INF 553 – Spring 2018
Assignment 2: Frequent Itemsets

Deadline: 02/26 2018 11:59 PM PST

Assignment Overview

This assignment contains one main algorithm. You will implement the SON algorithm using the Apache Spark Framework. You will use three different datasets, ranging from very small to very large. This will help you to test, develop and optimize your algorithm given the number of records at hand. More details on the structure of the datasets and instructions on how to use the input files will be explained in details in the following sections. The goal of this assignment is to make you understand how you can apply the frequent itemset algorithms you have learned in class on a large number of data and more importantly how you can make your implementation more performant and efficient in a distributed environment.

Environment Requirements

Python: 2.7 Scala: 2.11 Spark: 2.2.1

IMPORTANT: We will use these versions to compile and test your code. If you use other versions, there will be a 20% penalty since we will not be able to grade it automatically.
You can only use Spark RDD.

Write your own code!

For this assignment to be an effective learning experience, you must write your own code!
Do not share code with other students in the class!!
Here's why:

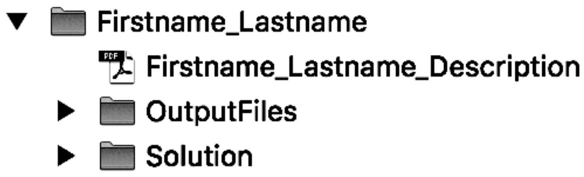
- The most obvious reason is that it will be a huge temptation to cheat: if you include code written by anyone else in your solution to the assignment, you will be cheating. As mentioned in the syllabus, this is a very serious offense, and may lead to you failing the class.
- However, even if you do not directly include any code you look at in your solution, it surely will influence your coding. Put another way, it will short-circuit the process of you figuring out how to solve the problem and will thus decrease how much you learn.

So, just don't look on the web for any code relevant to these problems. Don't do it.

Submission Details

For this assignment you will need to turn in a Python, Java, or Scala program depending on your language of preference. We will test your code using the same datasets but with different support thresholds values. This assignment will surely need some time to be implemented so please plan accordingly and start early!

Your submission must be a .zip file with name: **<Firstname>_<Lastname>_hw2.zip**. The structure of your submission should be identical as shown below. The Firstname_Lastname_Description.pdf file contains helpful instructions on how to run your code along with other necessary information as described in the following sections. The *OutputFiles* directory contains the deliverable output files for each problem and the *Solution* directory contains your source code.



Firstname_Lastname
Firstname_Lastname_Description
OutputFiles
Solution

SON Algorithm

In this assignment we implement the SON Algorithm to solve every problem (Problems 1 and 2) on top of Apache Spark Framework. We will rely on the fact that SON can process chunks of data in order to identify the frequent itemsets. You will need to find **all the possible combinations of the frequent itemsets** for any given input file that follows the format of the Amazon Review Datasets. In order to accomplish this task, you need to read Chapter 6 from the Mining of Massive Datasets book and concentrate on section 6.4 – Limited-Pass Algorithms. Inside the Firstname_Lastname_Description.pdf file we need you to describe the approach you used for your program. Specifically, in order to process each chunk which algorithm did you use, A-Priori, MultiHash, PCY, etc…

At the end of the assignment, Appendix A provides some more guidelines that will help you with the implementation and Appendix B specifies how to organize your Description pdf file.

For assignment 1 you used the Spark framework and most probably at this point you have a better understanding of the MapReduce operations. You can write your program in Python, Java or Scala. For this assignment you will need to find the collection of frequent itemsets of rated products using the Amazon Review dataset with which you are already familiar from homework 1. You will need to compute the frequent itemsets using SON algorithm, initially for a **synthetic testing** dataset (Problem 1), then for the Amazon Review dataset **(5-core review data of category Books and Beauty)** dataset (Problem 2).

We will provide you with csv files which come from the original 5-core json files. You can find all the data set needed from the */Data* directory.

We would like to compute two cases of possible frequent itemsets using the testing and books.csv, beauty.csv files.

__Case 1__

We would like to calculate the combinations of frequent products (as singletons, pairs, triples, etc.) that were **rated** and are **qualified as frequent given a support threshold value**.

In order to apply this computation, we will need to create a basket for each reviewer containing the ids of the products that were rated by this reviewer. If a product was rated more than one time from a reviewer, we consider that this product was rated only once. More specifically, the product ids are unique within each basket are unique. The generated baskets are similar to:

reviewer1 = (product$_{11}$, product$_{12}$, product$_{13}$, …)
reviewer2 = (product$_{21}$, product$_{22}$, product$_{23}$, …)
reviewer3 = (product$_{31}$, product$_{32}$, product$_{33}$, …)
…

__Case 2__

In the second case we want to calculate the combinations of frequent reviewers(as singletons, pairs, triples, etc.) that can be qualified as frequent given a support threshold value.

