Description

Yu Ru   ID:6847599249

Attention: Because when you transfer word to pdf, there could be some spaces in the document. So if you want to paste the command line, you should remove all the useless spaces before that. Thanks

1. Succinctly describe your approach to implement the algorithm.

   In this assignment, I design my SON algorithm with A-priori algorithm in each chunk.

   In my algorithm, I first use Spark map functions to filter the structured data, and then build the basket with the format (key,[all items in the basket]). After that I make combination for each basket, count the frequency of each item (single, double etc), split baskets into chunks, calculate the threshold for each chunk based on the basket numbers. I modify the threshold to 1/p-1 based on the data size. Also, I try to split as less chunk as I can to prevent false negative (because the number of frequent items in each chunk could differ a lot and the mission of this assignment is to find **ALL** frequent items). I filter the items based on the threshold in each chunk. This is the first pass SON algorithm.

   In the second pass, I merge all chunks together and filter all items with the total threshold to get the frequent item set.

   After that, I count all the frequent items appear in current size. Join them with the whole data to filter the data for next size of item sets.

2. Problem 1.

   Command line: The command lines are:

   Case1: ./bin/spark-submit /Users/Yu/Downloads/ Yu_Ru_SON.py 1 '/Users/Yu/Downloads/Assignment2/Data/movies.small2.csv' 3

   Case2: ./bin/spark-submit /Users/Yu/Downloads/ Yu_Ru_SON.py 2 '/Users/Yu/Downloads/Assignment2/Data/movies.small2.csv' 4
   Version: spark-1.6.1-bin-hadoop2.4

3. Problem 2.
   Command line: The command lines are:
   Case 1:
   Threshold 120: ./bin/spark-submit /Users/Yu/Downloads/ Yu_Ru_SON.py 1 '/Users/Yu/Downloads/ml-latest-small /ratings.csv' 120
   Threshold 150: ./bin/spark-submit /Users/Yu/Downloads/ Yu_Ru_SON.py 1 '/Users/Yu/Downloads/ml-latest-small/ratings.csv' 150
   Case 2:
   Threshold 180: ./bin/spark-submit /Users/Yu/Downloads/ Yu_Ru_SON.py 2 '/Users/Yu/Downloads/ml-latest-small/ratings.csv' 180
   Threshold 200: ./bin/spark-submit /Users/Yu/Downloads/Yu_Ru_SON.py 2 '/Users/Yu/Downloads/ml-latest-small/ratings.csv' 200

   Run time:
   Case 1: Threshold 120: 25 sec. Threshold 150: 13 sec.
   Case 2: Threshold 180: 420 sec. Threshold 200: 120 sec.
4. Problem 3:
   Command line: The command lines are:
   Case 1:
   Threshold 29000: ./bin/spark-submit /Users/Yu/Downloads/Yu_Ru_SON.py 1 '/Users/Yu/Downloads/ml-20m /ratings.csv' 29000
   Threshold 30000: ./bin/spark-submit /Users/Yu/Downloads/ Yu_Ru_SON.py 1 '/Users/Yu/Downloads/ml-latest-small/ratings.csv' 30000
   Case 2:
   Threshold 2500: ./bin/spark-submit /Users/Yu/Downloads/ Yu_Ru_SON.py 2 '/Users/Yu/Downloads/ml-20m /ratings.csv' 2500
   Threshold 3000: ./bin/spark-submit /Users/Yu/Downloads/ Yu_Ru_SON.py 2 '/Users/Yu/Downloads/ml-20m /ratings.csv' 3000

Run time:
Case 1: Threshold 29000: 34 min. Threshold 30000: 32 min.
Case 2: Threshold 2500: 602 sec. Threshold 3000: 482 sec.

About the Bottle Neck:
I think the reason we have to set a large threshold are:

1. If the threshold is small, the execution time would be really slow because we need to deal with too much data in each pass.
2. When the threshold is small in a big data set, the result sometimes would not be representative. For example, if we set the threshold as 4 in a 20-item basket, the result could represent something since it appears more than 20 percent. But if we are mining 200000-item basket and still use threshold 4, the result could almost be everything!

The bottle neck in my execution I think is that I need to make combinations in each basket and the run time complexity in the basket with size of K items would be O(n^K). So when n is huge, it's really horrible.