Wan D. Bae - Data Science (ITE4005)
컴퓨터전공 2013012289 한기훈
Mar 30th, 2017

Report for Assignment1 - Apriori

1. Summery of algorithm.

The Apriori algorithm I implemented follows the following procedure.

1. Read the transaction data from the input file.
2. Create a frequent set with one element.
3. Use the Apriori algorithm to create a frequent set of 2, 3, 4, ..., N elements.
4. Increase N until no more frequent sets are created.
5. Create an Association rule using the created frequent set.
6. Record the generated rule in the output file according to the given format.

2. Detailed description of code.

This code has three major functions. (Note: L means list of frequent sets.)

A. `getTransactions`

Read transaction data from a given input file. Since each line represents each transaction, we put the elements on the same line into a set, and then manage this set in a vector data structure.

For example, if you have 1 line, and you have `1, 3, 4, 5` there, make it `{1,3,4,5}` and put it in the vector data structure. This function returns a list of Transaction data sets.

Return type is `vector< set<int> >`

B. `generateFrequentSets`

Data structure `vector< map< set<int>, int > > L` exists.

The map stores the frequent set as the first argument and the absolute support count as the second argument.

L_n is represented by L [n] in the code. Therefore, L.resize (2) was performed to express the first L [1].

The for-loop then proceeds to create L [1]. Transaction data has its own unique elements, which counts the total number of elements.
If the count result satisfies the given minimum support condition, put it in L [1].

Next, there are three functions. (Expressed in C ++ 11 lambda function and auto type)

- `hasInfreqSubset`

The first argument is the target set, and the second argument is the list of the frequent set that was created just before.

If the size of the given set as the first argument is N, then the L given as the second argument is L [N - 1].

This function creates one subset of size 1 from the set given as the first argument, and checks to see if the subset exists in the second given argument, L. If it does not. exist, the target set means that it contains an infrequent set, and the corresponding set is also an infrequent set.

- `aprioriGenCandidate`

This function creates a Candidate set to create the next L based on the most. recently created L. In the most recent L, two different sets are extracted and a new set is created by doing Cartesian Product operation. In this case, if the size of the new set is one greater than the size of the existing set, check whether there is frequent set using `hasInfreqSubset` defined immediately before. If it is a frequent set, put the new set into the Candidate set.
Type `set< set<int> >` was used to store the Candidate sets.

- `checkSubset`

This function is given two arguments. Both are given a set and check whether the. second set is a subset of the first set.

Next, there is a for-loop.
Starting from L [2], L [k] is generated until it is no longer possible to create a frequent set. Inside the loop, the following process occurs.

- Create a candidate set
- Create a list of frequent sets of size k using the corresponding candidate and insert it into L [k]. (Note that L [k] is not the finished L [k]),
- Check the support count while traversing the incomplete L [k] to remove the infrequent set (ie, infrequent set)
- If L [k] is empty as a result of the removal operation, escape the for-loop because there is no frequent set of size k.

The last for-loop was created to gather all the generated frequent sets in one place.

C. `putAssociationRules`

At first we have the getAllSubset function. This function creates all subsets of a given set when given as an argument. (Except for the empty set.)

Next, it is a code that shows how to create an association rule between subset in the corresponding set in each frequent set, and record it in the output file according to the given format.

3. How to compile this code.

Just type ‘`make`’ in terminal. Or, please type below line.

```
$ g++ -O2 -o apriori.exe apriori.cpp --std=c++11
```

Above task will generate ‘`apriori.exe`’ file into same directory.

(Compiler must support C++11 standard)

(Compilation tested on Ubuntu 16.04.2 LTS. G++ version is written below.)

```
g++ (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609
```

4. Any other specification.

To avoid Banker’s Rounding problem, I use custom function to generate mathematical rounding results.