INF553 Foundations and Applications of Data Mining

Spring 2019

Assignment 2

Deadline: Feb. 25th 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 these library versions to compile and test your code. There will be a 20% penalty if we cannot run your code due to the library version inconsistency.

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

Your submission must be a zip file with name: **firstname_lastname_hw2.zip** (all lowercase). You need to pack the following files in the zip file (see Figure 1):

a. two Python scripts, named: (all lowercase)

firstname_lastname_task1.py

firstname_lastname_task2.py

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

firstname_lastname_task1.scala

firstname_lastname_task2.scala

b2. [OPTIONAL] one jar package, named: **firstname_lastname_hw2.jar** (all lowercase)

c. the task2 result explanation: (all lowercase)

firstname_lastname_explanation.pdf

d. You don't need to include your results. We will grade on your code with our testing data (data will be in the same format).

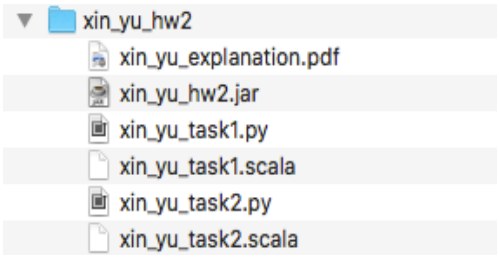

Figure 1: Submission Structure

3. Datasets

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

Then you need to generate a subset using business.json and review.json from the Yelp dataset (https://www.yelp.com/dataset) with the same structure as the simulated data. Figure 2 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 2: Input Data Format

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 the 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 (3 pts)

There are two CSV files (small1.csv and small2.csv) on the Blackboard. The small1.csv is just a test file that you can used to debug your code. For task1, **we will only test your code on small2.csv**.

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

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 containing the business ids reviewed by this user. If a business was reviewed more than once by a reviewer, we 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 (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 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. 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:

(1) Intermediate result
You should use "Candidates:"as the tag. For each line you should output the candidates of frequent itemsets you found after the first pass of SON algorithm followed by an empty line after each combination. The printed itemsets must be sorted in **lexicographical** order (Both user_id and business_id are type of string).

(2) Final result
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 intermediate results. The printed itemsets must be sorted 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')
```

Both the intermediate results and final results should be saved in ONE output result file "firstname_lastname_task1.txt".

Execution example:

Python:

spark-submit firstname_lastname_task1.py <case number> <support> <input_file_path> <output_file_path>

Scala:

spark-submit –class firstname_lastname_task1 firstname_lastname_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.

4.2.1 SON algorithm on Yelp data (3.5 pts):

(1) Data preprocessing

You need to generate a sample dataset from business.json and review.json with 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 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 3: user_business file

You do NOT need to submit the code and the output file of this data preprocessing step. Please do NOT include the code of this step in the task2.py or task2.scala file.

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

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

2. Filter 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 “firstname_lastname_task2.txt”.

Execution example:

Python:

spark-submit firstname_lastname_task2.py <filter threshold> <support> <input_file_path> <output_file_path>

Scala:

spark-submit –class firstname_lastname_task2 firstname_lastname_hw2.jar <filter threshold> <support> <input_file_path> <output_file_path>

4.2.2 Result explanation (0.5 pts):

You need to explore the results of your output from task 2, especially the frequent itemsets with the highest number of items. Here are some hints:

1. Compare the business name and category

2. Consider the location of the business

This is an open question, you are free to talk about any findings that you think are interesting. You need to submit your findings in a PDF file. No more than 150 words.

5. Evaluation Metric

Task 1:

Input File	Case	Support	Runtime (sec)
small2.csv	1	4	<=200
small2.csv	2	7	<=100

Task 2:

Input File	Filter Threshold	Support	Runtime (sec)
user_business.csv	70	50	<=1500

6. Grading Criteria

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

1. You can use your free 5-day extension separately or together.
2. There will be 10% bonus if you use both Scala and Python.
3. If you do not apply the SON algorithm, there will be no point for this assignment.
4. If we cannot run your programs with the command we specified, there will be 80% penalty.
5. If your program cannot run with the required Scala/Python/Spark versions, there will be 20% penalty.
6. If our grading program cannot find a specified tag, there will be no point for this question.

7. If the outputs of your program are unsorted or partially sorted, there will be 50% penalty.
8. If the header of the output file is missing, there will be 10% penalty.
9. We can regrade on your assignments within seven days once the scores are released. No argue after one week. There will be 20% penalty if our grading is correct.
10. There will be 20% penalty for late submission within a week and no point after a week.
11. There will be no point if the execution time in each task exceeds the runtime in Secion 6 Evaluation Metric.
12. Only when your results from Python are correct, the bonus of using Scala will be calculated. There is no partially point for Scala. See the example below:

Example situations

Task	Score for Python	Score for Scala (10% of previous column if correct)	Total
Task1	Correct: 3 points	Correct: 3 * 10%	3.3
Task1	Wrong: 0 point	Correct: 0 * 10%	0.0
Task1	Partially correct: 1.5 points	Correct: 1.5 * 10%	1.65
Task1	Partially correct: 1.5 points	Wrong: 0	1.5