Apriori algorithm for association rules discovery

Documentation

Przemysław Rosiński
Piotr Janaszek

Faculty of Mathematics and Information Science
Warsaw University of Technology

29 November 2011

TABLE OF CONTENTS

DESCRIPTION OF THE PROBLEM

The aim of this project is to develop the application for data mining. The functionality is focused on retrieving useful information out of given data. Mainly, there is given the set of transactions, where every transaction is the set of items, and the application is said to discover sets of items which appears frequently. The frequency is given by the user.

Such an application can be used for instance in markets, where transactions will be set of products bought by the customers. In this way the management will know, which sets of products are bought frequently.

BACKGROUND

In this program algorithm used for solving the problem is apriori algorithm for association rules discovery.

Used algorithm

Apriori algorithm is a classical and well-know method for discovering association rules. It uses bottom-up approach, which means that it starts with one element sets of elements and creates in next steps sets of size increased by one.

The algorithm is done as follows:

1. Retrieve set of transactions *T*; every transaction is a set of items
2. Retrieve *support* number; *support* is a fraction of number of transactions
3. Create set of items, such that every item is an element in at least one transaction
4. Set variable *length* to 1
5. Create set of candidates C_{length}; every candidate is a one element set containing one item
6. While C_{length} is not empty
   a. Create empty set L_{length} of frequent sets of size *length*
   b. For every candidate in C_{length} check if it is frequent, i.e. candidate set is a subset in at least *support* number of transactions; if yes then move it to L_{length}
   c. Increase *length* by 1
   d. Create new set of candidates C_{length} out of $L_{length-1}$

ALGORITHM

Apriori algorithm for association rules discovery is very simple and well-researched. It was proved that it returns all and only frequent sets with given support.

Complexity and correctness analysis

The complexity of the algorithm depends not only on the size of the input, but also on the content of it. In optimistic case, when support number is big enough, it can happen that any of one element candidate set will not be frequent and the computation will stop after one execution of the main loop. On the other hand, in worst-case scenario, when support number is equal to 1 transaction and every transaction contains every item, every subset of set of items must be checked and computation will be as long as variable length will extend number of items.

Let *n* denote number of transactions and *m* denote number of different items. The maximal number of association rules is $O(m2^{m-1})$ and the algorithm execution time complexity is $O(nm2^m)$.

Algorithm has got the stop property. It will stop for sure, when variable length will be greater than number of items, because then the set of candidates will be empty. But in most cases, it stops even in earlier stage.

TECHNICAL ASPECTS

Whole application is written in C# programming language in Visual Studio 2010 IDE. The technology used is Windows Forms. This environment allows creating easy and intuitive in use user interface.

Program imports and exports data using external files. They are of XML format. It is chosen deliberately because of its ease in usage and popularity.

Description of main data structures and procedures

There are three forms in the solution: *MainForm*, *GenerateInputForm* and *SetSupportForm*. In the first one the major of the application activity is taken, for instance generating input or finding frequent sets. Two other forms are called by *MainForm* as dialogs with users. *GenerateInputForm* is used for retrieving data needed to generate transactions. *SetSupportForm* allows user to change support value.

There are also three main sets on which the program operates: *transactions* – set of sets of items (list of transactions), *frequentSets* – set of set of items (list of frequents sets) and *items* – set of items. In the intermediate steps there are also used: *candidates* – set of set of items (list of sets of elements of given size that are candidates to be frequent) and *L* – set of set of items (subset of candidates of given size that are frequent).

There are two important methods worth to be mentioned on this stage. Method *generateTransactions* using three parameters (number of transactions, maximal number of items and maximal number of items in one transaction) creates input transaction and fills transactions set. Method *findFrequent* uses two parameters (list of candidate sets and size) and generates list of frequent sets of given size.

Comments in source

In Appendix A, there is presented summary of methods created in the application. Description was generated using default Microsoft Visual Studio 2010 feature.

External files

Both input file for importing transactions as well as output file for exporting frequent sets are in XML format. Below, there is presented syntax for both files.

