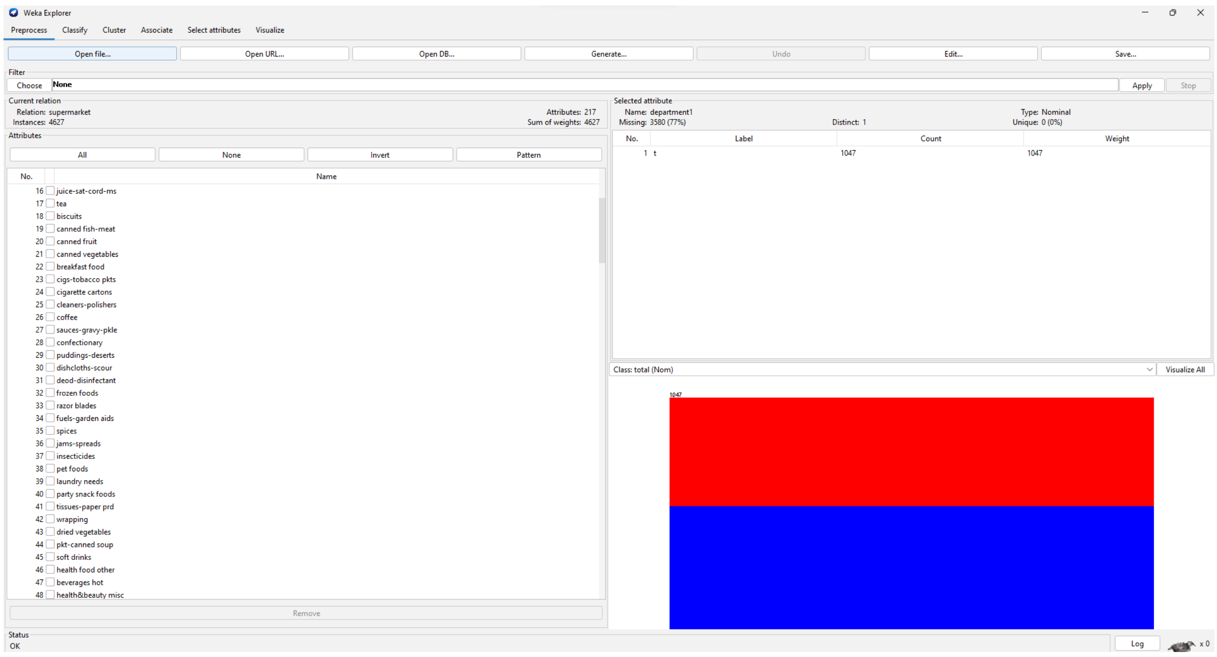
VIETNAMESE - GERMAN UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEER

Lab 5 Report

Student:	Le Thanh Hai - 10421016
Student:	Trinh The Hao - 10421017
Student:	Vu Ngoc Quang - 10421103

07 / 2022

1 Exercise 2.1,2,3,4,5,6



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

Filter: None

No.	Name
16	juice-sat-cord-ms
17	tea
18	biscuits
19	canned fish-meat
20	canned fruit
21	canned vegetables
22	breakfast food
23	cigs-tobacco pkts
24	cigarette cartons
25	cleaners-polishers
26	coffee
27	sauces-gravy-pkle
28	confectionary
29	puddings-deserts
30	dishcloths-scour
31	deod-disinfectant
32	frozen foods
33	razor blades
34	fuels-garden aids
35	spices
36	jams-spreads
37	insecticides
38	pet foods
39	laundry needs
40	party snack foods
41	tissues-paper prd
42	wrapping
43	dried vegetables
44	pkt-canned soup
45	soft drinks
46	health food other
47	beverages hot
48	health&beauty misc

