Homework 2

A succinct summary of how to run the code, which version of Python, Scala & Spark are used and approach followed to implement the algorithm.

Solution

- The aim of this assignment is finding combination of frequent itemsets for the MovieLens dataset. I have used Apriori and SON algorithm to develop my solution.
- Following is a step by step description of the implementation.
    - Based on the case number, RDD is created by reading the input CSV file using the groupByKey so that an RDD conforming to the required format is created.
    - Case 1 -> (movie1, movie2, movie3)
    - Case 2 -> (user1, user2, user3)
- The implementation comprises of multiple map reduce tasks. Initially I implemented the apriori algorithm for mining frequent itemsets.
- Candidates were generated using RDD operations such as `map`, `cartesian product`, `groupby` , set unions and filter.
- After candidates are generated, frequencies of these items are calculated in various input chunks using `map partition` and using partition support threshold they are filtered. This is the 2nd map phase of the SON algorithm.
- After computing candidate frequencies in various input chunks, the reduceByKey RDD operation is used to sum up the candidates occurances which is the 2nd reduce phase of the SON algorithm.
- Itemsets that do not have total support at least equal to support threshold are not transmitted to the output of the Reduce task.
- In order to avoid data shuffling, I have used map, filter and union and coalesce to reduce the number of partitions along with minimizing data shuffling. The code can be further optimized by using *aggregate*ByKey instead of reduceByKey which also minimizes shuffling.

Versions

```
Spark - 2.2.1
Scala - 2.11
Python - 2.7
```

Implementation Steps

Python

```
spark-submit Piyush_Umate_SON.py <case_number> <file_path> <support>
```

Scala

```
spark-submit --class FrequentItemsetsSON Piyush_Umate_SON.jar <case_number>
<file_path> <support>
```

Execution Times

Problem 1

Small2 (execution time in seconds)

	Case 1		Case 2	
	Support Threshold	Execution Time	Support Threshold	Execution Time
Python	3	21	5	12
Scala	3	17	5	11

Problem 2

MovieLens Small – Python (execution time in seconds)

Case 1		Case 2	
Support Threshold	Execution Time	Support Threshold	Execution Time
120	6.45	180	18.259
150	5.42	200	13.55

Problem 3

MovieLens Big - Python (execution time in seconds)

Case 1		Case 2	
Support Threshold	Execution Time	Support Threshold	Execution Time
30000	95	2800	88.05
35000	83.16	3000	84.95

Problem 2

MovieLens Small - Scala (execution time in seconds)

Case 1		Case 2	
Support Threshold	Execution Time	Support Threshold	Execution Time
120	6.2	180	14.1
150	5.1	200	11

Problem 3

MovieLens Big - Scala (execution time in seconds)

Case 1		Case 2	
Support Threshold	Execution Time	Support Threshold	Execution Time
30000	81	2800	113
35000	74	3000	103

```
Output file will be generated in the same folder in which the code is executed.
```