|  |
| --- |
| 哈尔滨工业大学深圳研究生院 |
| **数据挖掘实验报告** |
| 关联规则挖掘算法实验 |
|  |
| **刘岭岭 12S151017** |
| **2012/9/28** |

|  |
| --- |
|  |

关联规则挖掘算法实验

刘岭岭 12S151017

# Date and Set

September 28, 2012 C101

# Objectives

**Frequent patterns** are patterns (e.g., itemsets, subsequences, or substructures) that appear frequently in a data set. For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a frequent itemset. A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a (frequent) sequential pattern. A substructure can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences. If a substructure occurs frequently, it is called a (frequent) structured pattern. Finding frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data. Moreover, it helps in data classification, clustering, and other data mining tasks. Thus, frequent pattern mining has become an important data mining task and a focused themein data mining research.

Association rules is to find the relevance in records so as to make a direct for sales strategy.

The purpose of this experiment is to enable students to integrate theory with practice, master the ability to apply the theoretical knowledge to solve practical problems from the practical problems. Based on a deep understanding of what is frequent patterns and association rules can discover useful patterns in large amounts of data, and at the same time learn to achieve a high-level programming language association rule mining algorithm.

# Instruments

Person Computer:

CPU: at least 1GHz

Memory: at least 1GB

Hard: at least 20GB

Other Equipment: USB flash disk or others

Relation Software:

SPSS Clementine, Weka, Microsoft Visual Studio, Eclipse and so on

# Principles

Use the **Apriori** algorithm to find thefrequent itemsets, based on these itemsets, then generate strong association rules.

Association rule mining process concludes two stages. Stage one is to find **frequent itemsets** in records. Stage two is generate strong **association rules** based on these frequent itemsets produced in stage one.

In the first stage that finding frequent itemsets we should find itemsets that whose occurrences is greater than a threshold called **support threshold**. The Apriori algorithm finds frequent K+1-Itemsets in frequent K-Itemsets, which can save a lot of time in scanning the database. This process until can’t find any frequent itemset.

In the second stage, it uses frequent itemsets produced in the first stage generate association rules according **Minimum Confidence**.

# Datas

1. BASKETS1n(Market shopping dataset)

The dataset contains 18 attributes called *cardid*, *value*, *pmethod*, *sex*, *homeown*, *income*, *age*, *fruitveg*, *freshmeat*, *dairy*, *cannedveg*, *cannedmeat*, *frozenmeal*, *beer*, *wine*, *softdrink*, *fish*, *confectionery*. The front 7 attributes is a person’s information, and the last 11 attributes is a shopping record that describing the goods that the person buys.

Here use the last 11 attributes to generate association rules.

2. supermarket.arff

The dataset has lots of attributes which is same with BASKETS1n in content, but different in format. This data set is used in weka.

3. Learn-Dataset

I1 I2 I5

I1 I2

I2 I4

I1 I2 I4

I1 I3

I1 I2 I3 I5

I1 I2 I3

I2 I5

I2 I3 I4

I3 I4

This dataset is used to learn association rule Apriori algorithm. Its size is very small. This dataset is only used to learn, except for this it has not any values.

# Procedures

1. **Use SPSS Clementine Analyze BASKETS1n**

Step 1: Select nodes and models



Figure 1-1

Select Variable File node, Type node, and Apriori model as shown in Figure 1-1.

Step 2: Linking them

Link the three nodes(Variable File node, Type Filter node and Apriori node) to a flow as shown in Figure 1-2.

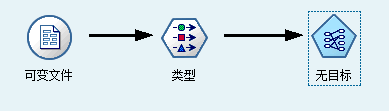


Figure 1-2

Step 3: Set input file path

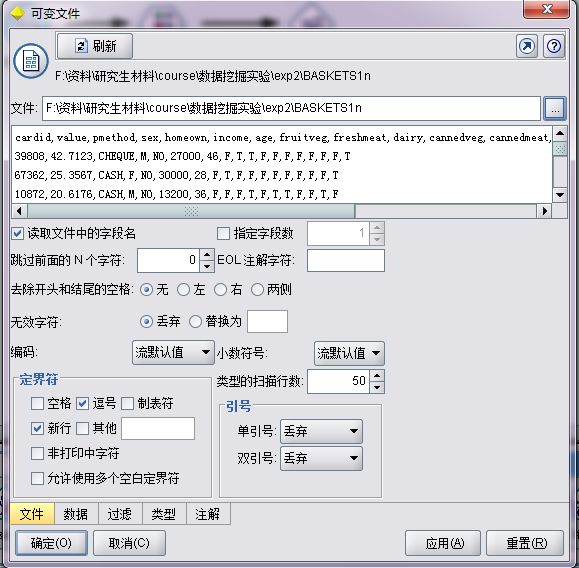


Figure 1-3

Here use dataset BASKETS1n to analyze. Copy the file path in the text field as shown in Figure 1-3 and click OK button loading data.

