DSCI553 Foundations and Applications of Data Mining

Fall 2023

Assignment 2

Deadline: Oct 5, 23:59 PM PST

1. Overview of the Assignment

In this assignment, you will implement the **SON Algorithm** using the Spark Framework. You will develop a program to find frequent itemsets in two datasets, one simulated dataset and one real-world generated 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). 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 points if you use Spark DataFrame or DataSet.

c. **Python standard library** set : https://docs.python.org/3/library/

2.2 Programming Environment

Python 3.6, JDK 1.8, Scala 2.12, and Spark 3.1.2

We will use these library versions to compile and test your code. There will be no point if we cannot run your code on Vocareum.

On Vocareum, you can call `spark-submit` located at `/opt/spark/spark-3.1.2-bin-hadoop3.2/bin/spark-submit`. (Do not use the one at /usr/local/bin/spark-submit). We use `--executor-memory 4G --driver-memory 4G` on Vocareum for grading.

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. We will report all detected plagiarism and severe penalties will be given for the students whose submissions are plagiarized.

2.4 What you need to turn in

We will grade all submissions on Vocareum and the submissions on the blackboard will be ignored. Vocareum produces a submission report after you click the "Submit" button (It takes a while since Vocareum needs to run your code in order to generate the report). Vocareum will **only grade Python scripts** during the **submission phase** and it will **grade both Python and Scala** during the **grading phase**.

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

task1.py, task2.py

b. **[OPTIONAL]** hw2.jar and two Scala scripts, named: (all lowercase)

hw2.jar, task1.scala, task2.scala

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

d. Students can submit an **unlimited number of times**. Only the latest submission will be accepted and graded.

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.

Then in task2 you need to generate a subset using the **Ta Feng dataset** with a structure similar to the simulated data.

Figure 1 shows the file structure of task1 simulated csv, 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 2) on top of 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 (3 pts)

There are two CSV files (`small1.csv` and `small2.csv`) in Vocareum under `'../resource/asnlib/publicdata'`. The small1.csv is just a test file that you can use 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 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 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 types 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:

Both the intermediate results and final results should be **saved in ONE output result 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')
```

Command line Format:

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>

Command line Example:

```
/opt/spark/spark-3.1.2-bin-hadoop3.2/bin/spark-submit --executor-memory 4G --driver-memory 4G task1.py 1 4 ../resource/asnlib/publicdata/small1.csv task1_output.txt
```

4.2 Task 2: Ta Feng data (4 pts)

In task 2, you will explore the Ta Feng dataset to find the frequent itemsets (**only case 1**). You will use data found here from Kaggle (https://bit.ly/2miWqFS) to find product IDs associated with a given customer ID each day. Aggregate all purchases a customer makes within a day into one basket. In other words, assume a customer purchases at once all items purchased within a day. The data file is provided at `../resource/asnlib/publicdata/ta_feng_all_months_merged.csv`

Note: Be careful when reading the csv file as spark can read the product id numbers with leading zeros. You can manually format Column F (PRODUCT_ID) to numbers (with zero decimal places) in the csv file before reading it using spark.

SON Algorithm on Ta Feng data:

You will create a data pipeline where the input is the raw Ta Feng data, and the output is the file described under "output file". You will pre-process the data, and then from this pre-processed data, you will create the final output. Your code is allowed to output this pre-processed data during execution, but you should NOT submit homework that includes this pre-processed data.

(1) Data preprocessing

You need to generate a dataset from the Ta Feng dataset with following steps:

1. Find the date of the purchase (column TRANSACTION_DT), such as December 1, 2000 (12/1/00)
2. At each date, select "CUSTOMER_ID" and "PRODUCT_ID".
3. We want to consider all items bought by a consumer each day as a separate transaction (i.e., "baskets"). For example, if consumer 1, 2, and 3 each bought oranges December 2, 2000, and consumer 2 also bought celery on December 3, 2000, we would consider that to be 4 separate transactions. An easy way to do this is to rename each CUSTOMER_ID as "DATE-CUSTOMER_ID". For example, if CUSTOMER_ID is 12321, and this customer bought apples November 14, 2000, then their new ID is "11/14/00-12321"
4. Make sure each line in the CSV file is "DATE-CUSTOMER_ID1, PRODUCT_ID1".
5. The header of CSV file should be "DATE-CUSTOMER_ID, PRODUCT_ID"

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

(please note DATE-CUSTOMER_ID and PRODUCT_ID are strings and integers, respectively)

DATE-CUSTOMER_ID	PRODUCT_ID
12/1/00-868266	8888215880127
12/1/00-1199833	648636100154
12/1/00-249713	4710032500879

Figure: customer_product file

Do NOT submit the output file of this data preprocessing step, but your code is allowed to create this 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 the time from reading the file till writing the results to the output file. You are asked to find the candidate and frequent itemsets (**similar to the previous task**) using the file you just generated. The following are the steps you need to do:

1. Reading the customer_product CSV file in to RDD and then build the case 1 market-basket model
2. Find out qualified customers-date who purchased more than *k* items. (*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.

Command line Format:

```
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>
```

Command line Example:

```
/opt/spark/spark-3.1.2-bin-hadoop3.2/bin/spark-submit --executor-memory 4G --driver-memory 4G task2.py
20 50 ../resource/asnlib/publicdata/ta_feng_all_months_merged.csv task2_output.txt
```

6. Evaluation Metric

Task 1:

Input File	Case	Support	Runtime (sec)
small1.csv	1	4	<=200
small2.csv	2	9	<=100

Task 2:

Input File	Filter Threshold	Support	Runtime (sec)
ta_feng_all_months_merged.csv	20	50	<=500

5. Grading Criteria

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

1. You can use your **free 5-day extension** separately or together
    a. https://forms.gle/edH8jw1mJjrLFRcm8
    b. This form will record the number of late days you use for each assignment. We will not count late days if no request is submitted. Remember to submit the request **BEFORE the deadline**.
2. There will be a 10% bonus if you use both Scala and Python.
3. We 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. If plagiarism is detected, there will be no point for the entire assignment and we will report all detected plagiarism.
4. All submissions will be graded on the Vocareum. Please strictly follow the format provided, otherwise you can't get the point even though the answer is correct.
5. If the outputs of your program are unsorted or partially sorted, there will be a **50% penalty**.
6. If you use Spark DataFrame, DataSet, sparksql, there will be a **20% penalty**.
7. We can regrade your assignments within seven days once the scores are released. **No argument after one week**.
8. There will be a **20% penalty for late submission** within a week and **no point** after a week.
9. Only when your results from Python are correct, the bonus of using Scala will be calculated. There is no partial point for Scala. See the example below:

Example situations

Task	Score for Python	Score for Scala	Total
		(10% of previous column if correct)	
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

6. Common problems causing fail submission on Vocareum/FAQ

(If your program runs seems successfully on your local machine but fail on Vocareum, please check these)

1. Try your program on Vocareum terminal. Remember to set python version as python3.6,
```
export PYSPARK_PYTHON=python3.6
```
And use the latest Spark
```
/opt/spark/spark-3.1.2-bin-hadoop3.2/bin/spark-submit
```

2. Check the input command line format.

3. Check the output format, for example, the header, tag, typo.
4. Check the requirements of sorting the results.
5. Your program scripts should be named as task1.py task2.py.
6. Check whether your local environment fits the assignment description, i.e. version, configuration.
7. If you implement the core part in python instead of spark, or implement it in a high time complexity way (e.g. search an element in a list instead of a set), your program may be killed on the Vocareum because it runs too slow.
8. You are required to only use Spark RDD in order to understand Spark operations more deeply. You will not get any points if you use Spark DataFrame or DataSet. Don't import sparksql.
9. Do not use Vocareum for debugging purposes, please debug on your local machine. Vocareum can be very slow if you use it for debugging.
10. Vocareum is reliable in helping you to check the input and output formats, but its function on checking the code correctness is limited. It can not guarantee the correctness of the code even with a full score in the submission report.
11. Some students encounter an error like: **the output rate …. has exceeded the allowed value ….bytes/s; attempting to kill the process.**

    To resolve this, please remove **all print statements** and set the Spark logging level such that it limits the logs generated - that can be done using sc.setLogLevel . Preferably, set the log level to either **WARN** or **ERROR** when submitting your code.