Input file

There can be many *items* in one *transaction* and many *transactions* in *transactions*.

```
<?xml version="1.0" encoding="utf-8"?>
<transactions>
      <transaction>
            <item> _ </item>
            <item> _ </item>
            ...
      </transaction>
      <transaction>
            <item> _ </item>
            <item> _ </item>
            ...
      </transaction>
      ...
</transactions>
```

Output file

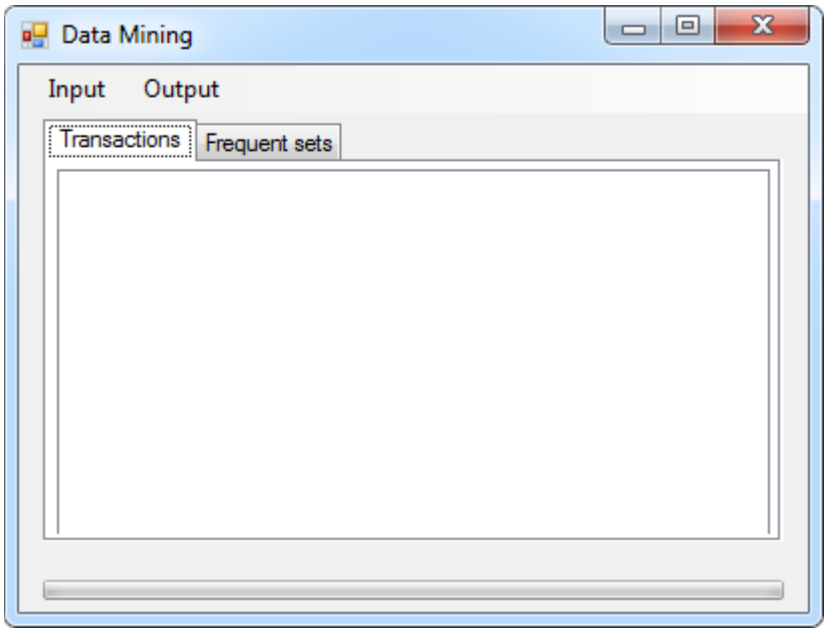
There can be many *items* in one *set* and many *sets* in *frequent*.

```
<?xml version="1.0" encoding="utf-8"?>
<frequent>
      <set>
            <item> _ </item>
            <item> _ </item>
            ...
      </set>
      <set>
            <item> _ </item>
            <item> _ </item>
            ...
      </set>
      ...
</frequent>
```

USER'S MANUAL

Graphical User Interface

After launching the application the following screen appears:



It is a simple window with menu bar at the top with two items in it *Input* and *Output*.

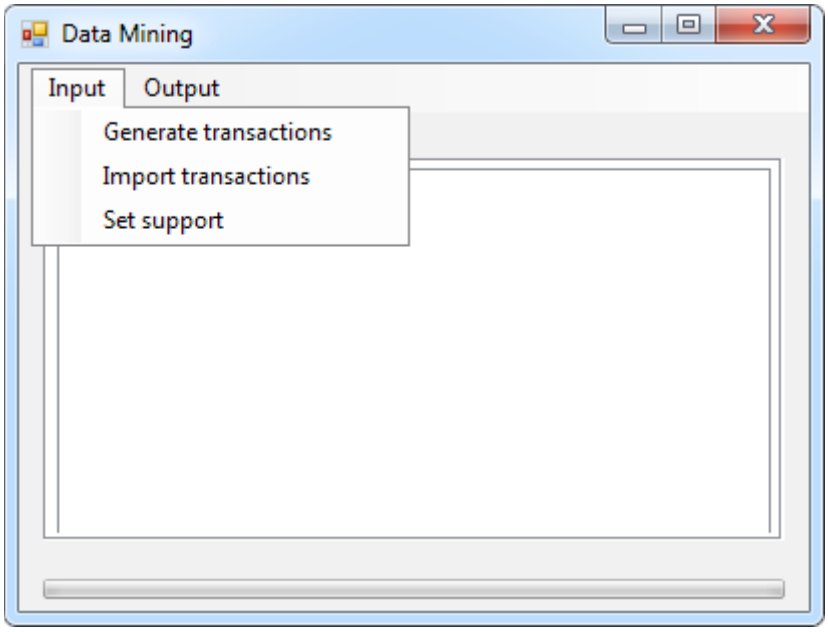
Below, in the center of the window, there are two tabs *Transactions* and *Frequent sets*, they simply shows in the list boxes the content of *transactions* set and *frequentSets* set, respectively.

