EXPERIMENT NO 6

NAME: Vishal Shashikant Salvi **UID: 2019230069**

Class: TE Comps **BATCH:C**

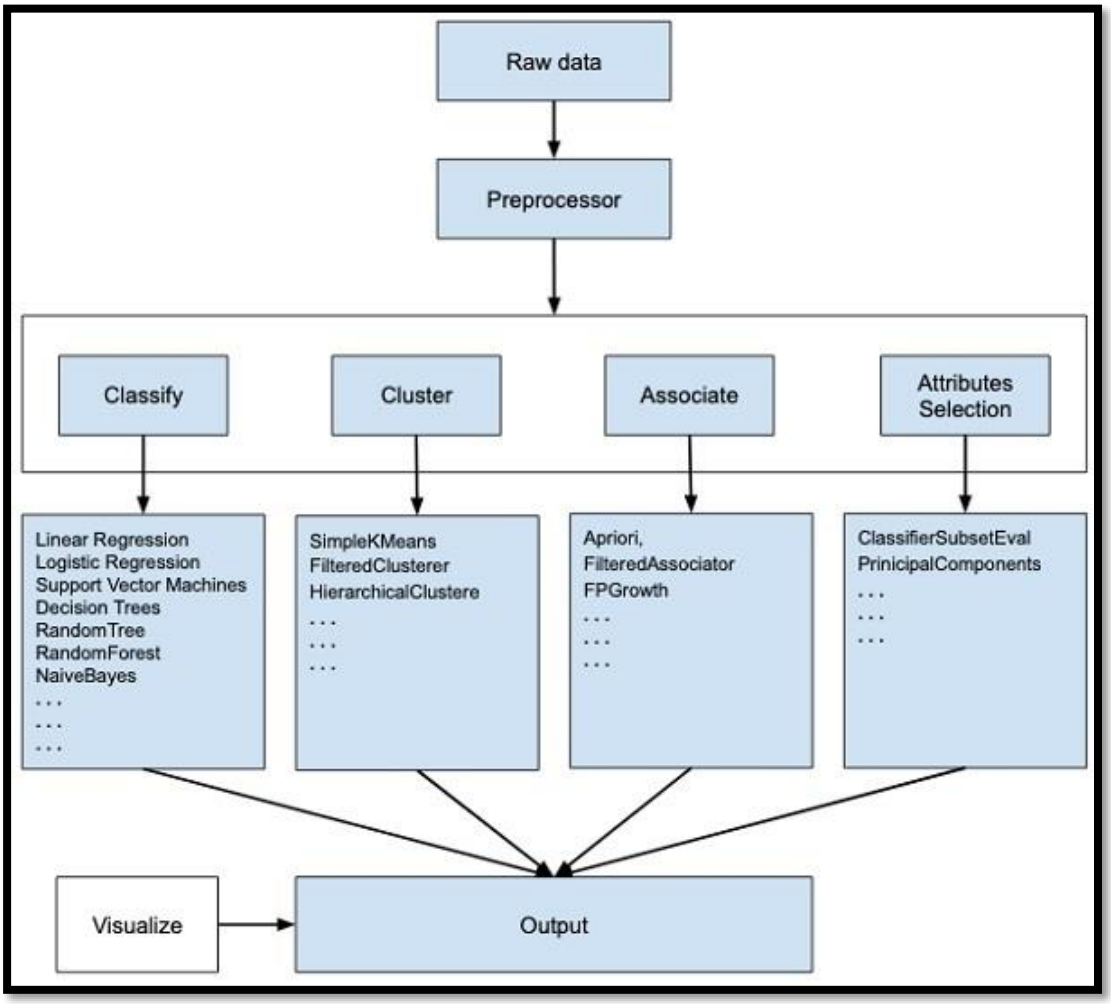
Aim: To implement Association Rule for Apriori Algorithm by using Weka tool.

Theory:

Weka

Weka is a comprehensive software that lets you to preprocess the big data, apply different machine learning algorithms on big data and compare various outputs. This software makes it easy to work with big data and train a machine using machine learning algorithms. This tutorial will guide you in the use of WEKA for achieving all the above requirements.

WEKA - an open source software provides tools for data preprocessing, implementation of several Machine Learning algorithms, and visualization tools so that you can develop machine learning techniques and apply them to real-world data mining problems. What WEKA offers is summarized in the following diagram −



If you observe the beginning of the flow of the image, you will understand that there are many stages in dealing with Big Data to make it suitable for machine learning .

First, you will start with the raw data collected from the field. This data may contain several null values and irrelevant fields. You use the data preprocessing tools provided in WEKA to cleanse the data.

Then, you would save the preprocessed data in your local storage for applying ML algorithms.

Next, depending on the kind of ML model that you are trying to develop you would select one of the options such as **Classify, Cluster**, or **Associate**. The **Attributes Selection** allows the automatic selection of features to create a reduced dataset.

Note that under each category, WEKA provides the implementation of several algorithms. You would select an algorithm of your choice, set the desired parameters and run it on the dataset.

Then, WEKA would give you the statistical output of the model processing. It provides you a visualization tool to inspect the data.

The various models can be applied on the same dataset. You can then compare the outputs of different models and select the best that meets your purpose.

Thus, the use of WEKA results in a quicker development of machine learning models on the whole.

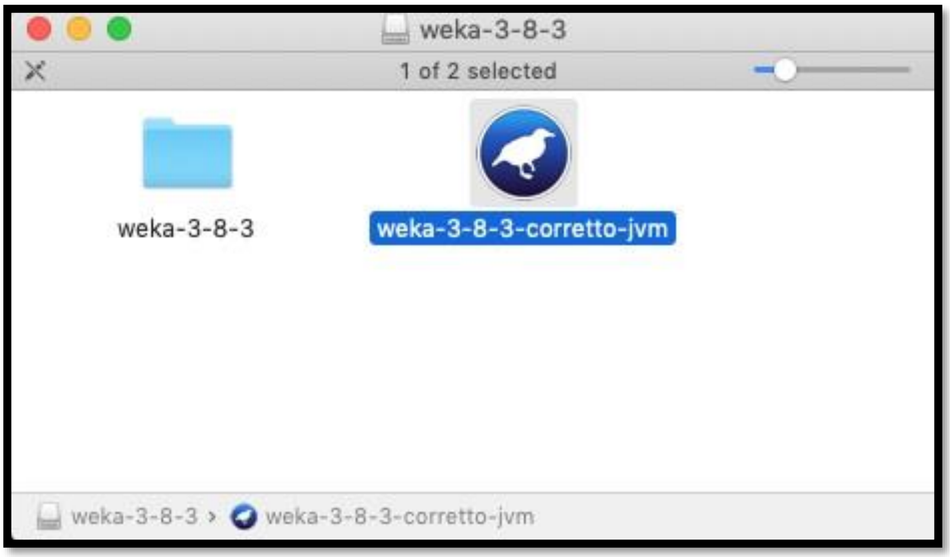
Now that we have seen what WEKA is and what it does, in the next chapter let us learn how to install WEKA on your local computer.

To install WEKA on your machine, visit WEKA's official website and download the installation file. WEKA supports installation on Windows, Mac OS X and Linux. You just need to follow the instructions on this page to install WEKA for your OS.

The steps for installing on Mac are as follows –

- Download the Mac installation file.
- Double click on the downloaded **weka-3-8-3-corretto-jvm.dmg file**.

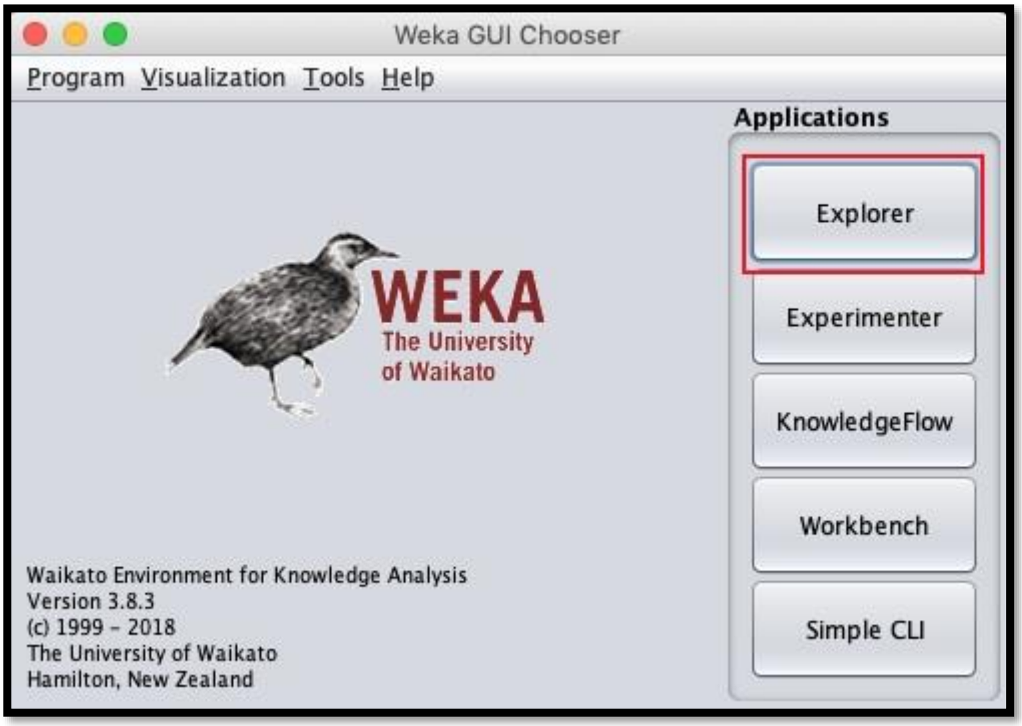
You will see the following screen on successful installation.