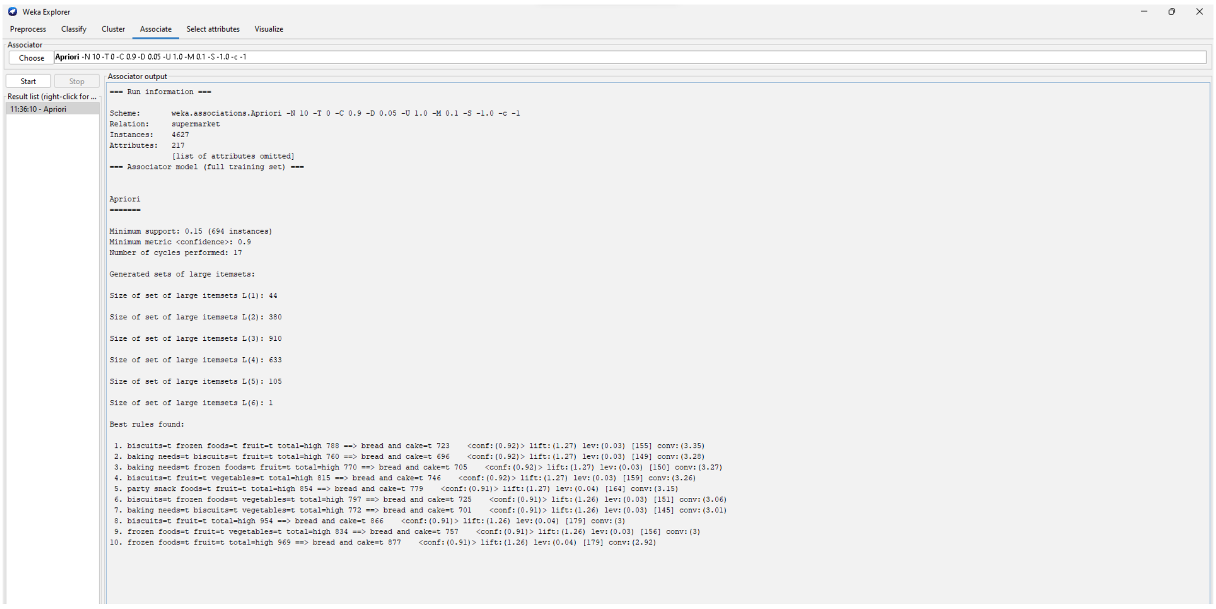
Selected attribute
Name: department1
Missing: 3580 (77%)
Distinct: 1
Type: Nominal
Unique: 0 (0%)

No.	Label	Count	Weight
1	t	1047	1047

Class: total (Nom)

Status
OK

Figure 1: 2.1 load



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

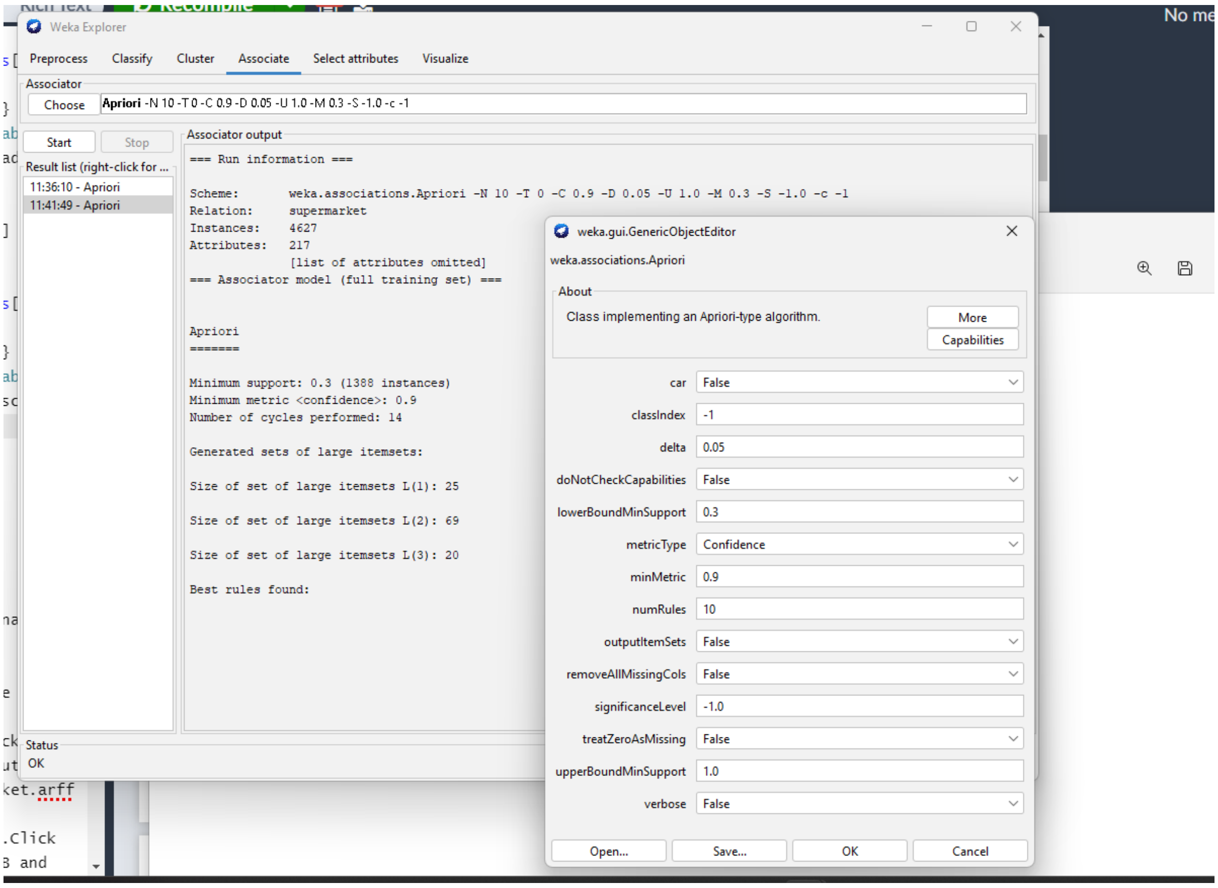
Result list: 11:36:10 - Apriori

```
=== 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:    4627
Attributes:   217
              [list of attributes omitted]
=== Associator model (full training set) ===


Apriori
=======

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) [155] conv:(3.35)
 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) [149] conv:(3.28)
 3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705    <conf:(0.92)> lift:(1.27) lev:(0.03) [150] conv:(3.27)
 4. biscuits=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746    <conf:(0.92)> lift:(1.27) lev:(0.03) [159] conv:(3.26)
 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] conv:(3.15)
 6. biscuits=t frozen foods=t vegetables=t total=high 797 ==> bread and cake=t 725    <conf:(0.91)> lift:(1.26) lev:(0.03) [151] conv:(3.06)
 7. baking needs=t biscuits=t vegetables=t total=high 772 ==> bread and cake=t 701    <conf:(0.91)> lift:(1.26) lev:(0.03) [145] conv:(3.01)
 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:(0.03) [156] conv:(3)
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:(2.92)
```