On the bottom there is a progress bar, above which necessary information about work done in the background is displayed.

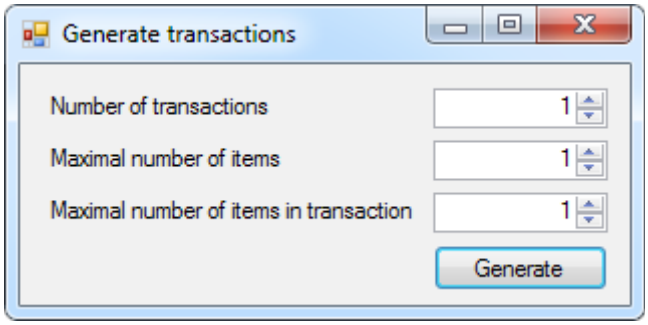
All actions are taken by the users via menu bar.

Menu

There are three options hidden under *Input* menu item:



1. *Generate transactions* option is used for creating set of transactions. In order to do so program needs some additional data and displays to the user another window with field to be filled.

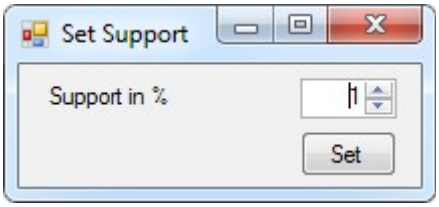


   Note: Every field value must be integer in the range from 1 to 2,147,483,647 (maximal value of int type).

   Note: Because of the fact that in every transaction one item can occur at most one time, *Maximal numbers of items in transaction* cannot be grater than *Maximal number of items*.

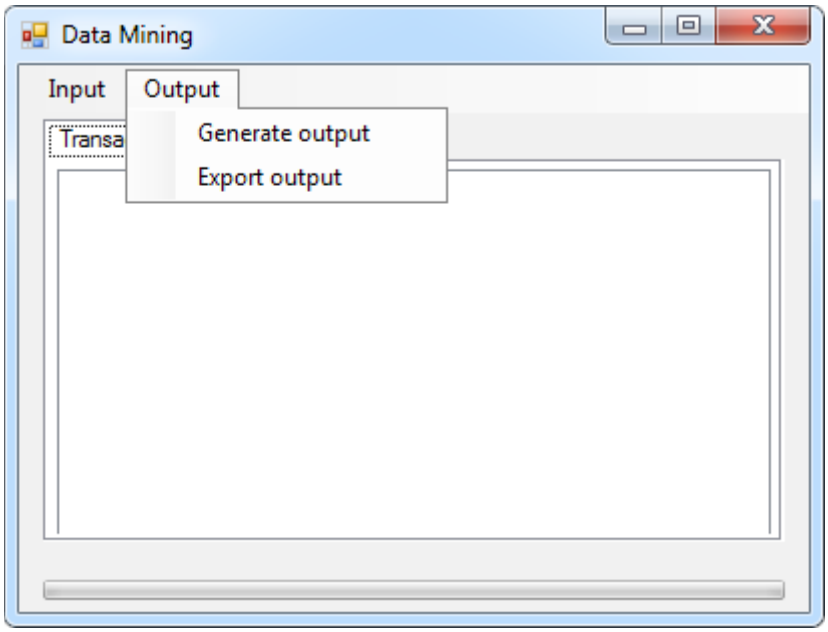
2. *Set support* option launches another window in which user enters support value.