- Click on the **weak-3-8-3-corretto-jvm** icon to start Weka.
- Optionally you may start it from the command line –

```
java -jar weka.jar
```

The WEKA GUI Chooser application will start and you would see the following screen –

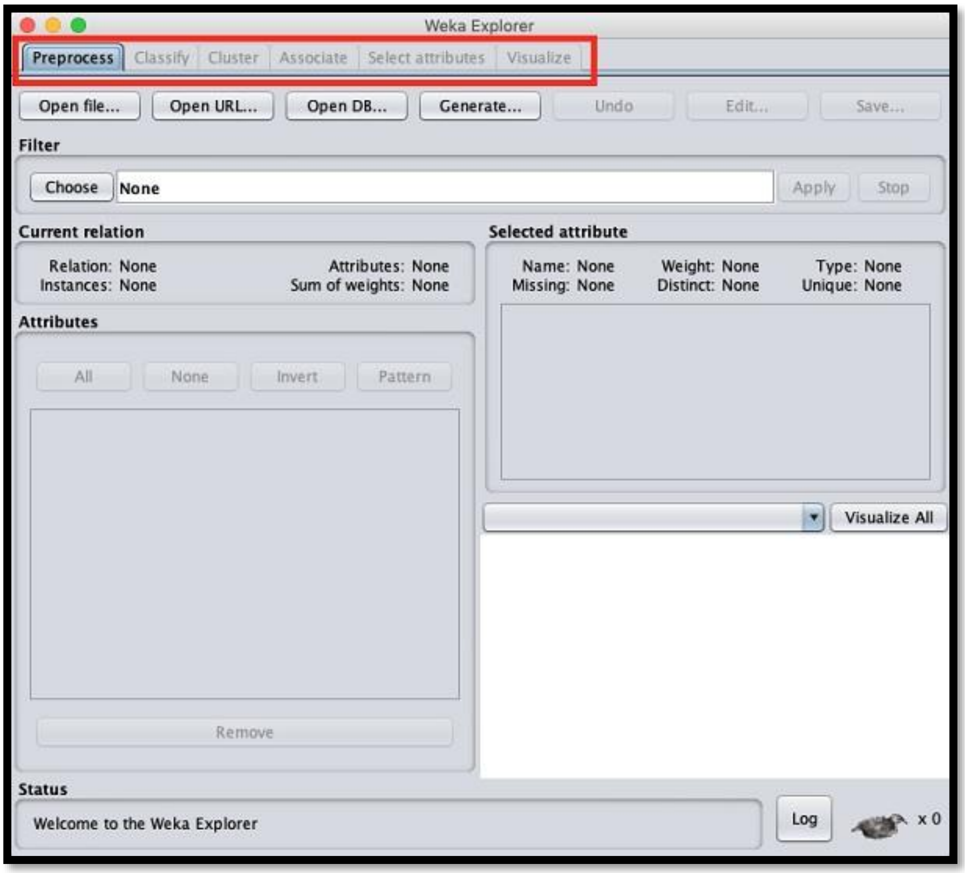


Waikato Environment for Knowledge Analysis
Version 3.8.3
(c) 1999 - 2018
The University of Waikato
Hamilton, New Zealand

The GUI Chooser application allows you to run five different types of applications as listed here –

- Explorer
- Experimenter
- KnowledgeFlow
- Workbench
- Simple CLI

When you click on the **Explorer** button in the **Applications** selector, it opens the following screen –



Filter: None

Relation: None
Instances: None
Attributes: None
Sum of weights: None

Name: None
Missing: None
Weight: None
Distinct: None
Type: None
Unique: None

Welcome to the Weka Explorer

On the top, you will see several tabs as listed here –

- Preprocess
- Classify
- Cluster
- Associate
- Select Attributes
- Visualize

Under these tabs, there are several pre-implemented machine learning algorithms. Let us look into each of them in detail now.

Preprocess Tab

Initially as you open the explorer, only the **Preprocess** tab is enabled. The first step in machine learning is to preprocess the data. Thus, in the **Preprocess** option, you will select the data file, process it and make it fit for applying the various machine learning algorithms.

Classify Tab

The **Classify** tab provides you several machine learning algorithms for the classification of your data. To list a few, you may apply algorithms such as Linear Regression, Logistic Regression, Support Vector Machines, Decision Trees, RandomTree, RandomForest, NaiveBayes, and so on. The list is very exhaustive and provides both supervised and unsupervised machine learning algorithms.

Cluster Tab

Under the **Cluster** tab, there are several clustering algorithms provided such as SimpleKMeans, FilteredClusterer, HierarchicalClusterer, and so on.

Associate Tab

Under the **Associate** tab, you would find Apriori, FilteredAssociator and FPGrowth.

Select Attributes Tab

Select Attributes allows you feature selections based on several algorithms such as ClassifierSubsetEval, PrinicipalComponents, etc.

Visualize Tab

Lastly, the **Visualize** option allows you to visualize your processed data for analysis.

As you noticed, WEKA provides several ready-to-use algorithms for testing and building your machine learning applications. To use WEKA effectively, you must have a sound knowledge of these algorithms, how they work, which one to choose under what circumstances, what to look for in their processed output, and so on. In short, you must have a solid foundation in machine learning to use WEKA effectively in building your apps.

WEKA supports a large number of file formats for the data. Here is the complete list −

- arff
- arff.gz
- bsi
- csv
- dat
- data
- json
- json.gz
- libsvm
- m
- names
- xrff
- xrff.gz

Weka - Association

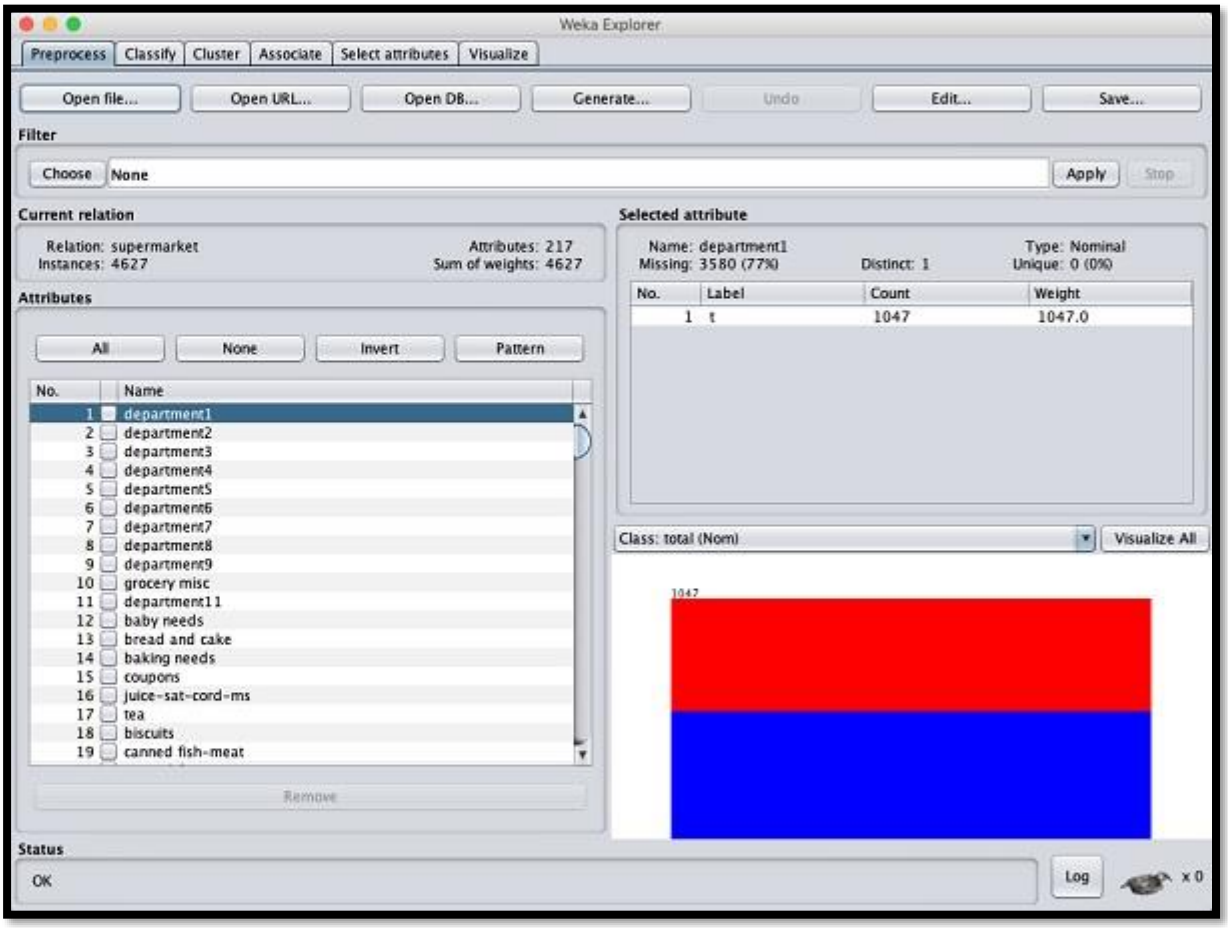
It was observed that people who buy beer also buy diapers at the same time. That is there is an association in buying beer and diapers together. Though this seems not well convincing, this association rule was mined from huge databases of supermarkets. Similarly, an association may be found between peanut butter and bread.

Finding such associations becomes vital for supermarkets as they would stock diapers next to beers so that customers can locate both items easily resulting in an increased sale for the supermarket.

The **Apriori** algorithm is one such algorithm in ML that finds out the probable associations and creates association rules. WEKA provides the implementation of the Apriori algorithm. You can define the minimum support and an acceptable confidence level while computing these rules. You will apply the **Apriori** algorithm to the **supermarket** data provided in the WEKA installation.