Step 4: Set Type Filter node

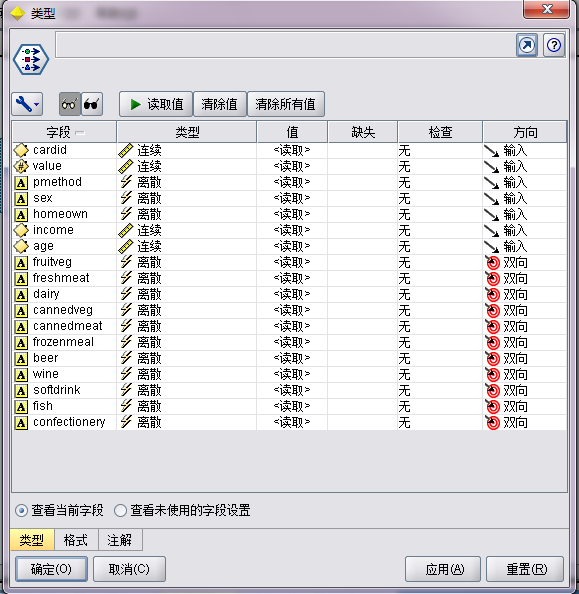


Figure 1-4

What we want is some relations between goods which are always been bought by people in a time. The front 7 attribute is useless here. So we only set the last 11 attributes’ direction as bilateral seeing in Figure 1-4. Click OK button complete filtering.

Step 5: Execute the flow and Produce model

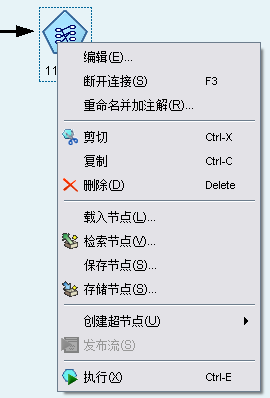


Figure 1-5

Right-click the flow and click the last option named “execute” in the Pop-up menu as shown in Figure 1-5 or use “Ctrl + E”. After the execution the flow will produce a model in the up-right corner of the main frame seeing Figure 1-6. Right-click the model and choose the “scan” option you can get the rules shown in Figure 1-7.



Figure 1-6



Figure 1-7

Step 6: Using the rules to predict

First we should build a flow as shown in Figure 1-8. Then execute the lower flow we can get some information from the system for each record as shown in Figure 1-9. For example, for the 16 record we predict the person will also buy “cannedveg” according rules “bear, frozenmeal => cannedveg” , with confidence = 0.859 as shown in Figure 1-9.

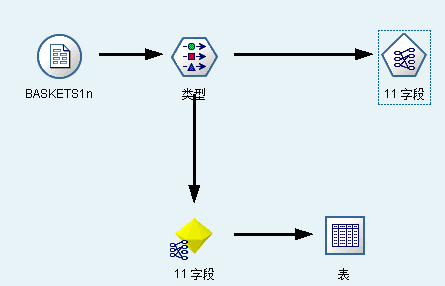


Figure 1-8

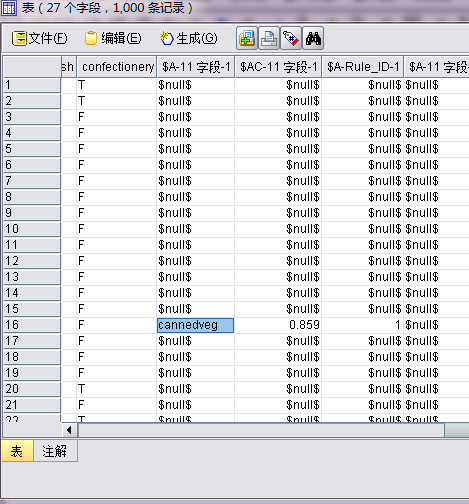


Figure 1-9

1. **Use Weka Analyze supermarket.arff**

Step 1: Choose Application and Load file

Firstly, open Weka, and choose the first application named “Explorer” as shown in Figure 2-1. Then load file “supermarket.arff” seeing in Figure 2-2.