3. *Import transactions* option allows user to import data from external XML file. Standard OpenFileDialog is used for selecting file from disk drive.



Note: Support is expressed in percentage. Value must be integer in the range from 1 to 100.

There are also two options hidden under *Output* menu item:



1. *Generate output* option launches algorithm for finding frequent sets.

2. Export output option allows user to export generated frequent sets to external XML file. Standard SaveFileDialog is used for selecting directory and file name.

How to use

There is one main user path activity.

1. Add transactions to the set by:
   a. generating transactions *(Menu → Input → Generate transactions)* or
   b. loading transactions from external file *(Menu → Input → Import transactions)*.
2. Set support *(Menu → Input → Set support)*.
3. Generate frequent sets *(Menu → Output → Generate output)*.
4. *(optional)* Save frequent sets to external file *(Menu → Output → Export output)*.

TESTS

Every program must be tested, when done. In this section results of tests are presented.

While, it has been proved that the algorithm works correctly and always returns the expected result, the test are done in order to compare computation time.

Data sets

Data sets used in testing were generated using generator developed as a part of the application. This generator is using uniform distribution of elements. Every tested data set consists of four important parameters: number of transactions, maximal number of items, maximal number of items in single transaction and support. Result of testing is time of execution.

Results

Data set	Number of transactions	Number of items	Number of items in transaction	Support	Time
1	1000	max 10	max 7	40%	0.008 s
				30%	0.018 s
				20%	0.023 s
				15%	0.101 s
				12%	0.105 s
				10%	0.106 s
2	20000	max 15	max 10	40%	0.041 s
				30%	0.325 s
				20%	0.309 s
				15%	1.669 s
				12%	1.696 s
				10%	1.640 s
3	50000	max 15	max 10	40%	0.077 s
				30%	0.816 s
				20%	0.759 s
				15%	3.969 s
				12%	3.988 s
				10%	3.961 s
4	100000	max 10	max 8	40%	0.665 s
				30%	0.635 s
				20%	2.359 s
				15%	2.355 s
				12%	5.420 s
				10%	5.438 s
5	100000	max 20	max 15	40%	2.849 s
				30%	3.164 s
				20%	3.698 s
				15%	22.540 s
				12%	22.425 s
				10%	1m 55.694s

Conclusions

It is clearly visible that in every data set there are threshold values of support between which execution times are similar. For example, in data set number 3 time for support 40% is about 77 ms, for 30% and 20% it is 10 times bigger but similar and for 15%, 12% and 10% it is about 4 s. It seems to be the aspect of going deeper into *findFrequent* function with bigger length of the desired set.

In small data sets (few thousands of elements; see data set 1) times of execution are very small, the measure error has big impact, especially when support is big.

On the other hand, in big data sets (hundreds of thousands of elements; see data set 4 and 5), every factor strongly affects the performance time. Data set 5 has about 2 times more elements that data set 4, but in most cases time of performance is more than doubled. Also support must be chosen wisely. Small change, even 2% (from 12% to 10% in data set 5) can make big difference – execution over 5 times longer.

PROJECT SUMMARY

Developed application works correctly even with almost one million of elements in transactions. Utilized environment and solutions were good choices, because application created using them is reliable and easy in use.

Apriori algorithm for association rules discovery is a very good tool for data mining for example in market or stores – handling many records still not makes execution time long.

REFERENCES

Literature

- “Fast Algorithm for Mining Association Rules”, Rakesh Agrawal, Ramakrishnan Srikant, IBM Almaden Research Center
- “Data Mining. Lecture notes”, Krzysztof Bryś, Warsaw University of Technology

Web pages

- www.cs.cmu.edu/afs/cs/academic/class/15721-f01/www/lectures/assoc.pdf
- www-users.cs.umn.edu/~kumar/Presentation/M7-dm-chap4.pdf
- en.wikipedia.org/wiki/Apriori_algorithm
- en.wikipedia.org/wiki/Association_rule_mining