Loading Data

In the WEKA explorer, open the **Preprocess** tab, click on the **Open file** ... button and select **supermarket.arff** database from the installation folder. After the data is loaded you will see the following screen −



Filter: None

Relation: supermarket
Instances: 4627
Attributes: 217
Sum of weights: 4627

No.	Name
1	department1
2	department2
3	department3
4	department4
5	department5
6	department6
7	department7
8	department8
9	department9
10	grocery misc
11	department11
12	baby needs
13	bread and cake
14	baking needs
15	coupons
16	juice-sat-cord-ms
17	tea
18	biscuits
19	canned fish-meat

Name: department1
Missing: 3580 (77%)
Distinct: 1
Type: Nominal
Unique: 0 (0%)

No.	Label	Count	Weight
1	t	1047	1047.0

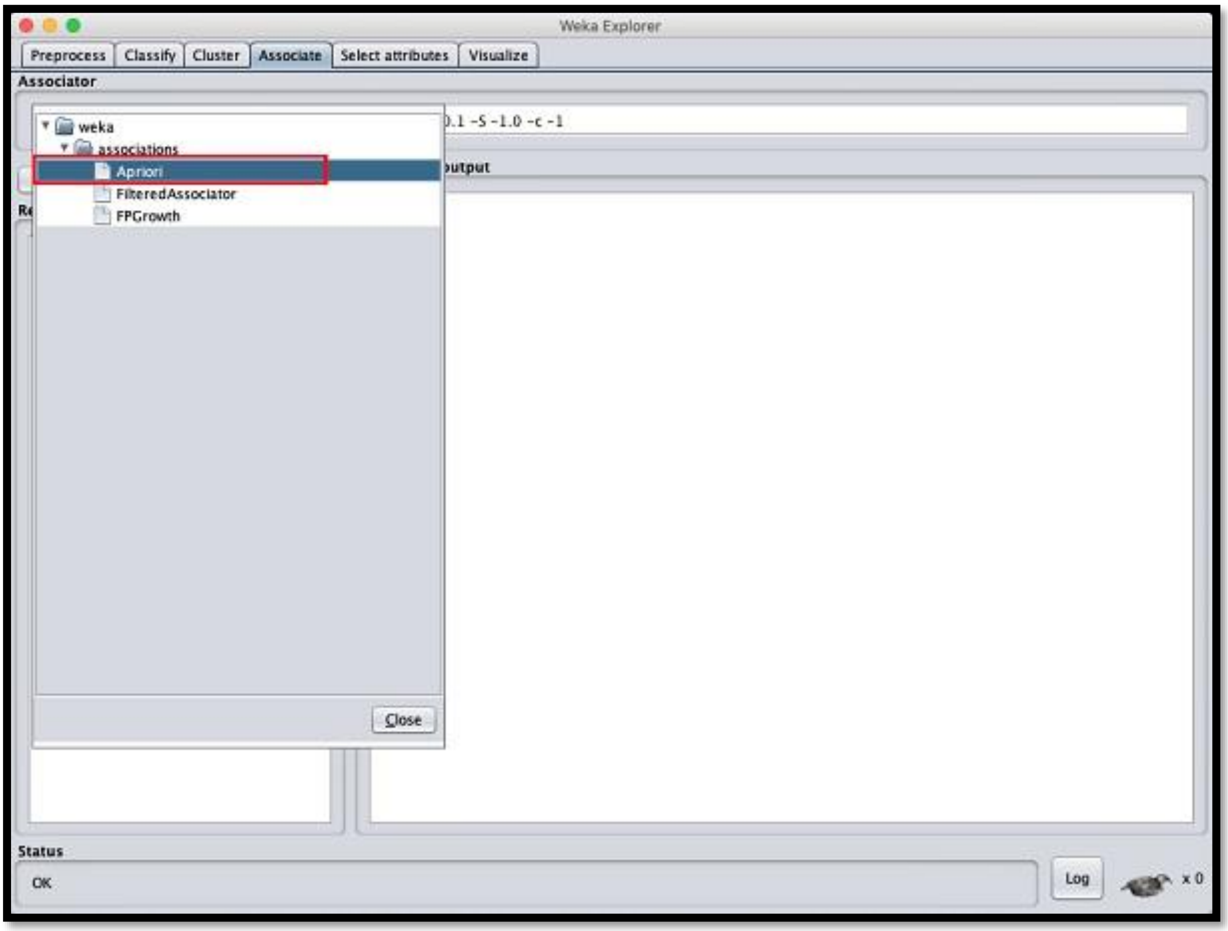
Class: total (Nom)

OK

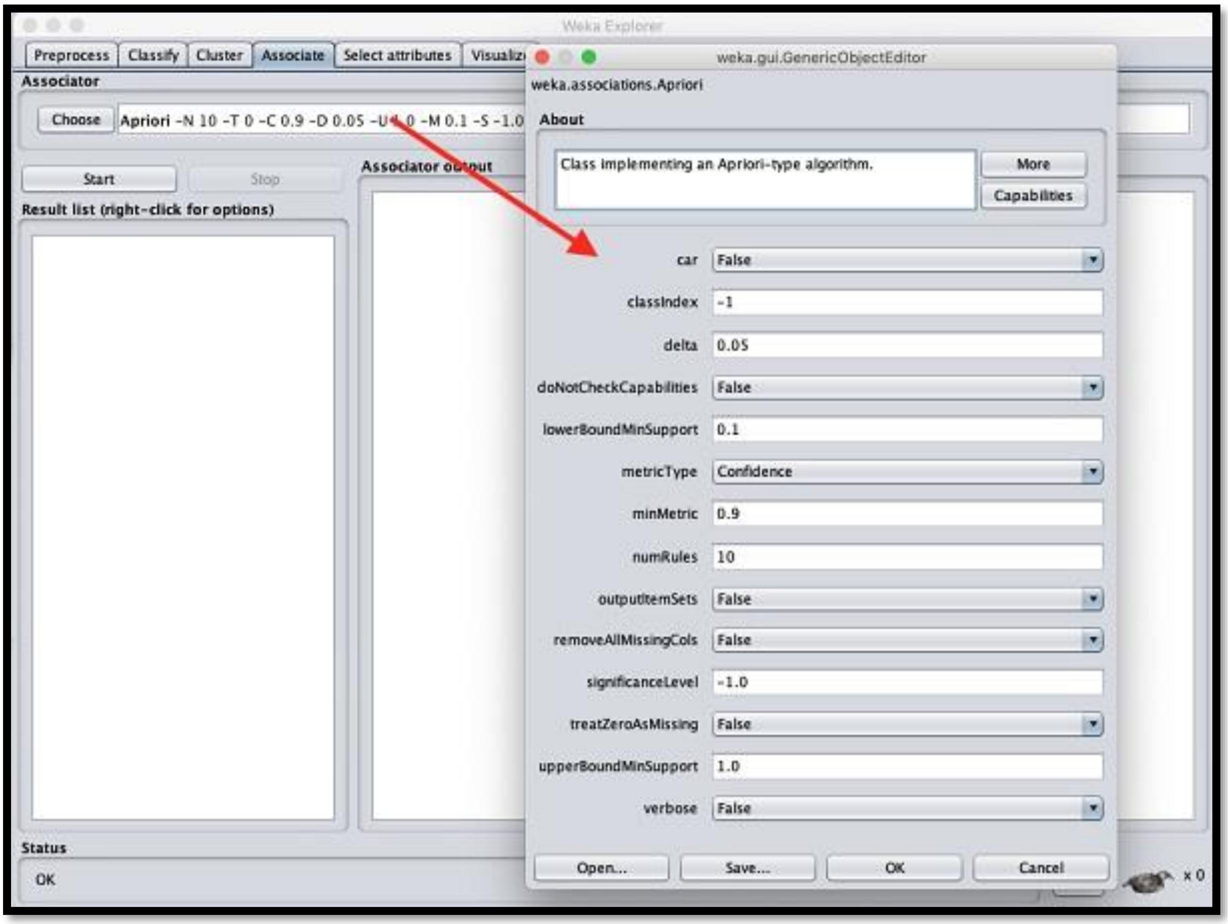
The database contains 4627 instances and 217 attributes. You can easily understand how difficult it would be to detect the association between such a large number of attributes. Fortunately, this task is automated with the help of Apriori algorithm.

Associator

Click on the **Associate** TAB and click on the **Choose** button. Select the **Apriori** association as shown in the screenshot –



To set the parameters for the Apriori algorithm, click on its name, a window will pop up as shown below that allows you to set the parameters –