Figure 2: 2.2 describe

Brief Explanation:

To load data ,In the WEKA explorer, open the Preprocess tab, click on the Open file ... button and select supermarket.arff database from the installation folder.Click on the Associate



Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.3 -S -1.0 -c -1

```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.3 -S -1.0 -c -1
Relation:     supermarket
Instances:    4627
Attributes:   217
              [list of attributes omitted]
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.3 (1388 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 14

Generated sets of large itemsets:

Size of set of large itemsets L(1): 25

Size of set of large itemsets L(2): 69

Size of set of large itemsets L(3): 20

Best rules found:
```

weka.associations.Apriori

Class implementing an Apriori-type algorithm.

car	False
classIndex	-1
delta	0.05
doNotCheckCapabilities	False
lowerBoundMinSupport	0.3
metricType	Confidence
minMetric	0.9
numRules	10
outputItemSets	False
removeAllMissingCols	False
significanceLevel	-1.0
treatZeroAsMissing	False
upperBoundMinSupport	1.0
verbose	False

Figure 3: 2.3 2.4 2.5 2.6

TAB and click on the Choose button. Select the Apriori association .

To describe the dataset, you need to load it in Weka and examine its attributes and instances. In Weka, you can open the dataset file (supermarket.arff) using the "Explorer" interface. The number of instances refers to the number of rows or examples in the dataset, while the number of items refers to the number of attributes or columns in the dataset.

Run Apriori algorithm with minsup=0.3 and minconf=0.9: In Weka, after loading the dataset, you can navigate to the "Associate" tab in the "Explorer" interface. Select the "Apriori" algorithm and set the minimum support (minsup) to 0.3 and the minimum confidence (minconf) to 0.9. Run the algorithm, and Weka will generate association rules based on these parameters.

Report top ten rules: Once the Apriori algorithm finishes running, Weka will display the generated association rules. You can sort them based on various metrics such as confidence, support, or lift. To report the top ten rules, you can sort the rules based on the desired metric and select the top ten rules with the highest values.

Run Apriori with different minsup and minconf: To run the Apriori algorithm with different minimum support and minimum confidence values, you can repeat the process described in step 2.3. Simply change the minsup and minconf parameters to the desired values, and Weka will generate new association rules based on these updated parameters.

Report top ten rules: Similarly, after running the updated Apriori algorithm, you can sort and report the top ten rules based on the desired metric as explained in step 2.4.

