Description File

Java – 1.8
Scala – 2.11
Spark – 2.3.1
Apriori Algorithm

Approach

The main class name is task1.scala in the JAR file Ashir_Alam_SON.jar. The algorithm reads a .csv file (test, small, large) to an RDD and groups the data based on the key(words) to get the baskets. Depending on the threshold value, we use MapPartitions function to divide the baskets into segments. We then call **Apriori** algorithm on each segment to find the candidate frequent itemsets with support equal to support/number of partitions. The reducebykey then works to remove the redundant values. We use hashmap to count the occurrence of each basket and filter the frequent item sets.

Command Line

spark-submit --driver-memory 4g --class task1 Ashir_Alam_SON.jar <input path> <support>
E g:-

```
spark-submit --driver-memory 4g --class task1 Ashir_Alam_SON.jar
"/Users/ashiralam/Downloads/inf553_assignment3/Data/yelp_reviews_small.txt" 1000
```

Problem 2:

.small.txt	1000	5 sec
.small.txt	500	15 sec

Problem 3:

.large.txt	100000	159 sec
.large.txt	120000	116 sec

Bonus

Given the size of the file large.txt, using a small threshold could result in a stack overflow and memory overflow errors. This might also lead to a program running for a long time and we cannot find the frequent itemsets in correct way. Increasing the threshold from 100000 to 120000 leads to a significant decrease in the execution time as the counts were reduced. Also, if we choose a small threshold it generates a large number of words and working on them might be difficult.
Maybe if we increase the size of file further to a huge extent such that the segments become large enough to be handled by a local machine, then it can limit the working of the algorithm in this machine.