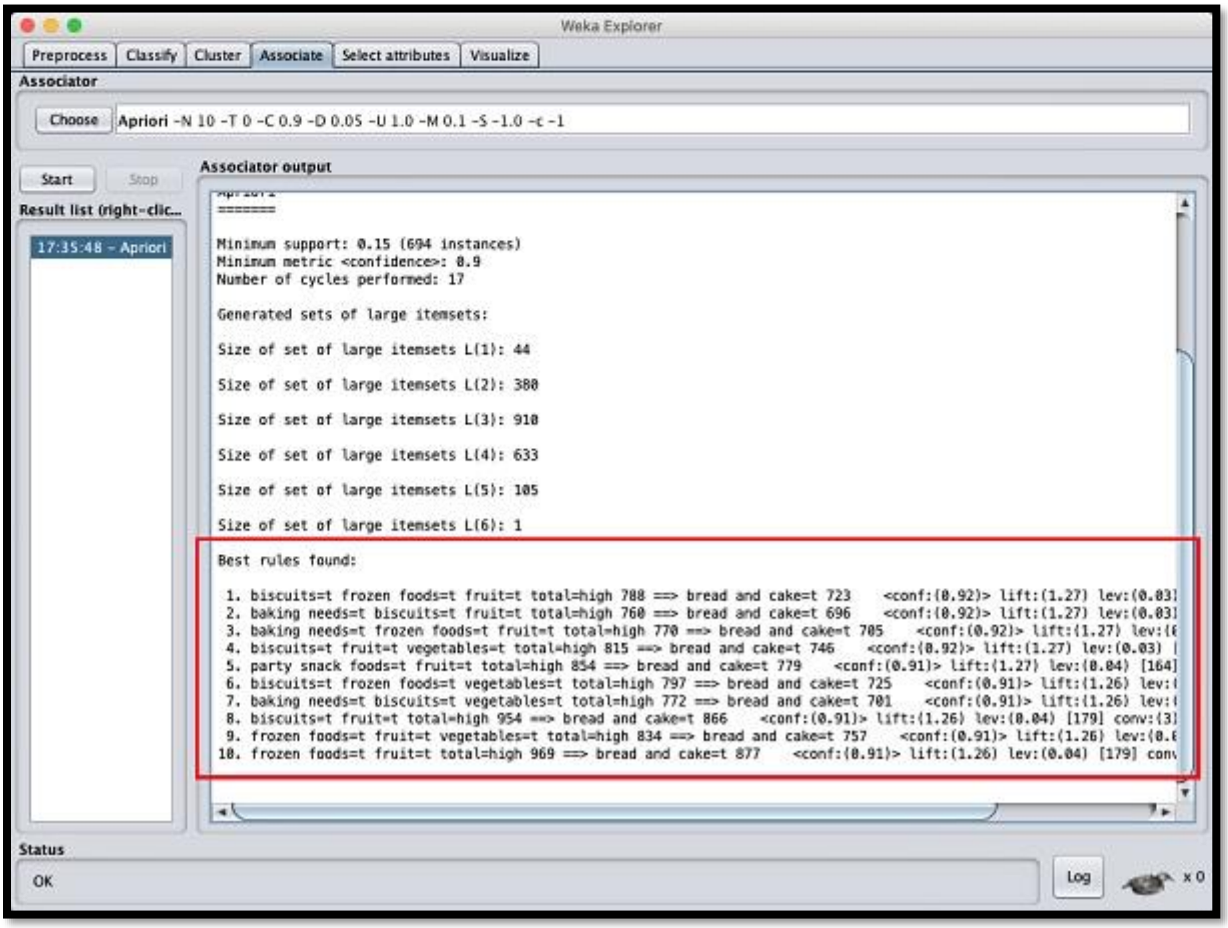
Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0

weka.associations.Apriori

Class implementing an Apriori-type algorithm.

car: False
classIndex: -1
delta: 0.05
doNotCheckCapabilities: False
lowerBoundMinSupport: 0.1
metricType: Confidence
minMetric: 0.9
numRules: 10
outputItemSets: False
removeAllMissingCols: False
significanceLevel: -1.0
treatZeroAsMissing: False
upperBoundMinSupport: 1.0
verbose: False

After you set the parameters, click the **Start** button. After a while you will see the results as shown in the screenshot below −



Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

```
=======

Minimum support: 0.15 (694 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

Size of set of large itemsets L(1): 44

Size of set of large itemsets L(2): 380

Size of set of large itemsets L(3): 910

Size of set of large itemsets L(4): 633

Size of set of large itemsets L(5): 105

Size of set of large itemsets L(6): 1

Best rules found:

 1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723    <conf:(0.92)> lift:(1.27) lev:(0.03]
 2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and cake=t 696    <conf:(0.92)> lift:(1.27) lev:(0.03]
 3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705    <conf:(0.92)> lift:(1.27) lev:(
 4. biscuits=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746    <conf:(0.92)> lift:(1.27) lev:(0.03) |
 5. party snack foods=t fruit=t total=high 854 ==> bread and cake=t 779    <conf:(0.91)> lift:(1.27) lev:(0.04) [164]
 6. biscuits=t frozen foods=t vegetables=t total=high 797 ==> bread and cake=t 725    <conf:(0.91)> lift:(1.26) lev:(
 7. baking needs=t biscuits=t vegetables=t total=high 772 ==> bread and cake=t 701    <conf:(0.91)> lift:(1.26) lev:(
 8. biscuits=t fruit=t total=high 954 ==> bread and cake=t 866    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(3]
 9. frozen foods=t fruit=t vegetables=t total=high 834 ==> bread and cake=t 757    <conf:(0.91)> lift:(1.26) lev:(0.6
10. frozen foods=t fruit=t total=high 969 ==> bread and cake=t 877    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv
```

At the bottom, you will find the detected best rules of associations. This will help the supermarket in stocking their products in appropriate shelves.

Apriori Algorithm:

Apriori algorithm was the first algorithm that was proposed for frequent itemset mining. It was later improved by R Agarwal and R Srikant and came to be known as Apriori. This algorithm uses two steps "join" and "prune" to reduce the search space. It is an iterative approach to discover the most frequent itemsets.

Apriori says:

The probability that item I is not frequent is if:

- P(I) < minimum support threshold, then I is not frequent.
- P (I+A) < minimum support threshold, then I+A is not frequent, where A also belongs to itemset.
- If an itemset set has value less than minimum support then all of its supersets will also fall below min support, and thus can be ignored. This property is called the Antimonotone property.

Applications of Apriori Algorithm

Some fields where Apriori is used:

1. **In Education Field:** Extracting association rules in data mining of admitted students through characteristics and specialties.
2. **In the Medical field:** For example Analysis of the patient's database.
3. **In Forestry:** Analysis of probability and intensity of forest fire with the forest fire data.
4. Apriori is used by many companies like Amazon in the **Recommender System** and by Google for the auto-complete feature.

Advantages

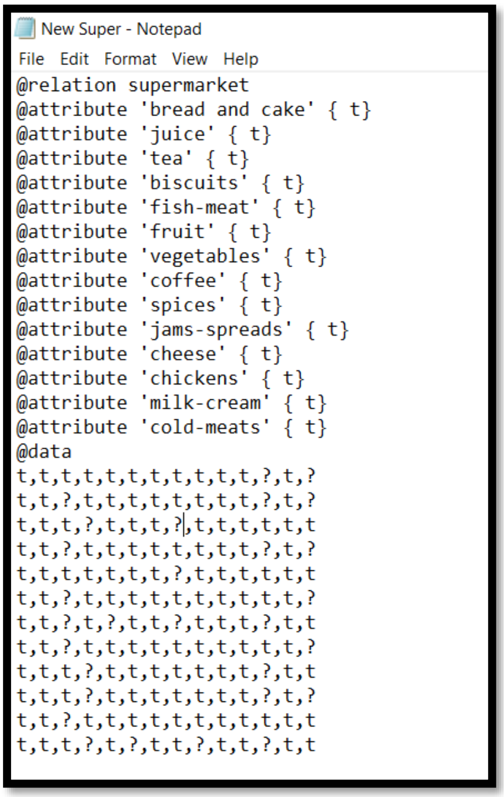
1. Easy to understand algorithm
2. Join and Prune steps are easy to implement on large itemsets in large databases

Disadvantages

1. It requires high computation if the itemsets are very large and the minimum support is kept very low.
2. The entire database needs to be scanned.

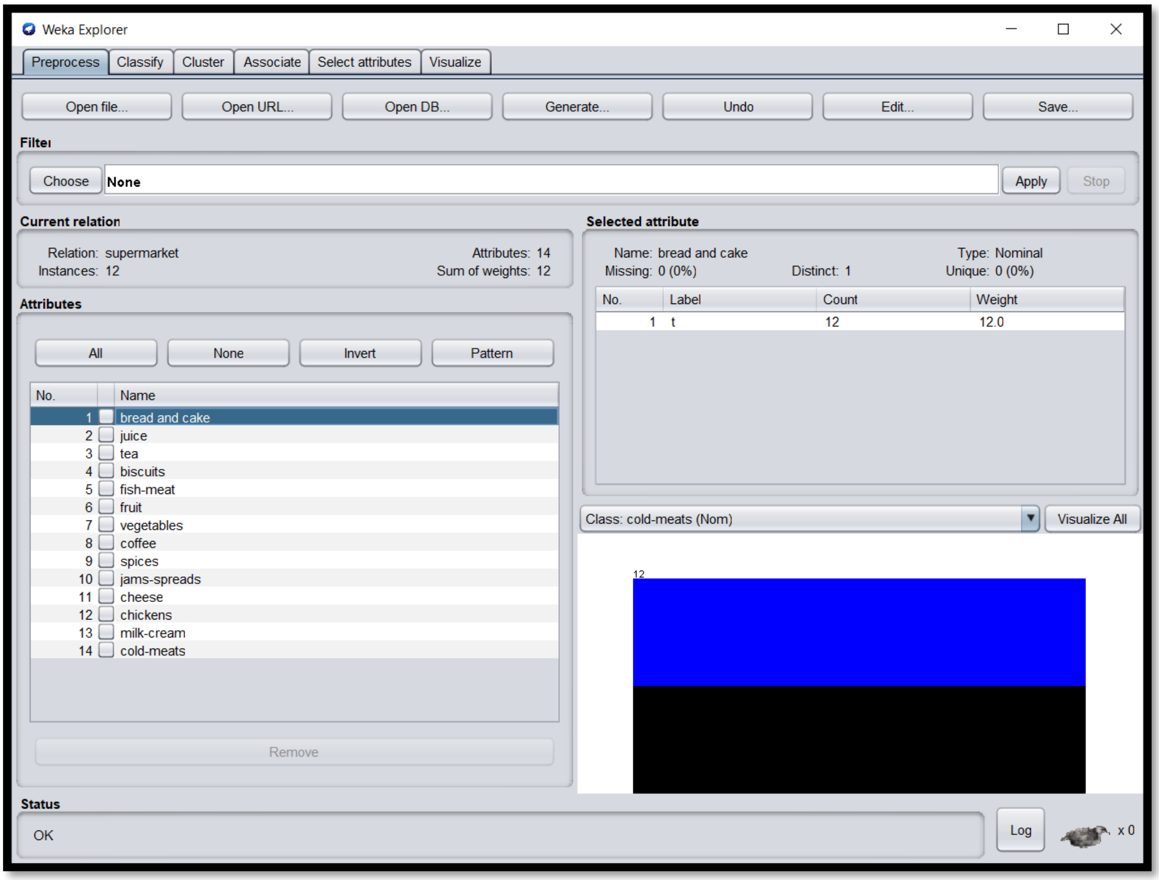
Screenshot:

New Super.arff



New Super - Notepad

```
@relation supermarket
@attribute 'bread and cake' { t}
@attribute 'juice' { t}
@attribute 'tea' { t}
@attribute 'biscuits' { t}
@attribute 'fish-meat' { t}
@attribute 'fruit' { t}
@attribute 'vegetables' { t}
@attribute 'coffee' { t}
@attribute 'spices' { t}
@attribute 'jams-spreads' { t}
@attribute 'cheese' { t}
@attribute 'chickens' { t}
@attribute 'milk-cream' { t}
@attribute 'cold-meats' { t}
@data
t,t,t,t,t,t,t,t,t,t,t,?,t,?
t,t,?,t,t,t,t,t,t,t,t,?,t,?
t,t,t,?,t,t,t,?,t,t,t,t,t,t
t,t,?,t,t,t,t,t,t,t,t,?,t,?
t,t,t,t,t,t,t,?,t,t,t,t,t,t
t,t,?,t,t,t,t,t,t,t,t,t,t,?
t,t,?,t,?,t,t,?,t,t,t,?,t,t
t,t,?,t,t,t,t,t,t,t,t,t,t,?
t,t,t,?,t,t,t,t,t,t,t,?,t,t
t,t,t,?,t,t,t,t,t,t,t,?,t,?
t,t,?,t,t,t,t,t,t,t,t,t,t,t
t,t,t,?,t,?,t,t,?,t,t,?,t,t
```



Weka Explorer

Filter

None

Current relation

Relation: supermarket Attributes: 14
Instances: 12 Sum of weights: 12

Attributes

No.		Name
1	□	bread and cake
2	□	juice
3	□	tea
4	□	biscuits
5	□	fish-meat
6	□	fruit
7	□	vegetables
8	□	coffee
9	□	spices
10	□	jams-spreads
11	□	cheese
12	□	chickens
13	□	milk-cream
14	□	cold-meats

Selected attribute

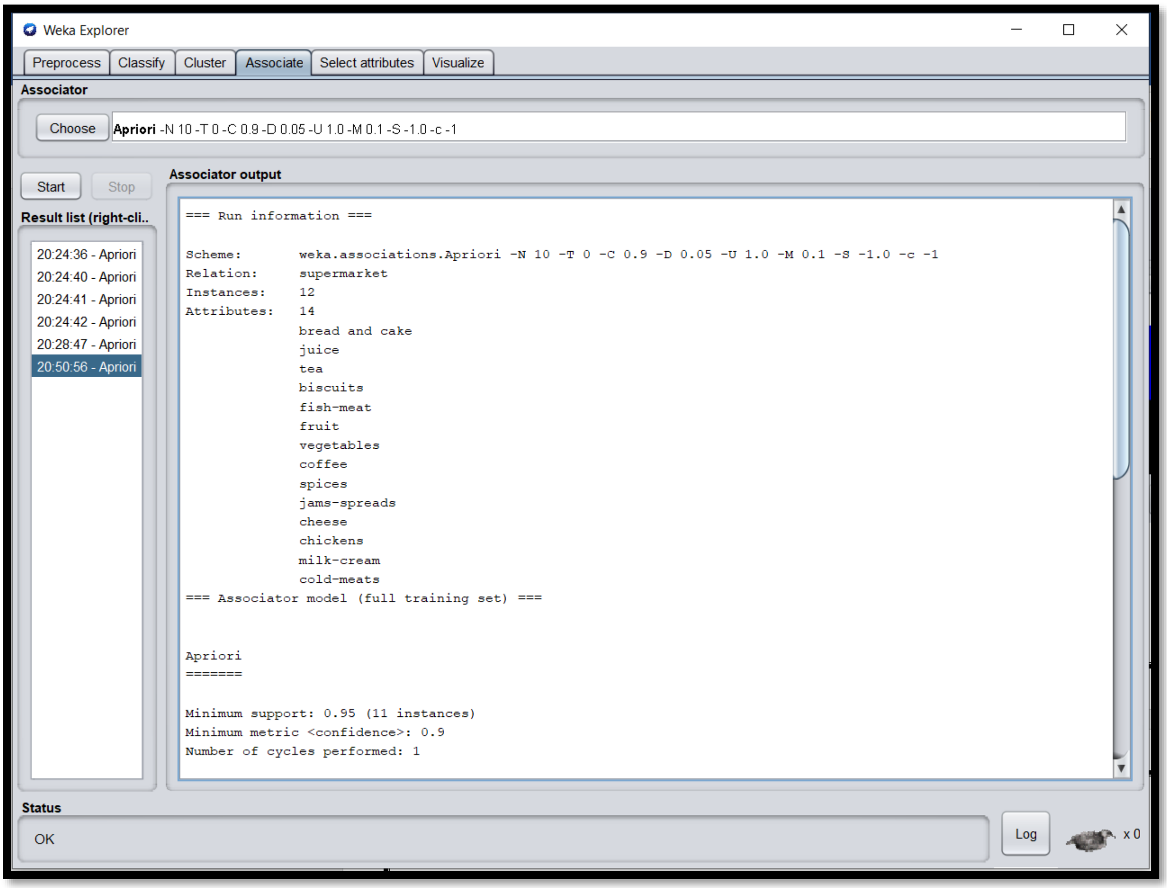
Name: bread and cake Type: Nominal
Missing: 0 (0%) Distinct: 1 Unique: 0 (0%)

No.	Label	Count	Weight
1	t	12	12.0

Class: cold-meats (Nom)

