DSCI 553 Foundations and Applications of Data Mining

Summer 2020

Assignment 2

**Deadline: Oct 1st 11:59 PM PST**

## 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. There will be **10% bonus** for each task if you also submit a Scala implementation and both your Python and Scala implementations are correct.

b. You are required to only use Spark RDD in order to understand Spark operations more deeply. You will not get any point if you use Spark DataFrame or DataSet.

### 2.2 Programming Environment

Python 3.6, Scala 2.11, and Spark 2.3.2

**We will use Vocareum to automatically run and grade your submission. We highly recommend that you first test your script on your local machine and then submit to Vocareum.**

### 2.3 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.

### 2.4 What you need to turn in

a. Three Python scripts, named: (all lowercase): **task1.py, task2.py, preprocess.py**

b1. [OPTIONAL] two Scala scripts, named: (all lowercase)

**task1.scala, task2.scala (No need to write preprocessing code in Scala)**

b2. [OPTIONAL] one jar package, named: **hw2.jar** (all lowercase)

Note. You don't need to include your output files for both tasks. We will grade on your code with our testing data (data will be in the same format).

### 3. Datasets

In this assignment, you will use one simulated dataset and one real-world dataset. In task 1, you will build and test your program with a small simulated CSV file that has been provided to you[1].

For task 2, you need to generate a subset using business.json and review.json from the Yelp dataset with the same structure as the simulated data. Figure 1 shows the file structure, the first column is user_id and the second column is business_id. In task2, you will test your code with this real-world data.

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

Figure 1: Input Data Format

**We will only provide submission report for small1.csv on Vocareum for task 1. No submission report will be provided for task2. You are encouraged to use command line to run the code for small2.csv as well as for task2 to get a sense of the running time.**

### 4. Tasks

In this assignment, you will implement the **SON algorithm** to solve all tasks (Task 1 and 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).

#### 4.1 Task 1: Simulated data (6 pts)

There are two CSV files (small1.csv and small2.csv) provided on the Vocareum in your workspace. The small1.csv is just a sample file that you can used to debug your code. For task1, **we will test your code on small2.csv for grading.**

In this task, you need to build two kinds of market-basket model.

**Case 1 (3 pts)**:

You will calculate the combinations of **frequent businesses** (as singletons, pairs, triples, etc.) that are qualified as frequent given a support threshold. You need to 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

[1]

consider this product was rated only once. More specifically, the business ids within each basket are unique. The generated baskets are similar to:

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

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

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

**Case 2 (3 pts)**:

You will calculate the combinations of **frequent users** (as singletons, pairs, triples, etc.) that are qualified as frequent given a support threshold. You need to create a basket for each business containing the user ids that commented on this business. Similar to case 1, the user ids within each basket are unique. The generated baskets are similar to:

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

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

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

**Input format:**

1. Case number: **Integer** that specifies the case. **1 for Case 1 and 2 for Case 2**.

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

3. Input file path: This is the path to the input file including path, file name and extension.

4. Output file path: This is the path to the output file including path, file name and extension.

**Output format:**

1. Console output - Runtime: **the total execution time from loading the file till finishing writing the output file**

You need to **print the runtime in the console** with the "Duration" tag: "Duration: <time_in_seconds>", e.g., "Duration: 100.00"

2. Output file:

(1) Output-1
You should use "Candidates:"as the tag. For each line you should output the candidates of frequent itemsets you find after the first pass of SON algorithm, followed by an empty line after each fequent-X itemset combination list. The printed itemsets must be sorted and arranged in **lexicographical** order. (Both user_id and business_id have the data type "string".)

(2) Output-2
You should use "Frequent Itemsets:"as the tag. For each line you should output the final frequent itemsets you found after finishing the SON algorithm. The format is the same with the Output-1. The printed itemsets must be sorted and arranged in **lexicographical** order.
Here is an example of the output file:

```
Candidates:
('100'),('101'),('102'),('103'),('105'),('97'),('98'),('99')

('100', '101'),('100', '98'),('100', '99'),('101', '102'),('101', '97'),('101', '98'),('101', '99'),('10

('100', '101', '98'),('100', '101', '99'),('101', '97', '99'),('102', '103', '105'),('102', '103', '98')

('102', '103', '105', '98'),('102', '103', '105', '99'),('102', '103', '98', '99'),('102', '105', '98',

('102', '103', '105', '98', '99')

Frequent Itemsets:
('100'),('101'),('102'),('103'),('97'),('98'),('99')

('100', '101'),('100', '98'),('101', '102'),('101', '97'),('101', '98'),('101', '99'),('102', '103'),('1

('100', '101', '98'),('101', '97', '99'),('102', '103', '99'),('97', '98', '99')
```

Figure 2: Output Data Format

Both the output-1 result and output-2 should be saved in ONE output result file.