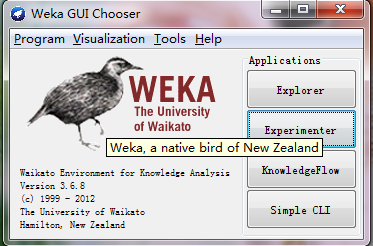


Figure 2-1

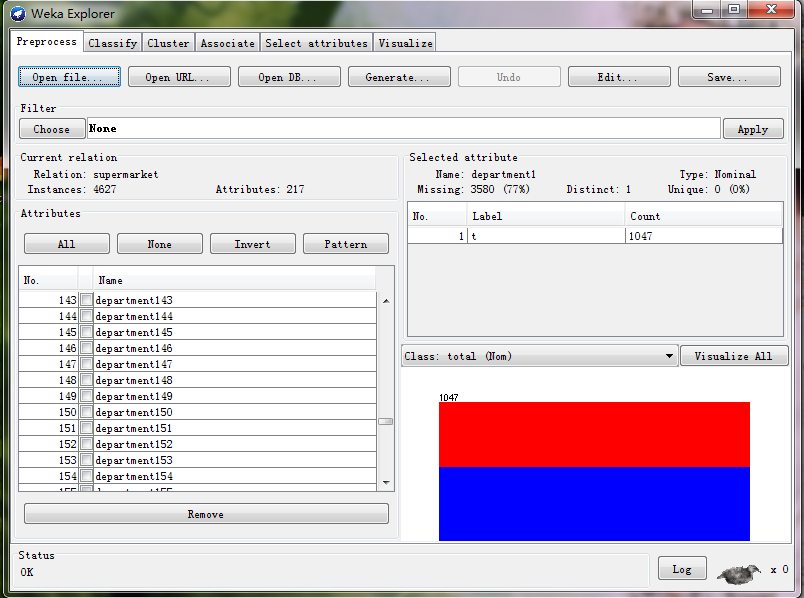


Figure 2-2

Step 2: Select Apriori Algorithm and Set Confidence and MinSub

In the association menu we select Apriori Algorithm seeing Figure 2-3. Then left-click on the Apriori we can set it’s parameter such as Confidence and MinSub as shown in Figure 2-4.

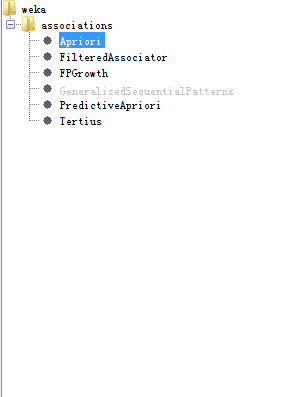


Figure 2-3

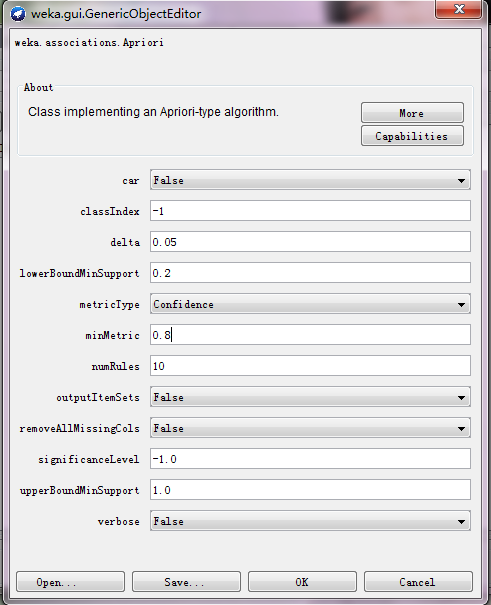


Figure 2-4

Step 3: Gerenate Rules

Execute the Apriori we can get some rules with given Confidence and MinSub as shown in Figure 2-5