Status

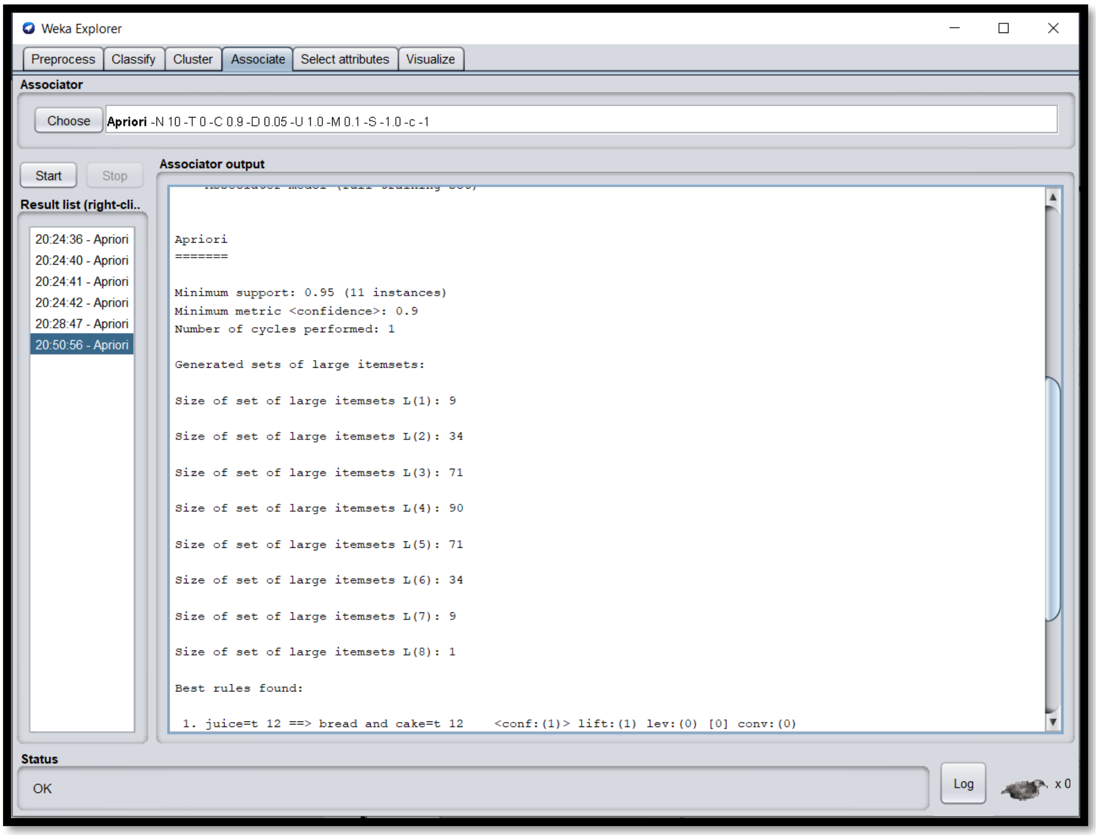
OK



Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

Associator output

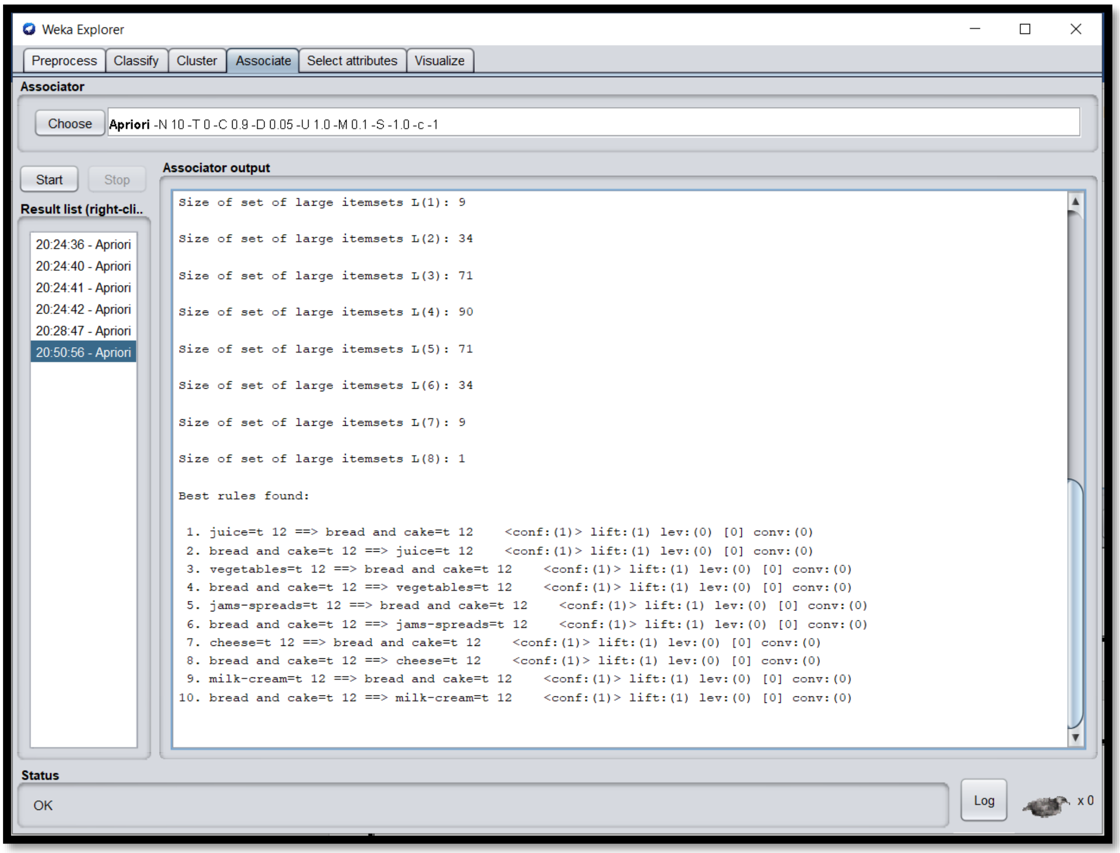
```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:     supermarket
Instances:    12
Attributes:   14
              bread and cake
              juice
              tea
              biscuits
              fish-meat
              fruit
              vegetables
              coffee
              spices
              jams-spreads
              cheese
              chickens
              milk-cream
              cold-meats
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.95 (11 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 1
```



Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

Associator output

```


Apriori
=======

Minimum support: 0.95 (11 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 1

Generated sets of large itemsets:

Size of set of large itemsets L(1): 9

Size of set of large itemsets L(2): 34

Size of set of large itemsets L(3): 71

Size of set of large itemsets L(4): 90

Size of set of large itemsets L(5): 71

Size of set of large itemsets L(6): 34

Size of set of large itemsets L(7): 9

Size of set of large itemsets L(8): 1

Best rules found:

 1. juice=t 12 ==> bread and cake=t 12    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
```



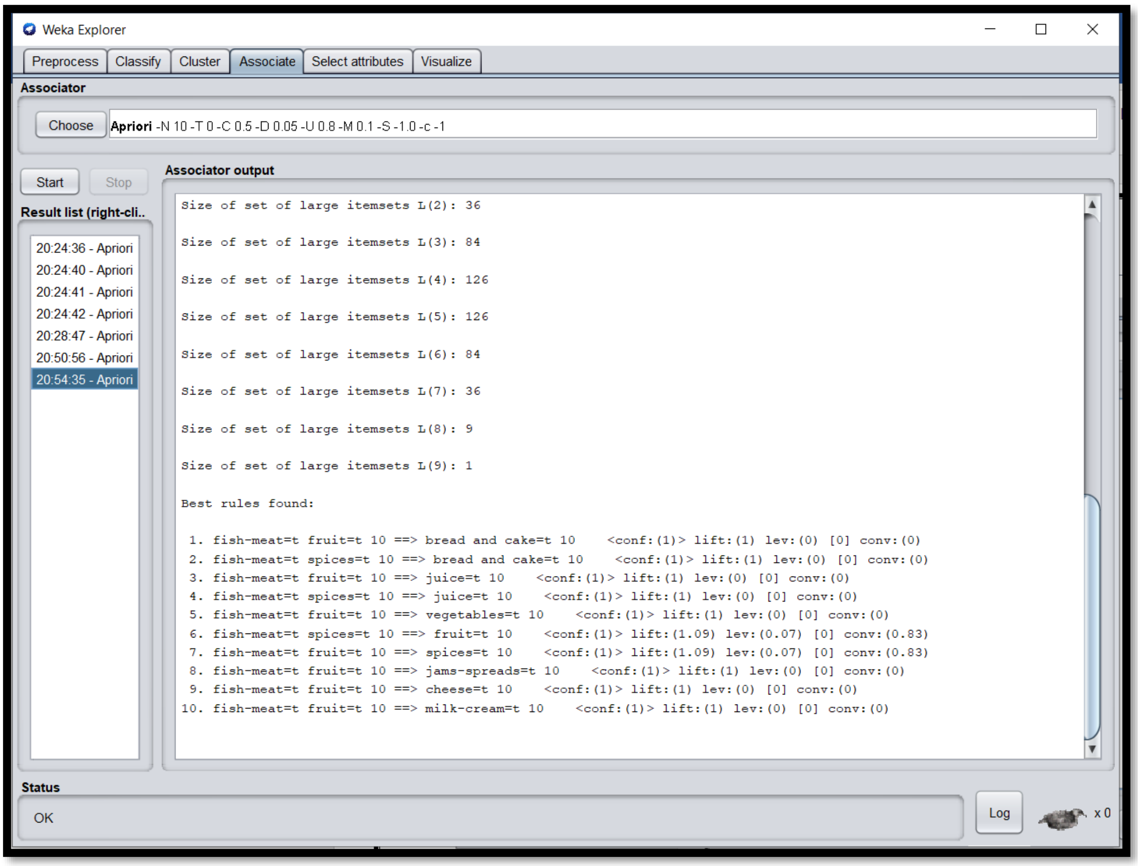
Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

20:24:36 - Apriori
20:24:40 - Apriori
20:24:41 - Apriori
20:24:42 - Apriori
20:28:47 - Apriori
20:50:56 - Apriori

```
Size of set of large itemsets L(1): 9

Size of set of large itemsets L(2): 34

Size of set of large itemsets L(3): 71

Size of set of large itemsets L(4): 90

Size of set of large itemsets L(5): 71

Size of set of large itemsets L(6): 34

Size of set of large itemsets L(7): 9

Size of set of large itemsets L(8): 1

Best rules found:

 1. juice=t 12 ==> bread and cake=t 12    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 2. bread and cake=t 12 ==> juice=t 12    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 3. vegetables=t 12 ==> bread and cake=t 12    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 4. bread and cake=t 12 ==> vegetables=t 12    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 5. jams-spreads=t 12 ==> bread and cake=t 12    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 6. bread and cake=t 12 ==> jams-spreads=t 12    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 7. cheese=t 12 ==> bread and cake=t 12    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 8. bread and cake=t 12 ==> cheese=t 12    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 9. milk-cream=t 12 ==> bread and cake=t 12    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
10. bread and cake=t 12 ==> milk-cream=t 12    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
```

OK

Confident = 0.5 Support = 0.8

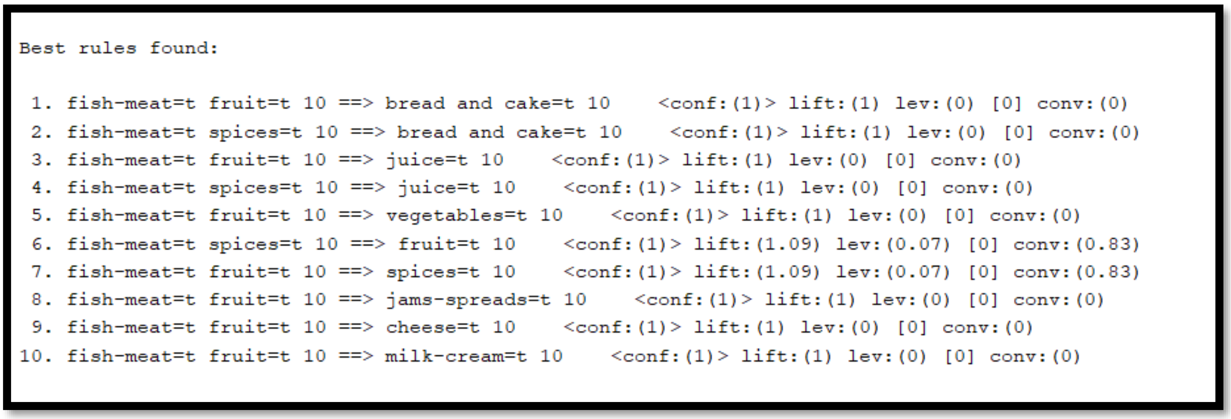


Apriori -N 10 -T 0 -C 0.5 -D 0.05 -U 0.8 -M 0.1 -S -1.0 -c -1

20:24:36 - Apriori
20:24:40 - Apriori
20:24:41 - Apriori
20:24:42 - Apriori
20:28:47 - Apriori
20:50:56 - Apriori
20:54:35 - Apriori