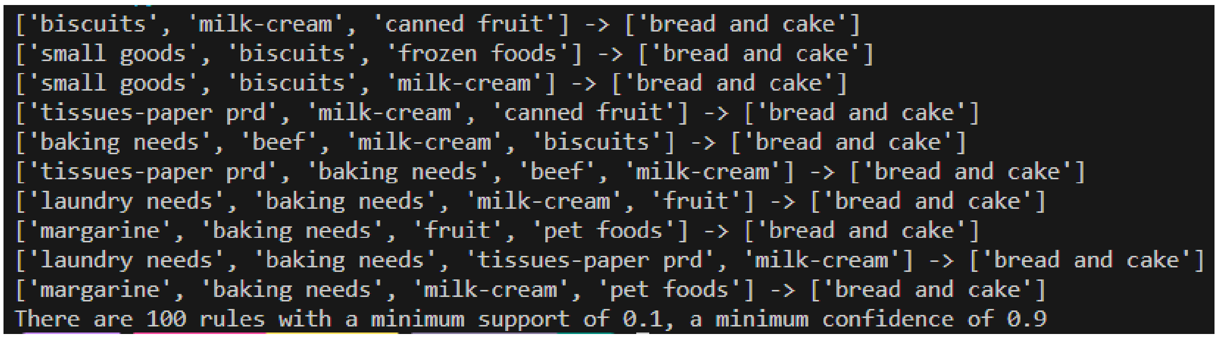
Please note that the specific steps and options may vary slightly depending on the version of Weka you are using. It's always a good idea to consult the Weka documentation or user guides for detailed instructions on how to perform these tasks within the software.

2 Exercise 2.7 & 2.8 & 2.9 & 2.10

```python
from scipy.io import arff
import pandas as pd
from scipy.io import arff
import apyori
from apyori import apriori

# Load the data set.
data = arff.loadarff("supermarket.arff")
df = pd.DataFrame(data[0])
```

```

# drop the total column
df = df.drop(['total'],axis=1)

# convert the data to a list of attributes
attributes = list(df.columns)

# transform the data into strings and into a numpy matrix for easier ↩
    handling
sales = df.to_numpy().astype(str)

transactions = [ [] for _ in range(df.shape[0]) ]  #Empty lists to be ↩
    filled with the specific products purchased.

#We iterate over every element in the sales matrix and, if a product ↩
    belongs to a transaction, we save it.
for i in range(df.shape[0]):
     for j in range(df.shape[1]):
         if sales[i,j] == "t":
             transactions[i].append(attributes[j])


mins = 0.1 # minimum support
minc = 0.9 # minimum confidence

# We use the apriori algorithm to find the association rules.
results = list(apriori(transactions,min_support=mins,min_confidence=↩
    minc))

for i in range(0,10):
     print(list(results[i][2][0][0]),'->',list(results[i][2][0][1]))

print('There are '+str(len(results))+' rules with a minimum support of↩
     '+str(mins)+
      ', a minimum confidence of ' +str(minc))
```

Brief Explanation: The code loads a supermarket dataset from an ARFF file and converts it into a pandas DataFrame. It removes the "total" column from the DataFrame. The data is transformed into a list of transactions, where each transaction represents a list of products purchased. The code defines minimum support (mins) and minimum confidence (minc) thresholds. The Apriori algorithm is applied to the transactions list to find association rules that meet the minimum support and minimum confidence criteria. The code prints the first 10 association rules found, showing the antecedent (products purchased together) and consequent (product purchased after the antecedent). Finally, the code prints the total number of association rules found with the specified minimum support and minimum confidence thresholds.