In order to apply this computation, we will need to create a basket for each product containing the ids of the reviewers who rated this product. If a product was rated more than one time from a reviewer, we consider it was a rated only once by this reviewer. More specifically, the reviewer ids are unique within each basket. The generated baskets are similar to:

product1 = (reviewer$_{11}$, reviewer$_{12}$, reviewer$_{13}$, …)
product2 = (reviewer$_{21}$, reviewer$_{22}$, reviewer$_{23}$, …)
product3 = (reviewer$_{31}$, reviewer$_{32}$, reviewer$_{33}$, …)
…

Finally, in the section Problem 1, we will describe explicitly how you should run your program, and what should be the format of your expected output. Everything that is described in section Problem 1 must be applied to the subsequent sections as well (i.e., Problem 2)

Problem 1 (20 Points)
Implementing SON using Spark with the Testing Dataset

Under the */Data* folder of the assignment you will find two small sample datasets. The *Data/small1.csv* dataset can be used to verify the correctness of your implementation. We will also require you to submit, for each of the two above cases, one output for evaluation for the *Data/small2.csv* dataset, as described in the following Deliverables section.

<u>**Execution Requirements**</u>

Input Arguments:

1. **Case Number:** An integer value specifying which case from the ones we just described we want to compute the frequent itemsets. The input is an integer value. **1 for case 1** and **2 for case 2**.

1. **Input.csv:** This is the path to the input ratings file containing all the transactions. Each line corresponds to a transaction. Each transaction has items that are comma separated. For Problem 1 you can use the *Data/small1.csv* file to test the correctness of your algorithm.

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

Output:
A file in the format shown in the snapshot of the Execution Example section below. In particular, for each line you should output the frequent itemsets you found for the current combination followed by an empty line after each combination. The printed itemsets must be sorted in **lexicographical order** (Both reviewerID and productID are type of string). A high level description of this format is:

(frequent_singleton1), (frequent_singleton2), …, (frequent_singletonK)

(frequent_pair1), (frequent_pair2), …, (frequent_pairM)

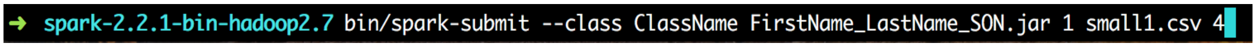
(frequent_triple1), (frequent_triple2), …, (frequent_tripleN)
…

Execution Example

The first argument passed to our program (in the below execution) is the case number. The second input is the path to the ratings input file, and the third is the support threshold value. Following we present examples of how you can run your program with spark-submit both when your application is a Java/Scala program or a Python script.

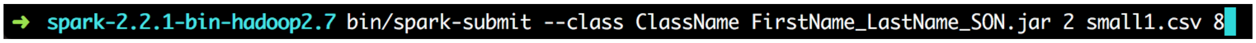
A. **Example of running a Java/Scala application with spark-submit:**
Notice that the argument class of the spark-submit specifies the main class of your application and it is followed by the jar file of the application.

For Case 1



```
➜ spark-2.2.1-bin-hadoop2.7 bin/spark-submit --class ClassName FirstName_LastName_SON.jar 1 small1.csv 4
```

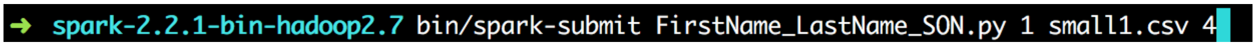
For Case 2



```
➜ spark-2.2.1-bin-hadoop2.7 bin/spark-submit --class ClassName FirstName_LastName_SON.jar 2 small1.csv 8
```

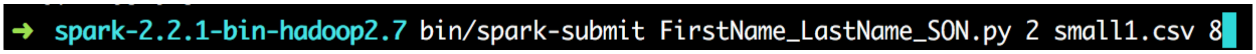
B. **Example of running a Python application with spark-submit:**

Case 1



```
➜ spark-2.2.1-bin-hadoop2.7 bin/spark-submit FirstName_LastName_SON.py 1 small1.csv 4
```

Case 2



```
➜ spark-2.2.1-bin-hadoop2.7 bin/spark-submit FirstName_LastName_SON.py 2 small1.csv 8
```

The solution of the above execution for case 1 is similar to the following snapshot. Since both the product ids and the reviewer ids are strings, the format of the output will be the same in both cases:

Solution of A.Case1 and B.Case1 Snapshots, with input case number 1 , input file small1.csv and support threshold equal to 4:

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

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

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

Deliverables for Problem 1

1. **Script or Jar File and Source Code**
   Please name your Python script as: <firstname>_<lastname>_SON.py.
   Or if you submit a jar file as: <firstname>_<lastname>_SON.jar.