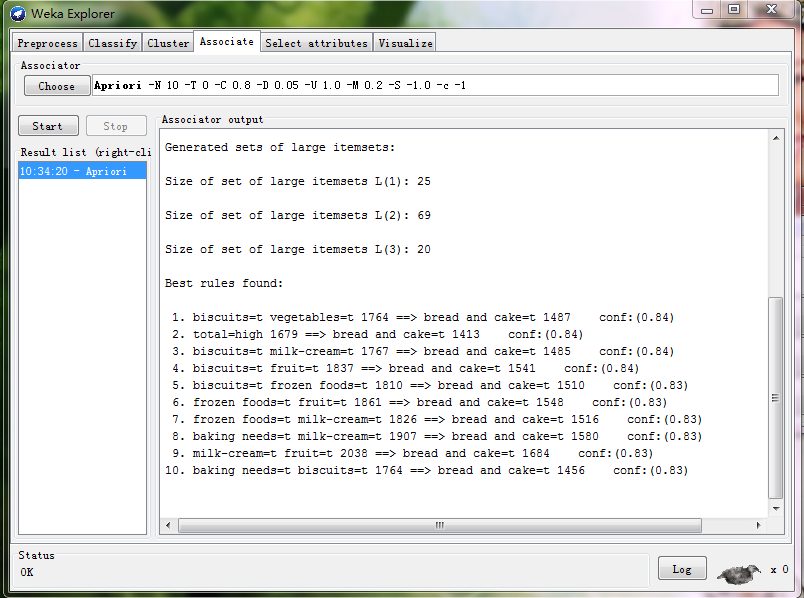


Figure 2-5

1. **Write A Program Based on Apriori**

Step 1: build input file format

InputFile -> Record **\n** InputFile | **empty**

Record -> Good **\t** Record | **empty**

Good->[a-zA-Z1-9]\*

From the regex above we can see, the input file consists of many records using ‘\n’ as interval. A record consists of many goods using ‘\t’ as interval. A good is a string which is the good’s name. Figure 3-1 and 3-2 show two examples.

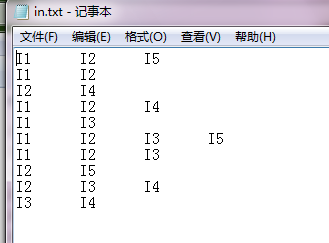


Figure 3-1

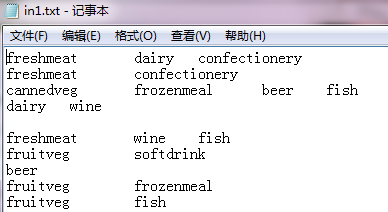


Figure 3-2

Step 2: Design the program

**Record** class : This class can store a record using a vector<int>(Here each good is mapped to a id)

**Set** class : This class defines a set which have some operations such as A – B, A B, A B.

**SetList** class: This class is a list which contains many sets whose size is the same. Here SetList is used to store Ck. The most import method of this class is **SetList \* getNextGenerarion(vector<int> \* f1, vector<Record \*> \* records, int sub)**. This method is used to generate Ck+1 from Ck.

**Rules** class: This class store the rules such as A 🡺 B which sub and confidence.

**InputNode** class: This class is used to read input file from hard disk. The class using map<string, int> to change a string to a int number.

**Apriori** class: This class is the most important class. In this class the most important method is void **work()**, the pseudo code is:

find frequent\_1 itemsets

Ck <- frequent-1 itemsets

While(Ck != NULL){

Using Ck produce Ck+1

Ck <- Ck+1

}

Using frequent itemsets to produce rules

Step 3: Execute the Program

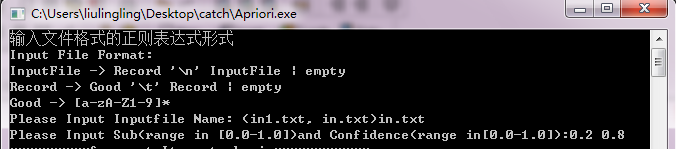


Figure 3-3

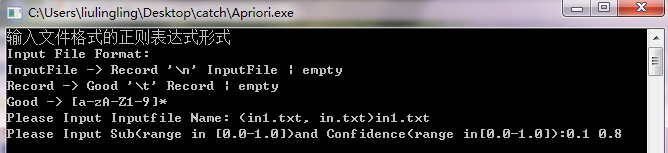


Figure 3-4

Execute the program and input file name and set MinSub and Confidence As shown in Figure 3-3 and Figure 3-4. Then we can get some rules. For file in.txt we can get two rules: (I1) 🡺 (I2) with confidence = 0.833, (I1 I5) 🡺 (I2) with confidence = 1.000 as shown in Figure 3-5. For file in1.txt we can three rules: (cannedveg frozenmeal) 🡺 (beer) with conficence = 0.843, (cannedveg beer) 🡺 (frozenmeal) with confidence = 0.874, (frozenmeal beer) 🡺 (cannedveg) with confidence = 0.859 as shown in Figure 3-6 which is the same with the answer the clementine. From the two figures we can see the time the program cost which contains Load File time, Apriori time, and Total time. The frequent itemsets is also printed with its appeared times. These information is stored in file “out.txt” and “out1.txt”.