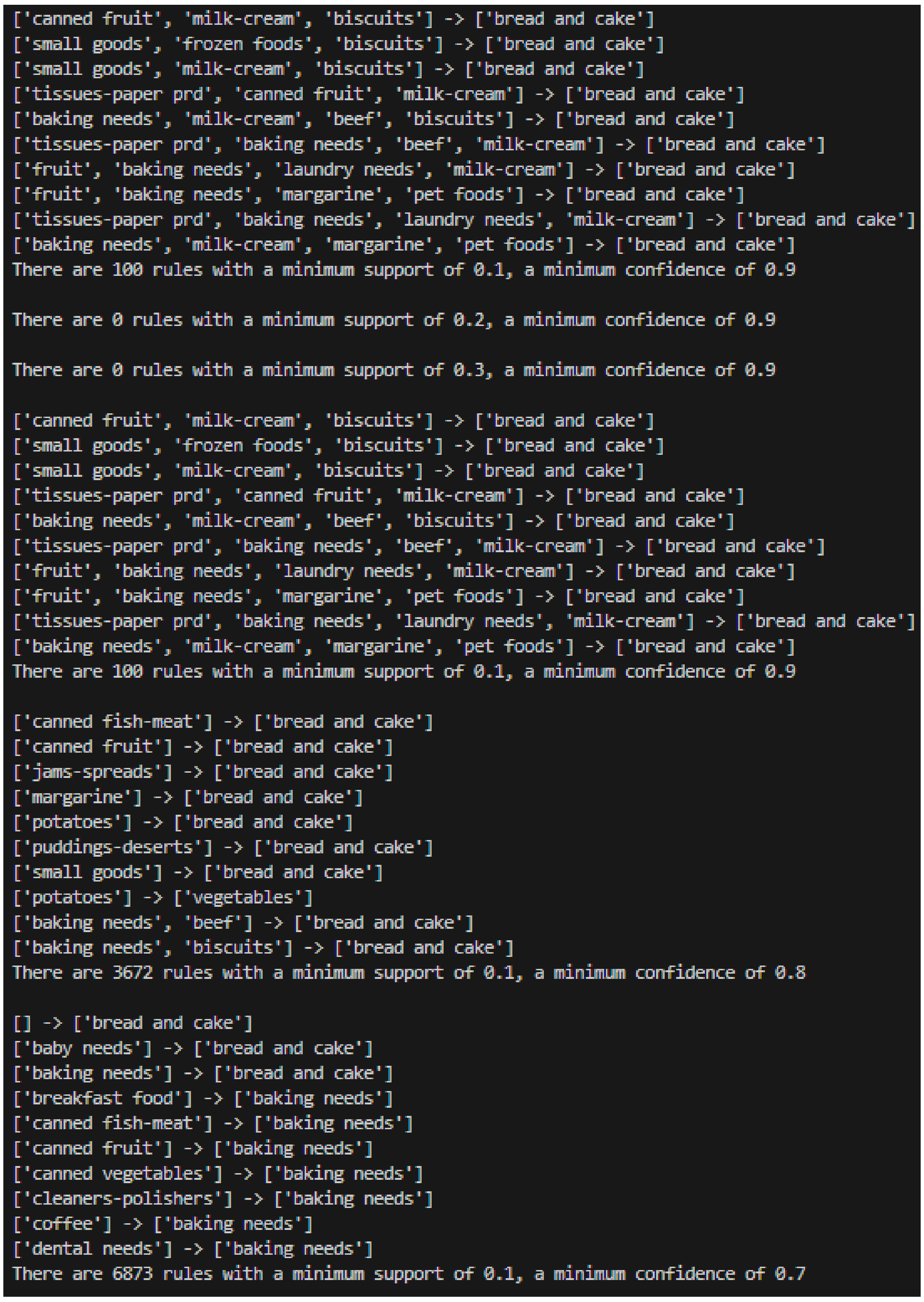


```
['biscuits', 'milk-cream', 'canned fruit'] -> ['bread and cake']
['small goods', 'biscuits', 'frozen foods'] -> ['bread and cake']
['small goods', 'biscuits', 'milk-cream'] -> ['bread and cake']
['tissues-paper prd', 'milk-cream', 'canned fruit'] -> ['bread and cake']
['baking needs', 'beef', 'milk-cream', 'biscuits'] -> ['bread and cake']
['tissues-paper prd', 'baking needs', 'beef', 'milk-cream'] -> ['bread and cake']
['laundry needs', 'baking needs', 'milk-cream', 'fruit'] -> ['bread and cake']
['margarine', 'baking needs', 'fruit', 'pet foods'] -> ['bread and cake']
['laundry needs', 'baking needs', 'tissues-paper prd', 'milk-cream'] -> ['bread and cake']
['margarine', 'baking needs', 'milk-cream', 'pet foods'] -> ['bread and cake']
There are 100 rules with a minimum support of 0.1, a minimum confidence of 0.9
```

