Vijayakumar Gedigeri INF553 HW2, 6th March 2017

1. Implementation

The assignment is done in Python language and uses pyspark library for Spark functionalities. Program is written for Python version 2.7 and Spark version 2.0.1. This implementation follows SON algorithm (Map Reduce paradigm) to find out the frequent item sets of all possible sizes in each partition (Chunk) the main data set is divided into. 2 phase of SON algorithm is employed here where in the first Map-Reduce phase, all possible candidate item sets are determined. In second phase of Map Reduce, the counts of each candidate item set from each partition are added, and those having support less than the actual support passed through command line are eliminated. Remaining item sets are declared as frequent and are written to the file.

In order to generate candidate item sets, Apriori algorithm is used. Monotonicity of the itemsets is used to generate candidate itemsets of next size. Only singletons that are frequent are considered for generating pairs, and likewise triples are generated from frequent pairs etc. Python library function "itertools.combinations" is used to generate pairs, triples etc. The algorithm is run as part of first Map phase using Spark's mapPartitions method on each chunk of the data that contains baskets of movies or baskets of users. Support threshold for each partition is adjusted based on the chunk size and entire data set size. These candidate itemsets are collected from all partitions and are sent to each partition (One more mapPartitions call) to get the actual count of their presence in baskets in <key, value> pair format; candidate itemset being the key and its count being the value. As part of second Reduce, counts of each key is added and only keys that have count more than the support are declared as frequent.

2. Commands to execute the programs

My machine has default setting for –master local[4] and –driver-memory as 5g. So, I don't pass these parameters via command line. If this isn't the case in your system, please pass above parameters as well.

Problem 1 with small data set

a. Case 1
```
bin/spark-submit Vijayakumar_Gedigeri_SON.py 1 Data/movies.small2.csv 3
```
b. Case 2
```
bin/spark-submit Vijayakumar_Gedigeri_SON.py 2 Data/movies.small2.csv 5
```

Problem 2: With ml-latest-small data set

a. Case 1 with support 120
   bin/spark-submit Vijayakumar_Gedigeri_SON.py 1 ml-latest-small/ratings.csv 120
b. Case 1 with support 150
   bin/spark-submit Vijayakumar_Gedigeri_SON.py 1 ml-latest-small/ratings.csv 150
c. Case 2 with support 180
   bin/spark-submit Vijayakumar_Gedigeri_SON.py 2 ml-latest-small/ratings.csv 180
d. Case 2 with support 200
   bin/spark-submit Vijayakumar_Gedigeri_SON.py 2 ml-latest-small/ratings.csv 200

Execution table

Case 1		Case 2	
Support	Execution Time	Support	Execution Time
120	42 secs	180	312 secs
150	21.37 secs	200	122.16 secs

Problem 3: With ml-20m data set

a. Case 1 with support 29000
   bin/spark-submit Vijayakumar_Gedigeri_SON.py 1 ml-20m/ratings.csv 29000
b. Case 1 with support 30000
   bin/spark-submit Vijayakumar_Gedigeri_SON.py 1 ml-20m/ratings.csv 30000
c. Case 2 with support 2500
   bin/spark-submit Vijayakumar_Gedigeri_SON.py 2 ml-20m/ratings.csv 2500
d. Case 2 with support 3000
   bin/spark-submit Vijayakumar_Gedigeri_SON.py 2 ml-20m/ratings.csv 3000

Execution table

Case 1		Case 2	
Support	Execution Time	Support	Execution Time
29000	685 secs	2500	630 secs
30000	590 secs	3000	355.65 secs

3. Output
   Frequent itemsets of all sizes are written to file. The name of file that gets created is shown at the end of program execution.
   For e.g.
   bin/spark-submit Vijayakumar_Gedigeri_SON.py 1 ml-20m/ratings.csv 29000

   For above input, output file will be created as “output_case1_s29000.txt”

   Output files for all above inputs are available in OutputFiles directory.

4. Bonus question: Why the large support threshold was used for big data set?
   Answer: ml-20m data set is a very large data set and contains millions of records. There is a total of around 140,000 baskets each containing movies rated by the users (When grouped). These baskets are divided into equal partitions of each size around 8855 per chunk. An item can appear in most of these baskets, and its support in entire data might come as close to as the number of baskets. But, realistically, items having such high support are very less. When support is low, most of the items that do not appear in most baskets will be identified as frequent item sets, and number of such item sets will be too large. With a large support as 29000 or 30000, we are trying to limit the support to 1812 to 1875 per each chunk so that number of items that satisfy this support are less in number. Since we use property of Monotonicity of frequent items, by limiting the number of singletons first with such a high support, we can decrease the number of pairs that are subsequently going to be generated and so on. That way, program can find frequent item sets in a time of 10 minute or so from this very huge data set. If the support is way less like 10,000 or 15000, the number of item sets going to be declared as frequent will be huge in number, and it would require a lot of time to find such items. Apriori will take a long time to generate the candidate sets as the item sets appearing with less support will be more. If the candidate sets are more, it’s load on second map phase where they need to be counted. So, eventually, lot of time is spent in finding the true frequent item sets.