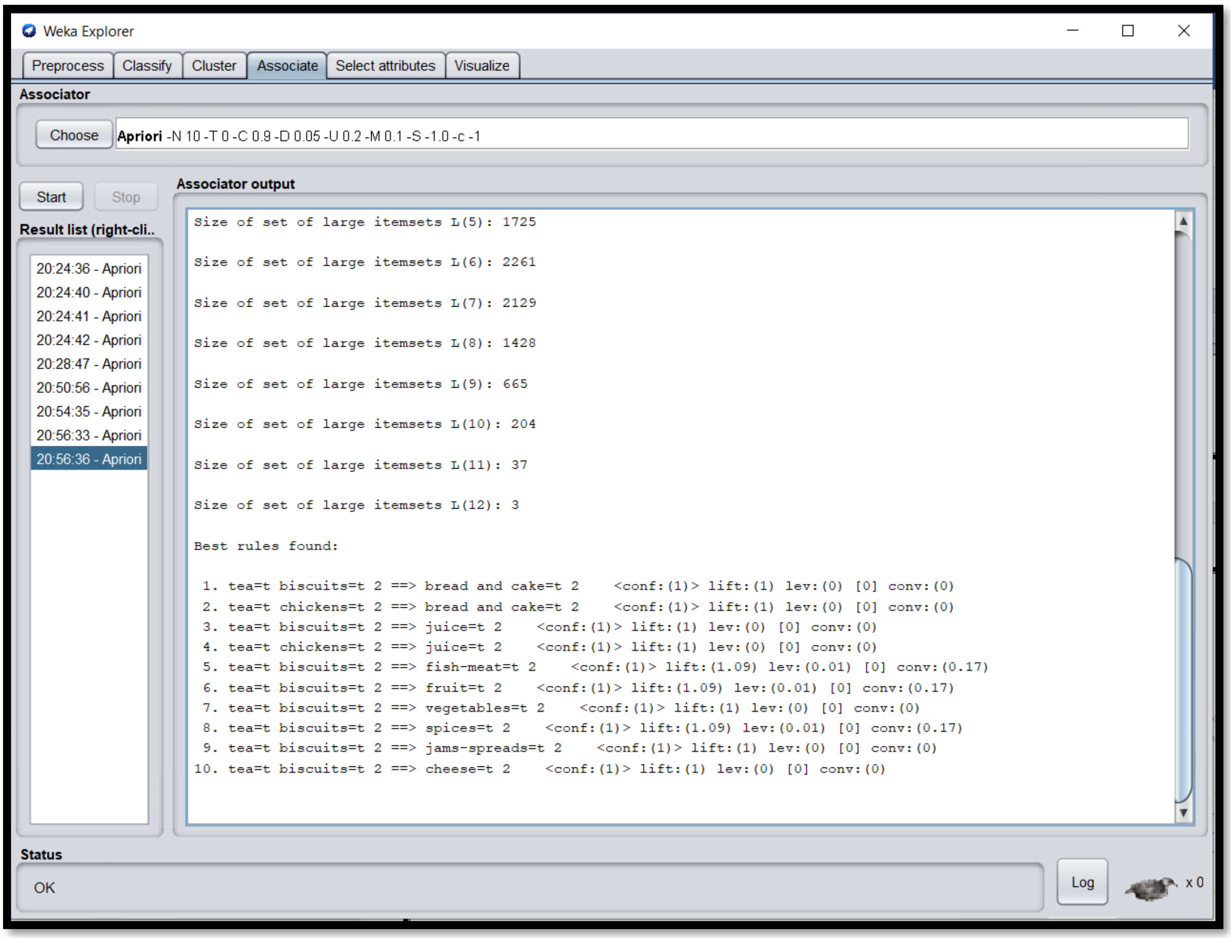
```
Size of set of large itemsets L(2): 36

Size of set of large itemsets L(3): 84

Size of set of large itemsets L(4): 126

Size of set of large itemsets L(5): 126

Size of set of large itemsets L(6): 84

Size of set of large itemsets L(7): 36

Size of set of large itemsets L(8): 9

Size of set of large itemsets L(9): 1

Best rules found:

 1. fish-meat=t fruit=t 10 ==> bread and cake=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 2. fish-meat=t spices=t 10 ==> bread and cake=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 3. fish-meat=t fruit=t 10 ==> juice=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 4. fish-meat=t spices=t 10 ==> juice=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 5. fish-meat=t fruit=t 10 ==> vegetables=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 6. fish-meat=t spices=t 10 ==> fruit=t 10    <conf:(1)> lift:(1.09) lev:(0.07) [0] conv:(0.83)
 7. fish-meat=t fruit=t 10 ==> spices=t 10    <conf:(1)> lift:(1.09) lev:(0.07) [0] conv:(0.83)
 8. fish-meat=t fruit=t 10 ==> jams-spreads=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 9. fish-meat=t fruit=t 10 ==> cheese=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
10. fish-meat=t fruit=t 10 ==> milk-cream=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
```

OK



```
Best rules found:

 1. fish-meat=t fruit=t 10 ==> bread and cake=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 2. fish-meat=t spices=t 10 ==> bread and cake=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 3. fish-meat=t fruit=t 10 ==> juice=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 4. fish-meat=t spices=t 10 ==> juice=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 5. fish-meat=t fruit=t 10 ==> vegetables=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 6. fish-meat=t spices=t 10 ==> fruit=t 10    <conf:(1)> lift:(1.09) lev:(0.07) [0] conv:(0.83)
 7. fish-meat=t fruit=t 10 ==> spices=t 10    <conf:(1)> lift:(1.09) lev:(0.07) [0] conv:(0.83)
 8. fish-meat=t fruit=t 10 ==> jams-spreads=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 9. fish-meat=t fruit=t 10 ==> cheese=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
10. fish-meat=t fruit=t 10 ==> milk-cream=t 10    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
```

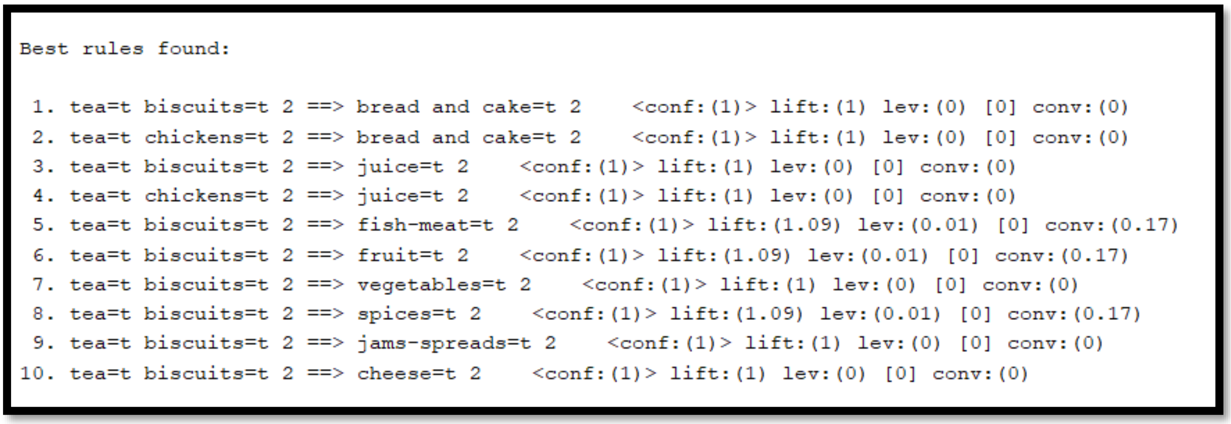
Confident = 0.9 and Support = 0.2



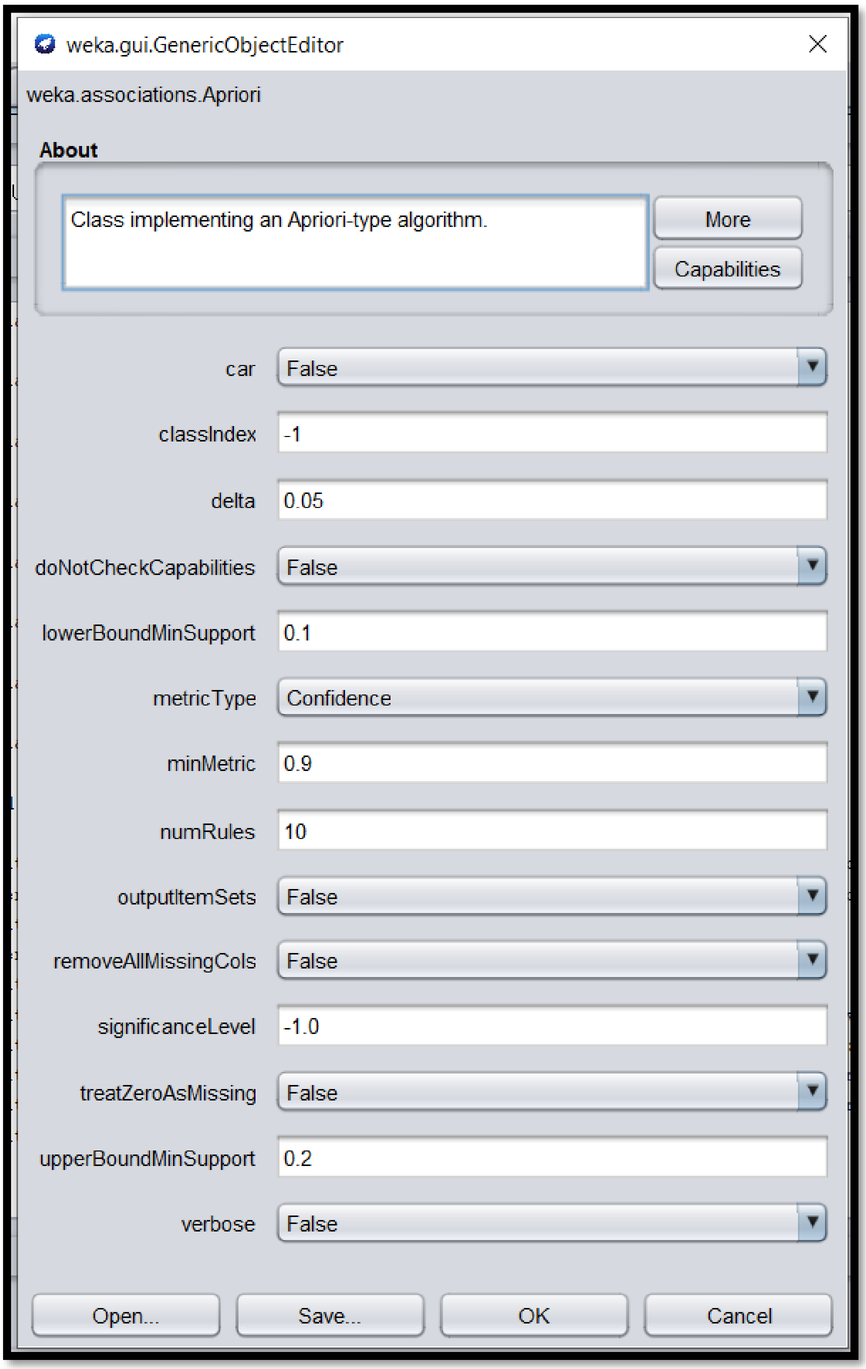
Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 0.2 -M 0.1 -S -1.0 -c -1

```
Size of set of large itemsets L(5): 1725

Size of set of large itemsets L(6): 2261

Size of set of large itemsets L(7): 2129

Size of set of large itemsets L(8): 1428

Size of set of large itemsets L(9): 665

Size of set of large itemsets L(10): 204

Size of set of large itemsets L(11): 37

Size of set of large itemsets L(12): 3

Best rules found:

 1. tea=t biscuits=t 2 ==> bread and cake=t 2    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 2. tea=t chickens=t 2 ==> bread and cake=t 2    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 3. tea=t biscuits=t 2 ==> juice=t 2    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 4. tea=t chickens=t 2 ==> juice=t 2    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 5. tea=t biscuits=t 2 ==> fish-meat=t 2    <conf:(1)> lift:(1.09) lev:(0.01) [0] conv:(0.17)
 6. tea=t biscuits=t 2 ==> fruit=t 2    <conf:(1)> lift:(1.09) lev:(0.01) [0] conv:(0.17)
 7. tea=t biscuits=t 2 ==> vegetables=t 2    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 8. tea=t biscuits=t 2 ==> spices=t 2    <conf:(1)> lift:(1.09) lev:(0.01) [0] conv:(0.17)
 9. tea=t biscuits=t 2 ==> jams-spreads=t 2    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
10. tea=t biscuits=t 2 ==> cheese=t 2    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
```