3 Exercise 2.11 & 2.12

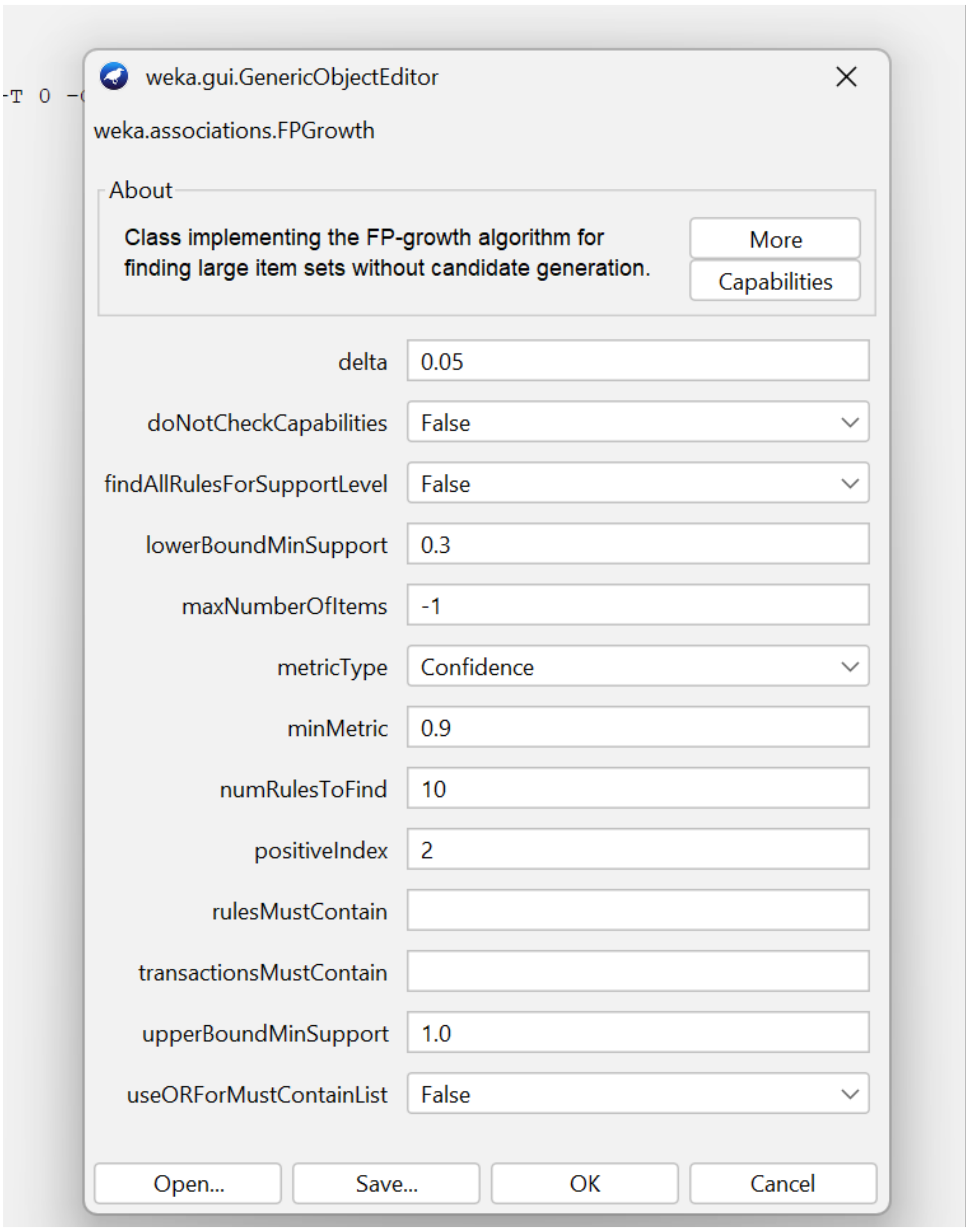
```python
from scipy.io import arff
import pandas as pd
from scipy.io import arff
import apyori
from apyori import apriori

# Load the data set.
data = arff.loadarff("supermarket.arff")
df = pd.DataFrame(data[0])

# drop the total column
df = df.drop(['total'],axis=1)

# convert the data to a list of attributes
attributes = list(df.columns)

# transform the data into strings and into a numpy matrix for easier handling
sales = df.to_numpy().astype(str)

transactions = [ [] for _ in range(df.shape[0]) ]  #Empty lists to be filled with the specific products purchased.

#We iterate over every element in the sales matrix and, if a product belongs to a transaction, we save it.
for i in range(df.shape[0]):
    for j in range(df.shape[1]):
        if sales[i,j] == "t":
            transactions[i].append(attributes[j])