**Execution example:**

Python:

spark-submit task1.py  <case number> <support> <input_file_path> <output_file_path>

Scala:

spark-submit –class task1 hw2.jar  <case number> <support> <input_file_path> <output_file_path>

### 4.2 Task 2: Yelp data (3.6 pts)

In task2, you will explore the Yelp dataset to find the user travel pattern (utilizing your code for the task1). You will jointly use the yelp_academic_dataset_business.json and yelp_academic_dataset_review.json to generate the input user-business CSV file yourselves.

With the Yelp review dataset, one interesting question is to find active reviewers. A straight forward definition fo active reviewer is a reviewer who posts a certain number of reviews. However, in this task, we view the active level of reviewers from a different perspective. We will find the common set of states that reviewers travel. To find where the reviewers travel, we extract the states of the business that the users reviews. Specifically, “business.json” contains business_id, state for each business, and “review.json” contains user_id and state for each review.

**(1) Data preprocessing**

You need to generate a sample dataset from yelp_academic_dataset_business.json and yelp_academic_dataset_review.json [2] with following steps:

1. You will study the business whose star is 4.0 or more, (i.e., stars >= 4.0). Note that you are supposed to use “stars” in business.json not review.json)

2. Select “user_id” and “business_id” from review.json whose “business_id” whose star is more than 4.0 . Each line in the CSV file would be “user_id1, state”.

[2] https://www.yelp.com/dataset

3. The header of CSV file should be “user_id, state”

You need to save the dataset in CSV format. Figure 3 shows an example of the output file

| user_id | state |
|---|---|
| 1 | 'NV' |
| 1 | 'CA' |
| 1 | 'QC' |
| 2 | 'NV' |

Figure 3: user_state file

**You need to submit the code for this data preprocessing step. The preprocessing code will NOT be graded for correctness. No need to submit the generated user-business file. We will use different filters to generate another dataset for grading.**

**As similer to the taks1, the format for SON algorithm should follow**

**user1: [state11, state 12, state 13, ...]**

**user2: [state21, state 22, state 23, ...]**

**user3: [state31, state 2, business33, ...]**

**(2) Apply SON algorithm**

The requirements for task 2 are similar to task 1. However, you will test your implementation with the large dataset you just generated. For this purpose, you need to report the total execution time. For this execution time, we take into account also the time from reading the file till writing the results to the output file. You are asked to find the frequent state sets (**only case 1**) from the file you just generated. The following are the steps you need to do:

1. Reading the user_state.csv file in to RDD and then build the model;

2. Find out qualified users who reviewed more than *k* businesses. (*k* is the filter threshold);

3. Apply the SON algorithm code to the filtered market-basket model;

**Input format:**

1. Filter threshold: **Integer** that is used to filter out qualified users.

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

3. Input file path: This is the path to the input file including path, file name and extension.

4. Output file path: This is the path to the output file including path, file name and extension.

**Output format:**

1. Runtime: **the total execution time from loading the file till finishing writing the output file**

You need to **print the runtime in the console** with the "Duration" tag, e.g., "Duration: 100".

2. Output file

The output file format is the same with task 1. Both the intermediate results and final results should be saved in ONE output result file.

**Execution example:**

Python:

spark-submit task2.py <support> <input_file_path> <output_file_path>

Scala:

spark-submit –class task2 hw2.jar <support> <input_file_path> <output_file_path>

**5. Evaluation Metric**

**Task 1:**

| Input File | Case | Support | Runtime (sec) |
|---|---|---|---|
| small2.csv | 1 | 4 | <=200 |
| small2.csv | 2 | 9 | <=200 |

**Task 2:**

| Input File | Star threshold | Support | Runtime (sec) |
|---|---|---|---|
| user_state.csv | 4.0 | 4 | <=2,000 |

## 6. Grading Criteria

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

1. You can use your free 5-day extension separately or together. You must submit a late-day request via https://forms.gle/vWZPAujxsHc2w3QB7. This form is recording the number of late days you use for each assignment. By default, we will not count the late days if no request submitted.

2. There will be 10% bonus for each task (i.e., 0.36pts, 0.3pts, and 0.3pts) if your Scala implementations are correct. Only when your Python results are correct, the bonus of using Scala will be calculated. There is no partial point for Scala.
3. There will be no point if your submission cannot be executed on Vocareum.
4. There is no regrading. Once the grade is posted on the Blackboard, we will only regrade your assignments if there is a grading error. No exceptions.
5. There will be 20% penalty for the late submission within one week and no point after that.
6. If you use your late days, there wouldn't be the 20% penalty but the final deadline wouldn't be extended.