see Section 4.2.1). You will build and 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 an example of the file format, the first column is "year", the second column is "user_id", and the third column is "business_id".

year	user_id	business_id
2017	1	99
2018	1	105
2019	1	109
2017	1	102
2017	2	111
2018	2	95

Figure 1: Input Data Format

4. Tasks

The goal of this assignment is to find all the possible combinations of the frequent itemsets in a given input file within the required time. You will implement the **SON algorithm** to solve Task 1 and Task 2. In the first pass of SON, you will process each data partition using **the A-Priori algorithm for Task 1 and the PCY algorithm for Task 2.** Complete the provided skeleton code in task1.py and task2.py. You can write your own helper functions to support the tasks.

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 (8 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 will implement two cases (argument: *c*) of market-basket models:

- **Case 1**: Calculate frequent businesses
- **Case 2**: Calculate frequent users

For each case, you need to find all frequent itemsets, including singletons, pairs, triples, etc., based on a given support threshold (argument: *t*) in a specified year (argument: *y*). To achieve this, you will use the **SON algorithm** by applying the **A-Priori algorithm to each partition of the data in the first pass** to generate candidate itemsets, and then counting the occurrences of these candidates across the entire dataset in the second pass to identify globally frequent itemsets.

Case 1 (4 pts):

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

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

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

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

Case 2 (4 pts):

You will create a basket for each business containing the user ids who reviewed this business in the specified year. Similar to case 1, the user ids within each basket are unique. The baskets 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 parameter names!)**

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

1. Year used (**--y**): Integer that specifies what year is used for the analysis
2. Case number (**--c**): Integer that specifies the case: **1 for Case 1, and 2 for Case 2**.
2. Threshold (**--t**): Integer that defines the support threshold to qualify as a frequent itemset.
3. Input file path (**--input_file**): Path to the input file, including path, file name, and extension.
4. Output file path (**--output_file**): Path to the output file, including path, file name, and extension.

Execution example:

$ python task1.py --y <year> --c <case number> --t <minimum_count> --input_file <input_file_path> --output_file <output_file_path>

Example: $ python task1.py --y 2016 --c 1 --t 7 --input_file ./data/small1.csv --output_file ./results/task1_output.json

4.1.3 Output Format

You must output the results, including (1) the candidates of frequent itemsets, (2) the final frequent itemsets, and (3) runtime (seconds), and, in **one** JSON file (.json).

(1) 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.

(2) Frequent Itemsets: You need to output the **final frequent itemsets** along with their support. You should use "**Frequent Itemsets**" as the tag.

(3) Runtime Duration: You need to save the runtime with a "**Runtime**" tag: "Runtime": <time_in_seconds>, e.g., "Runtime": 100.00. The total execution time (seconds) from loading the file till finishing writing the output file. **The provided code includes specific lines for measuring execution time.**

Here is an example of the output file format:


```
{
    "Candidates":
    [
        [["11"], ["12"], ["24"]],
        [["12", "24"]]
    ],
    "Frequent Itemsets":
    [
        [{"itemset": ["11"], "support": 20},
        {"itemset": ["12"], "support": 21},
        {"itemset": ["24"], "support": 20}],
        [{"itemset": ["12", "24"], "support": 15}]
    ],
    "Runtime": 11.328850746154785
}
```


Figure 2: Task 1 Example Output

4.2 Task 2: Yelp data (7 pts)

4.2.1 Task Description

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

Data preprocessing (0 pts)

You will **run the provided preprocessing code "gen_nevada_data.py"** to generate a sample dataset from business.json and review.json with the 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.

3. Generate a CSV file with the columns “user_id”, “business_id”. Figure 3 shows an example of the output file.

user_id	business_id
bAhqAPoWaZYcyYi7bs024Q	LUN6swQYa4xJKaM_UEUOEw
3CJUJILq7CLHk_9OrvpvQg	I4Nr-MVc26qWr08-S3Q1ow
zFYFuufYWQSPj0r5lrKQKg	wJj1EwYcXHdvA9zKqmb5hQ
II8eB5mYk200GW-m-wpsug	z9aXGRH8xtqpNDFE5_I3KA
h27iTTIaDicevvaopobUdA	X5uxdU9GHoUAqo2wmbg3OQ

Figure 3: user_business file

You DO NOT need to submit the code or data for this step. TAs will use different filtering criteria to generate datasets for grading.

Apply SON algorithm combined with PCY algorithm (7 pts)

You will find the frequent business sets **(Task 1 Case 1)** with the large dataset you generated. To achieve this, you will use the **SON algorithm** by applying the **PCY algorithm to each partition of the data in the first pass** to generate candidate itemsets, and then counting the occurrences of these candidates across the entire dataset in the second pass to identify globally frequent itemsets (Hint: Use a hash table to hash the pairs into hash buckets).

You need to do the following steps:

1. Load the preprocessed user_business CSV file and build Case 1 market-basket model
2. Filter qualified users who reviewed more than ***f* unique** businesses (argument: *f*)
3. Apply the SON algorithm combined with the PCY algorithm to the market-basket model. Hints are provided with comments in the code skeleton file - task2.py

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

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

1. Filter threshold (**--f**): Integer that is used to filter out qualified users who reviewed **more than** *f* distinct businesses.

2. Threshold (**--t**): Integer that defines the support threshold to qualify as a frequent itemset.

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

5. Output file path (**--output_file**): Path to the output file, including path, file name, and extension.

Execution example:

$ python task2.py --f <filter users> --t <threshold> --input_file <input_file_path> --output_file <output_file_path>

Example: $ python task2.py --f 4 --t 7 --input_file ./data/user_business.csv --output_file ./results/task2_output.json

4.2.3 Output Format

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

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. Correct implementations with execution time below these requirements will get full points. Slower execution times that output correct results will get half points.

Input File	Case(c)	Threshold (t)	Runtime (sec)
small2.csv	1	4	<=20
small2.csv	2	9	<=14

Input File	Filter Threshold (f)	Threshold (t)	Runtime (sec)
user_business.csv	4	8	<=60

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 2025/3/17 23:59:59 CDT and your submission is 2025/3/18 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 Gradescopeis available now.

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.

```
Stdout when running your task files
logs for task1c1
Setting default log level to "WARN".

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

23/03/08 19:20:08 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable


[Stage 0:>                                                          (0 + 1) / 1]
```

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).