```

```python

mins = [0.1,0.2,0.3] # minimum support
minc = 0.9 # minimum confidence

for m in mins:
    # We use the apriori algorithm to find the association rules.
    results = list(apriori(transactions,min_support=m,min_confidence=minc))
    if len(results) > 10:
        for i in range(0,10):
            print(list(results[i][2][0][0]),'->',list(results[i][2][0][1]))
    else:
        for i in range(0,len(results)):
            print(list(results[i][2][0][0]),'->',list(results[i][2][0][1]))

    print('There are '+str(len(results))+' rules with a minimum support of '+str(m)+
          ', a minimum confidence of ' +str(minc))
    print('')

mins = 0.1 # minimum support
minc = [0.9,0.8,0.7] # minimum confidence

for m in minc:
    # We use the apriori algorithm to find the association rules.
    results = list(apriori(transactions,min_support=mins,min_confidence=m))
    if len(results) > 10:
        for i in range(0,10):
            print(list(results[i][2][0][0]),'->',list(results[i][2][0][1]))
    else:
        for i in range(0,len(results)):
            print(list(results[i][2][0][0]),'->',list(results[i][2][0][1]))

    print('There are '+str(len(results))+' rules with a minimum support of '+str(mins)+
          ', a minimum confidence of ' +str(m))
    print('')
```



```
['canned fruit', 'milk-cream', 'biscuits'] -> ['bread and cake']
['small goods', 'frozen foods', 'biscuits'] -> ['bread and cake']
['small goods', 'milk-cream', 'biscuits'] -> ['bread and cake']
['tissues-paper prd', 'canned fruit', 'milk-cream'] -> ['bread and cake']
['baking needs', 'milk-cream', 'beef', 'biscuits'] -> ['bread and cake']
['tissues-paper prd', 'baking needs', 'beef', 'milk-cream'] -> ['bread and cake']
['fruit', 'baking needs', 'laundry needs', 'milk-cream'] -> ['bread and cake']
['fruit', 'baking needs', 'margarine', 'pet foods'] -> ['bread and cake']
['tissues-paper prd', 'baking needs', 'laundry needs', 'milk-cream'] -> ['bread and cake']
['baking needs', 'milk-cream', 'margarine', 'pet foods'] -> ['bread and cake']
There are 100 rules with a minimum support of 0.1, a minimum confidence of 0.9

There are 0 rules with a minimum support of 0.2, a minimum confidence of 0.9

There are 0 rules with a minimum support of 0.3, a minimum confidence of 0.9

['canned fruit', 'milk-cream', 'biscuits'] -> ['bread and cake']
['small goods', 'frozen foods', 'biscuits'] -> ['bread and cake']
['small goods', 'milk-cream', 'biscuits'] -> ['bread and cake']
['tissues-paper prd', 'canned fruit', 'milk-cream'] -> ['bread and cake']
['baking needs', 'milk-cream', 'beef', 'biscuits'] -> ['bread and cake']
['tissues-paper prd', 'baking needs', 'beef', 'milk-cream'] -> ['bread and cake']
['fruit', 'baking needs', 'laundry needs', 'milk-cream'] -> ['bread and cake']
['fruit', 'baking needs', 'margarine', 'pet foods'] -> ['bread and cake']
['tissues-paper prd', 'baking needs', 'laundry needs', 'milk-cream'] -> ['bread and cake']
['baking needs', 'milk-cream', 'margarine', 'pet foods'] -> ['bread and cake']
There are 100 rules with a minimum support of 0.1, a minimum confidence of 0.9

['canned fish-meat'] -> ['bread and cake']
['canned fruit'] -> ['bread and cake']
['jams-spreads'] -> ['bread and cake']
['margarine'] -> ['bread and cake']
['potatoes'] -> ['bread and cake']
['puddings-deserts'] -> ['bread and cake']
['small goods'] -> ['bread and cake']
['potatoes'] -> ['vegetables']
['baking needs', 'beef'] -> ['bread and cake']
['baking needs', 'biscuits'] -> ['bread and cake']
There are 3672 rules with a minimum support of 0.1, a minimum confidence of 0.8

[] -> ['bread and cake']
['baby needs'] -> ['bread and cake']
['baking needs'] -> ['bread and cake']
['breakfast food'] -> ['baking needs']
['canned fish-meat'] -> ['baking needs']
['canned fruit'] -> ['baking needs']
['canned vegetables'] -> ['baking needs']
['cleaners-polishers'] -> ['baking needs']
['coffee'] -> ['baking needs']
['dental needs'] -> ['baking needs']
There are 6873 rules with a minimum support of 0.1, a minimum confidence of 0.7
```

4 Exercise 2.13 & 2.14

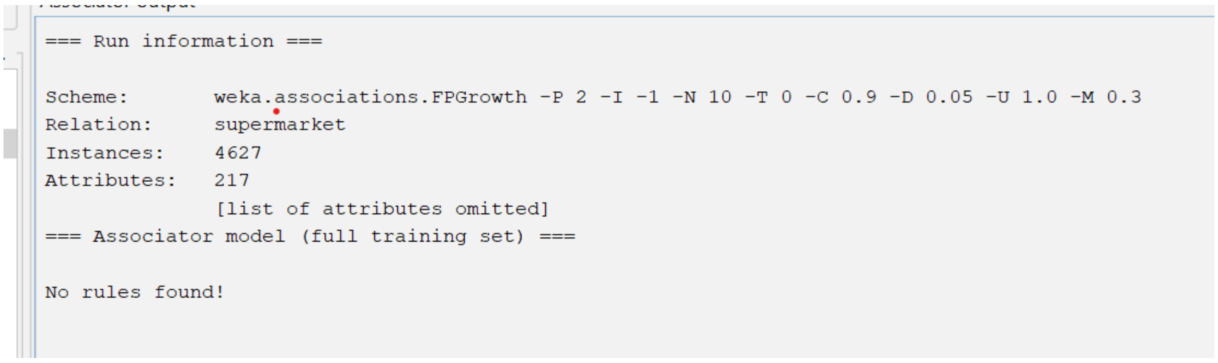

weka.gui.GenericObjectEditor

weka.associations.FPGrowth

About

Class implementing the FP-growth algorithm for finding large item sets without candidate generation.