   The python script or the .jar file of your implementation should be inside the *Solution* directory of your submission. You must also include a directory, any directory name is fine, with your source code inside *Solutions*.

2. **Output Files**
   We need two output files for Problem 1.
   For case 1, run your program against *Data/small2.csv* dataset with support 3.
   For case 2, run your program against *Data/small2.csv* dataset with support 5.

The format of the output should be exactly the same as the above snapshot for both cases. The names of the output files should be as:
<firstname>_<lastname>_SON_Small2.case1.txt
<firstname>_<lastname>_SON_Small2.case2.txt

The above output files should be placed inside the *OutputFiles* directory of your submission.

3. **Description**
Inside the Firstname_LastName_Description.pdf document please write the command line that you used with spark-submit in order to run your code. Specify also the Spark version that you use to write your code. If it is a jar file, please specify the name of the main class of your app as shown in the above snapshots. We will use this in order to rerun your code against different support values if needed.

Problem 2 (80 Points)
Implementing SON using Spark with the Amazon Review Data

The requirements for Problem 2 are similar to Problem 1. However, here we would like to **check for the performance of our implementation using a larger dataset**. We would like to find the frequent itemsets from a larger number of records. For this purpose, a good indicator of how well our algorithm works is the total execution time. For this execution time, **we take into account also the time of reading the files from the disk.** Following, we provide a table of execution time for two threshold values for each file described in the first section. You can use this table as an evaluation metric of your implementation.

File Name	Case Number	Support	Runtime (sec)
beauty.csv	1	50	≤ 1000
beauty.csv	2	40	≤ 500
books.csv	1	1200	≤ 1500
books.csv	2	1500	≤ 240

<u>Deliverables for Problem 2</u>

1. **Output Files**
We need four output files for Problem 2.
The format of the output should be exactly the same as the one for Problem 1.
The output files should be named as:
<firstname>_<lastname>_SON_Beauty.case1-50.txt
<firstname>_<lastname>_SON_Beauty.case2-40.txt
<firstname>_<lastname>_SON_Books.case1-1200.txt
<firstname>_<lastname>_SON_Books.case2-1500.txt

The above output files should be placed inside the *OutputFiles* directory of your submission.

2. **Description**

   Inside the Firstname_LastName_Description.pdf document of your submission please include a table that is exactly the same with the one provided on the top of this section. You must use the same support threshold values as the table above and include the execution times of your implementation for each case. We will run your code so make sure the times you record on the table are the ones corresponding to your implementation.

Grade breakdown

a. Four correct output files (10pts each)

b. Your execution time needs to be smaller than the ones in the table (10pts each)

General Instructions:

1. Make sure your code compiles before submitting
2. Make sure to follow the output format and the naming format.

Grading Criteria:

1. If your programs cannot run with the commands you provide, your submission will be graded based on the result files you submit and 20% penalty for it.
2. If the files generated by your programs are not sorted based on the specifications, there will be 20% penalty.
3. If your program generates more than one file, there will be 20% penalty.
4. **If you don't provide the source code and just the .jar file in case of a Java/Scala application there will be 60% penalty.**
5. **If your submission does not state inside the Description pdf file how to run your code, which Spark version you used and which approach you followed to implement your algorithm there will be a penalty of 30%.**
6. There will be 20% penalty for late submission within a week and 0 grade after a week.
7. You can use your free 5-day extension.
8. There will be 10% bonus if you use Scala for the entire assignment.
9. **There will 0 grade if you use Dataframe or Dataset.**

APPENDIX A

- You need to take into account the Monotonicity of the Itemsets
- You need to leverage Spark capabilities of processing partitions/chunks of data and analyze the data within each partition.
- You need to reduce the support threshold according to the size of your partitions.
- You should emit appropriate (key, value) pairs in order to speed up the computation time.
- Try to avoid data shuffling during your execution.

Pay great attention on the thresholds number for each case. The lower the threshold the more the computation. Do not try arbitrary threshold values. Try testing values within the given ranges.

APPENDIX B

Please include the following information inside your description document.

- Succinctly describe your approach to implement the algorithm.
- Command line command to execute your program
- Problem 2 execution table