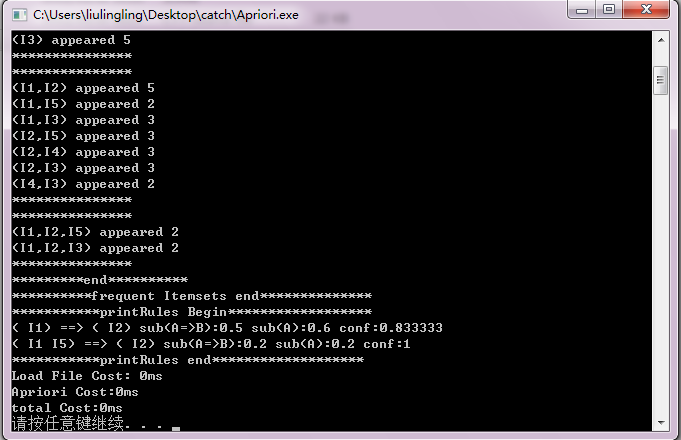


Figure 3-5

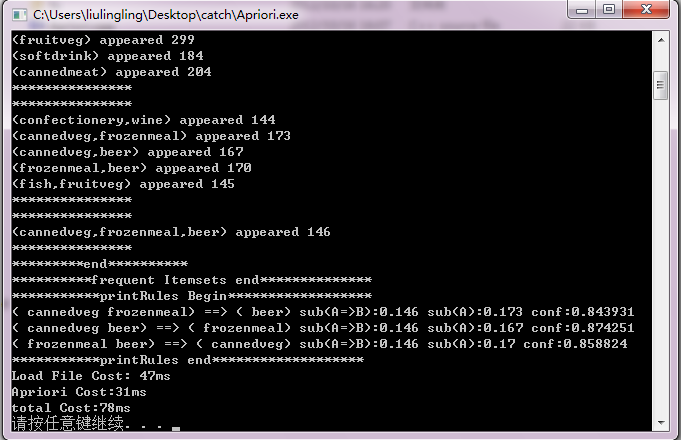


Figure 3-6

# Conclusion and Analysis

From the result of “in.txt” we can get five 1-frequent-itemsets, seven 2-frequent-itemsets, two 3-frequent-itemsets and two rules (I1) 🡺 (I2) with confidence = 0.833, (I1 I5) 🡺 (I2) with confidence = 1 as shown in Figure 4-1. So we can say if someone buy good I1 and I5, he is more likely to buy I2, according to this we can put these three things together.

From the result of “in1.txt” we can get eleven 1-frequent-itemsets, five 2-frequent-itemsets, one 3-frequent-itemsets and three rules (cannedveg frozenmeal) 🡺 (beer) with confidence = 0.843, (cannedveg beer) 🡺 (frozenmeal) with confidence = 0.874, (frozenmeal beer) 🡺 (cannedveg) with confidence = 0.859 as shown in Figure 4-2 which is consistent with the result using Clementine. So we can say if someone buy two of these three things he is more likely to buy the last thing. According to this we can put frozenmeal, beer and cannedveg together.

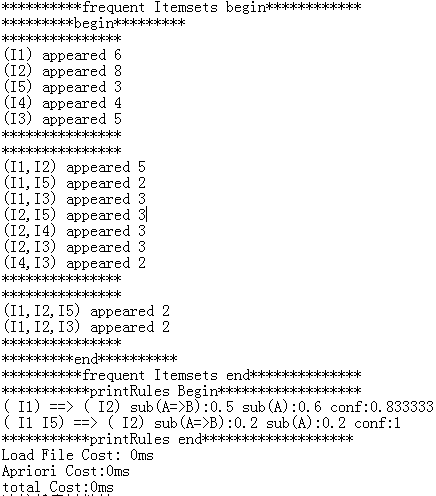


Figure 4-1

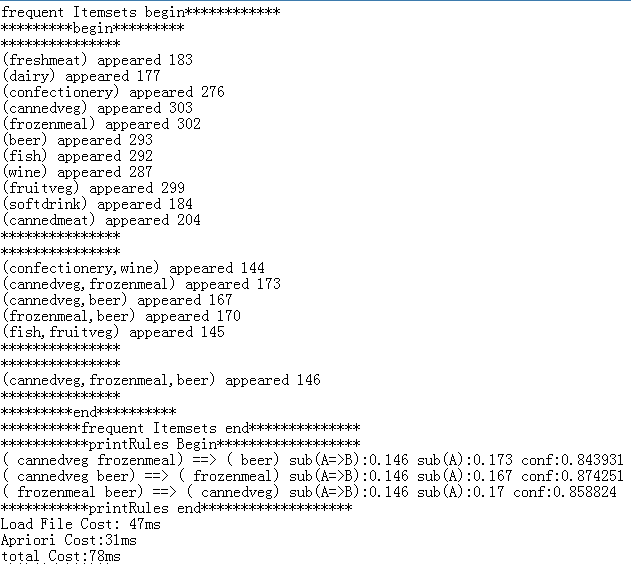


Figure 4-2