OK



```
Best rules found:

 1. tea=t biscuits=t 2 ==> bread and cake=t 2    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 2. tea=t chickens=t 2 ==> bread and cake=t 2    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 3. tea=t biscuits=t 2 ==> juice=t 2    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 4. tea=t chickens=t 2 ==> juice=t 2    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 5. tea=t biscuits=t 2 ==> fish-meat=t 2    <conf:(1)> lift:(1.09) lev:(0.01) [0] conv:(0.17)
 6. tea=t biscuits=t 2 ==> fruit=t 2    <conf:(1)> lift:(1.09) lev:(0.01) [0] conv:(0.17)
 7. tea=t biscuits=t 2 ==> vegetables=t 2    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
 8. tea=t biscuits=t 2 ==> spices=t 2    <conf:(1)> lift:(1.09) lev:(0.01) [0] conv:(0.17)
 9. tea=t biscuits=t 2 ==> jams-spreads=t 2    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
10. tea=t biscuits=t 2 ==> cheese=t 2    <conf:(1)> lift:(1) lev:(0) [0] conv:(0)
```



weka.gui.GenericObjectEditor

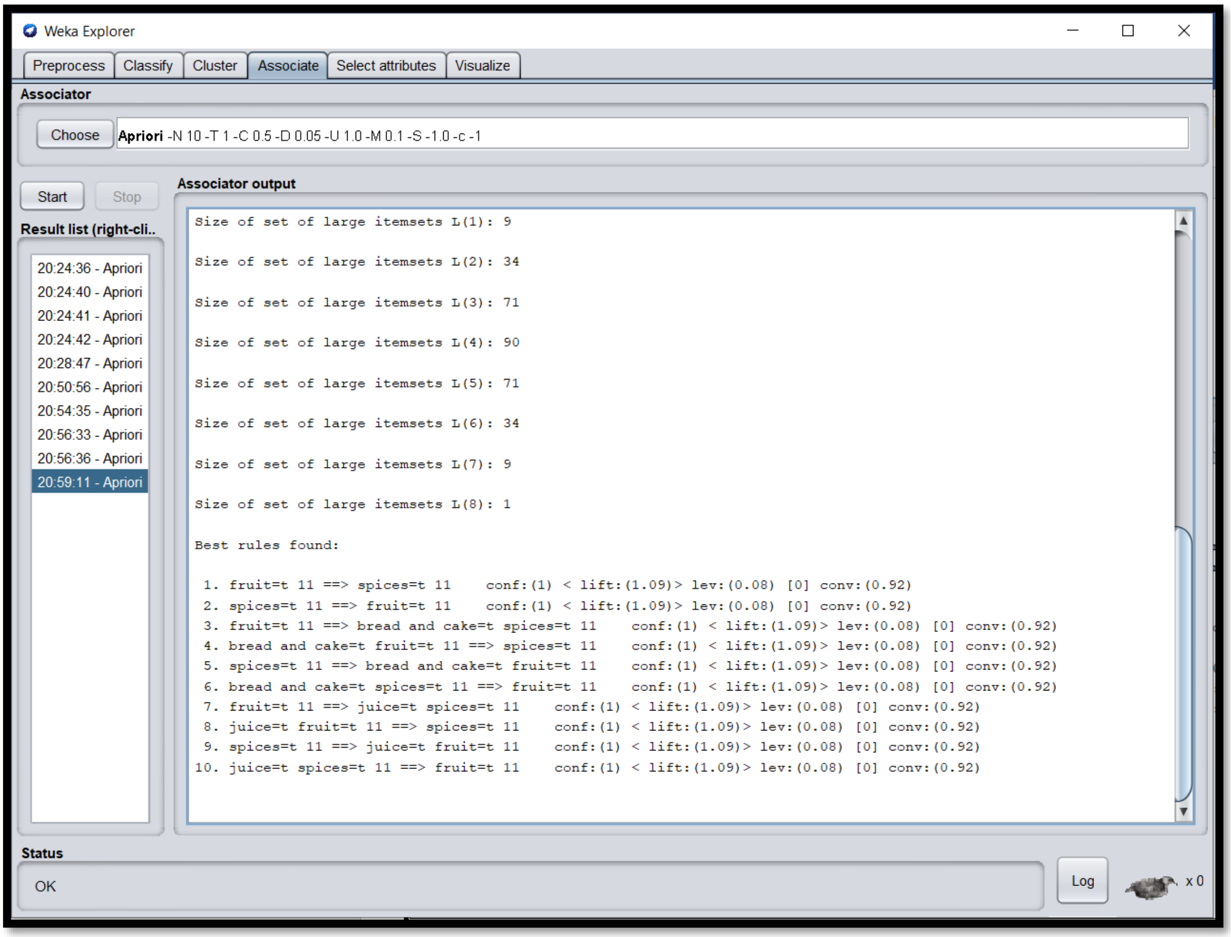
weka.associations.Apriori

About

Class implementing an Apriori-type algorithm.

car	False
classIndex	-1
delta	0.05
doNotCheckCapabilities	False
lowerBoundMinSupport	0.1
metricType	Confidence
minMetric	0.9
numRules	10
outputItemSets	False
removeAllMissingCols	False
significanceLevel	-1.0
treatZeroAsMissing	False
upperBoundMinSupport	0.2
verbose	False

Lift=0.5 and Support = 1.0



Apriori -N 10 -T 1 -C 0.5 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

```
Size of set of large itemsets L(1): 9

Size of set of large itemsets L(2): 34

Size of set of large itemsets L(3): 71

Size of set of large itemsets L(4): 90

Size of set of large itemsets L(5): 71

Size of set of large itemsets L(6): 34

Size of set of large itemsets L(7): 9

Size of set of large itemsets L(8): 1

Best rules found:

 1. fruit=t 11 ==> spices=t 11    conf:(1) < lift:(1.09)> lev:(0.08) [0] conv:(0.92)
 2. spices=t 11 ==> fruit=t 11    conf:(1) < lift:(1.09)> lev:(0.08) [0] conv:(0.92)
 3. fruit=t 11 ==> bread and cake=t spices=t 11    conf:(1) < lift:(1.09)> lev:(0.08) [0] conv:(0.92)
 4. bread and cake=t fruit=t 11 ==> spices=t 11    conf:(1) < lift:(1.09)> lev:(0.08) [0] conv:(0.92)
 5. spices=t 11 ==> bread and cake=t fruit=t 11    conf:(1) < lift:(1.09)> lev:(0.08) [0] conv:(0.92)
 6. bread and cake=t spices=t 11 ==> fruit=t 11    conf:(1) < lift:(1.09)> lev:(0.08) [0] conv:(0.92)
 7. fruit=t 11 ==> juice=t spices=t 11    conf:(1) < lift:(1.09)> lev:(0.08) [0] conv:(0.92)
 8. juice=t fruit=t 11 ==> spices=t 11    conf:(1) < lift:(1.09)> lev:(0.08) [0] conv:(0.92)
 9. spices=t 11 ==> juice=t fruit=t 11    conf:(1) < lift:(1.09)> lev:(0.08) [0] conv:(0.92)
10. juice=t spices=t 11 ==> fruit=t 11    conf:(1) < lift:(1.09)> lev:(0.08) [0] conv:(0.92)
```

OK

Conclusion:

Apriori algorithm is an efficient algorithm that scans the database only once. It reduces the size of the item sets in the database considerably providing a good performance. Thus, data mining helps consumers and industries better in the decision-making process. Also I learn how to implementation of apriori algorithm in Weka tool.