Simple Randomized algorithm and SON algorithm

The implementation of Simple randomized algorithm

Step1: Set a class to offers two method: findFrequentItemsets (in the provided input file) and findAssociations (within the frequent itemsets)
Step2: Create a map of provided baskets and items, then set of real frequent items and set of candidates.
Step3: Create prepare function, The file has to be as follow: One line per basket, the items in one basket are integers, the items are separated by a space. We need to generate a list of candidate singletons, get the frequent singletons and iterate over the candidates to filter them.
Step4: Find the frequent itemsets and generates the candidates
Step5 : Find associations
Step6: Combination function –
This utility class computes the combination of item in a set. A combination is intended as a selection K of items from a collection of lengh N (the inputSet)
Step7: Output the results of the implemented Apriori algorithm

The implementation of SON algorithm

Steps:

1. File Split

As a result of the need to average the original input file to every mapper, so the Hadoop default TextInputFormat can not be used; although there is a NLineInputFormat, each mapper can be responsible for the file split of the N line, but each record generated in this way is a row of data, representing only one pen for each call map (). Information can not operate on the entire file. So I also write the MultiLineInputFormat inheritance NLineInputFormat, and the nextKeyValue () method of override's RecordReader, add a loop that executes the N time, and change the value of each record to the N row data, and the key value becomes the number of rows of this file split.

Job 1:

Map Function:

For the N line in the input file, the Apriori operation is performed on it, and each transaction is read into the ArrayList<TreeSet<Integer>>, and the number of singletons is calculated. I use data structures such as TreeMap, TreeSet, and not HashMap, HashSet because they have sort functions and store data with red and black trees, so the data is only O (log (n)) based on the key value; I store all candidate itemsets in TreeMap<Integer, TreeMap<String, format. In order to perform the operation of the collection, the String of the k- tuple is converted to array, and then to TreeMap, the actions of the subset and the subset can be combined and judged by the add () and containsAll () methods in Set.

Then frequent singlestons is used to compute candidate pairs, and candidate triples is calculated using frequent pairs and frequent singletons. When the number is not up to the threshold, the threshold value is threshold frequency multiplied by the number of lines of this

file split, K will increase to the user's given value, or when all candidate k-tuples is unable to reach the threshold, then all candidate itemsets is passed to reducer.

Reduce Function:

Write the candidate itemsets generated by Mapper to the file.

Job 2:

Map Function:

Need to start with all the transactions's input files and the output files produced in the last phase, so the override setup () method allows each mapper to read the candidate itemsets from Hadoop filesystem to TreeMap<String, Integer>. Then calculate the number of times each itemset appears in the sub file, and if more than 0 times, the number will be transmitted to reducer.

Reduce Function:

Statistics the number of times that all candidate itemsets appear, and exclude those who do not reach the threshold, and finally get the real frequent itemsets.