INF553 Foundations and Applications of Data Mining

Spring 2021

Assignment 2

Deadline: Mar 2nd 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 learn how to implement and apply the algorithms you have learned in class on real, large datasets efficiently in a distributed environment.

2. Requirements

2.1 Programming Requirements

a. You must use Python to implement all tasks. There will be a 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. You will not get any point if you use Spark DataFrame or DataSet.

2.2 Programming Environment

Python 3.6, Scala 2.11.8 and Spark 2.3.0

We will use Vocareum to automatically run and grade your submission. We highly recommend that you first test/debug your scripts on your local machine and once they are ready, submit them to Vocareum.

2.3 Write your own code

Do not share your 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.

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 (also previous) sections for plagiarism detection. We will report all detected plagiarism.

2.4 What you need to turn in

a. Four Python scripts, named: (all lowercase): **task1.py, task2.py, task3.py, preprocess.py, task3_fp.py (optional)**

b1. [OPTIONAL] 4 Scala scripts, named: (all lowercase)

task1.scala, task2.scala, task3.scala, task3_fp.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 on Vocareum.

For task 2, you need to first generate a subset of business.json and review.json data from the Yelp dataset. You should generate the data subsets using the same data structure as the simulated data (Figure 1). Figure 1 shows the data structure, where 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 a submission report for small1.csv on Vocareum for task 1. No submission report will be provided for task2. You are encouraged to use the 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 Task 1 and 2 on top of the 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 textbook, especially section 6.4 – Limited-Pass Algorithms, for implementing your algorithm(s). (Hint: you can choose either A-Priori, MultiHash, or PCY algorithm to process each chunk of the data) For task 3 you need to use the FPGrowth algorithm in spark.mllib to solve Task 2.

4.1 Task 1: Simulated data (3 pts)

There are two CSV files ($ASNLIB/publicdata/small1.csv and $ASNLIB/publicdata/small2.csv) provided on Vocareum in your workspace. The file small1.csv is just a sample file that you can use 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 models.

Case 1 (1.5 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. The basket will contain the business ids reviewed by a user. If a business was reviewed more than once by a reviewer, we consider this product was rated only once. Specifically, the business ids within each basket are unique since each basket is a set of ids. Examples of user baskets are:

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

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

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

Case 2 (1.5 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. The basket will contain the ids from users who commented on this business. Similar to case 1, the user ids in each basket are unique. Examples of business baskets are:

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

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

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

Input format:

1. Case number: An **integer** that specifies the case, **1 for Case 1 and 2 for Case 2**.

2. Support: An **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>", eg: "Duration: 100.00"

2. Output file:

(1) Output-1
You should use "Candidates:" to indicate your candidate section in your output file. For each line you should output the candidates of frequent itemsets identified after the first pass of SON algorithm, followed by an empty line after each frequent X itemset combination list (i.e., X can be single, pair, triple, and so on). The printed itemsets must be sorted in the **lexicographical** order. (Both user_id and business_id have the data type "string".)

(2) Output-2

You should use "Frequent Itemsets:" to indicate your frequent itemset section in your output file. For each line you should output the final frequent itemsets identified after finishing the SON algorithm. The output format for the frequent X itemsets is the same as in Output-1. The printed itemsets must be sorted in the **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')
```

Or refer to $ASNLIB/publicdata/example_output for example. Both the output-1 result and output-2 should be saved in ONE output 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 (4 pts)

In task2, you will explore the Yelp dataset to find the frequent business sets (**only case 1**). You will jointly use the business.json and review.json to generate the input user-business CSV file yourselves.

(1) Data preprocessing

You need to generate a sample dataset from business.json and review.json with the following steps:

1. The state of the business you need is Nevada, i.e., filtering 'state'== 'NV'.

2. Select "user_id" and "business_id" from review.json whose "business_id" is from Nevada. Each line in the CSV file would be "user_id1, business_id1".

3. The header of CSV file should be "user_id, business_id"

You need to save the dataset in CSV format. Figure 2 shows an example of the output file (or refer to $ASNLIB/publicdata/example_csv).

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

Figure 2: user_business file

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

(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 also 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 business sets (**only case 1 in task 1**) from the file you just generated. The following are the steps you need to do in this task:

1. Reading the user_business CSV file into RDD and then build the case 1 market-basket model;

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

3. Apply the SON algorithm to those qualified users.

Input format:

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

2. Support: An **integer** that defines the minimum count (support) 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 <filter threshold> <support> <input_file_path> <output_file_path>

Scala:

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

4.3 Task 3: FP-growth algorithm (1 pt)

In task 3, you will use the preprocessed user_business.CSV from task 2 but use the fp-growth algorithm in Spark MLlib to obtain the frequent itemsets. The input format and steps for task for task 3 are the same as task 2 except that you directly use fp-growth in step 3 instead of the SON algorithm.

After you apply fp-growth algorithm, compare the differences with the frequent itemsets of task 2 by simply generating the **number** of frequent itemsets from task 2 and task3 and their intersections, respectively.

Output format:

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

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

2. Output file

Please refer to the format in the example figure below. Note that the numbers in this example is a made-up example. Your results from tasks 2 and 3 should have the same results.

```
Task2,30
Task3,50
Intersection,10
```

Bonus

If you implement the SON with the fp-growth algorithm, named **task3_fp.py** ,yourself instead of using the MLlib, you will get a one point bonus.

Execution example:

Python:

spark-submit task3.py <filter threshold> <support> <input_file_path> <output_file_path>

Scala:

spark-submit –class task3 hw2.jar <filter threshold> <support> <input_file_path> <output_file_path>

5. Evaluation Metric

Task 1:

Input File	Case	Support	Runtime (sec)
small2.csv	1	4	<=150
small2.csv	2	9	<=150

Task 2:

Input File	Filter Threshold	Support	Runtime (sec)
user_business.csv	70	50	<=400

Task 3:

Input File	Filter Threshold	Support	Runtime (sec)
user_business.csv	70	50	<=100

If you implement your own fp-growth, the runtime limit is 400s

6. Grading Criteria

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

1. You can use your free 5-day extension separately or together, and you need to email your TA to indicate that you are using the free days within 24 hours of your submission.

2. There will be 10% bonus for each task (i.e., 0.3 pts, 0.4 pts, 0.1pts) if your Scala implementations are correct. If you implement fp-growth yourself, you get 1 bonus pts for Python and 0.1 pts for scala. 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 programs cannot be executed on Vocareum.
Please start your assignment early! You can resubmit on Vocareum. We will grade your last submission.

4. There is no regrade. 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 a week and no point after a week.

6. There will be no point if the total execution time exceeds the Section 5 evaluation metric

7. If the outputs of your program are unsorted or partially sorted, there will be 50% penalty.