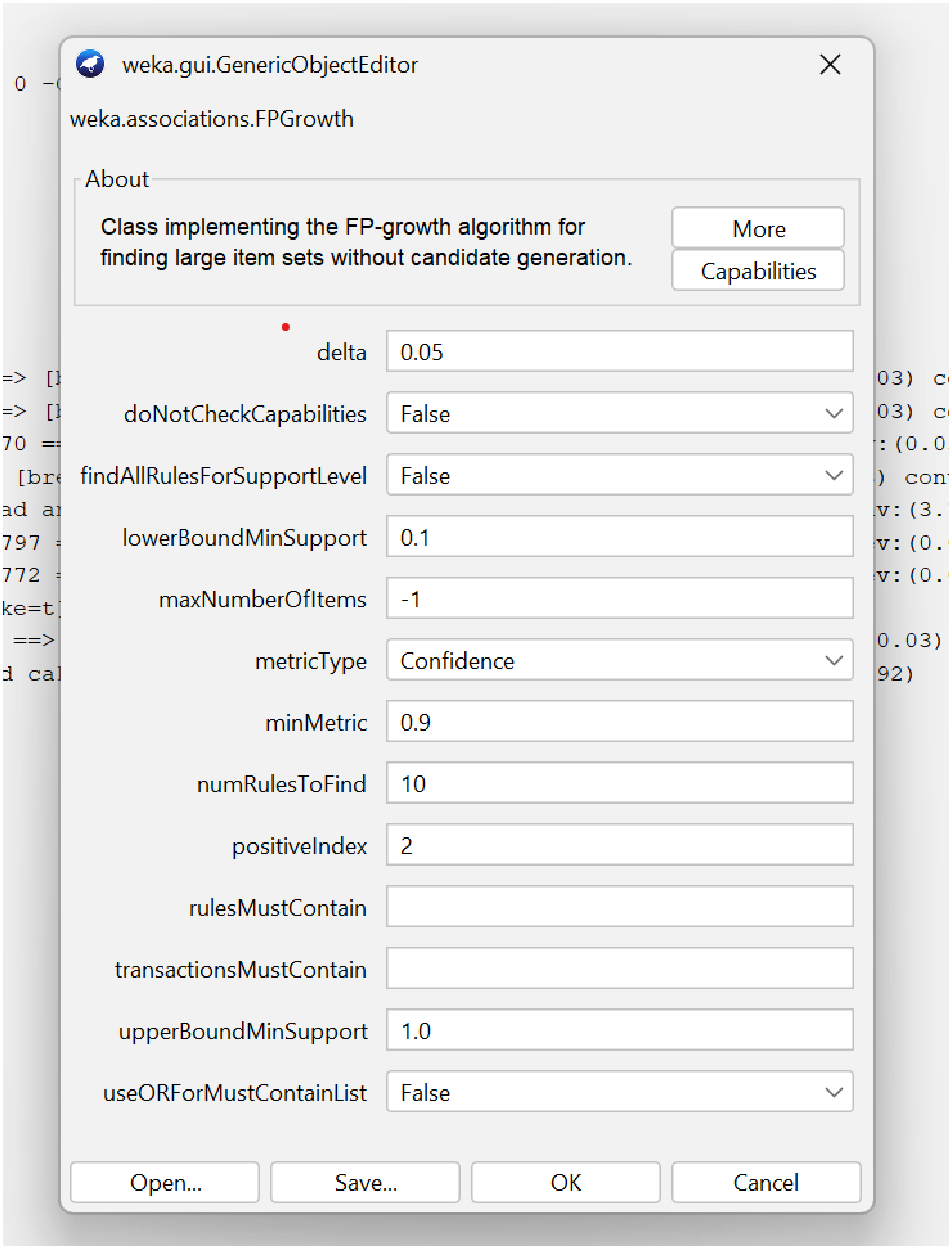
delta	0.05
doNotCheckCapabilities	False
findAllRulesForSupportLevel	False
lowerBoundMinSupport	0.3
maxNumberOfItems	-1
metricType	Confidence
minMetric	0.9
numRulesToFind	10
positiveIndex	2
rulesMustContain	
transactionsMustContain	
upperBoundMinSupport	1.0
useORForMustContainList	False


```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.3
Relation:     supermarket
Instances:    4627
Attributes:   217
              [list of attributes omitted]
=== Associator model (full training set) ===

No rules found!
```

Description:

We tried to set the minsup value to 0.3 and the mincof value to 0.9 and the output we got was there were no rules found. We will try to figure out what happened by setting the difference values in the next part.

5 Exercise 2.15 & 2.16



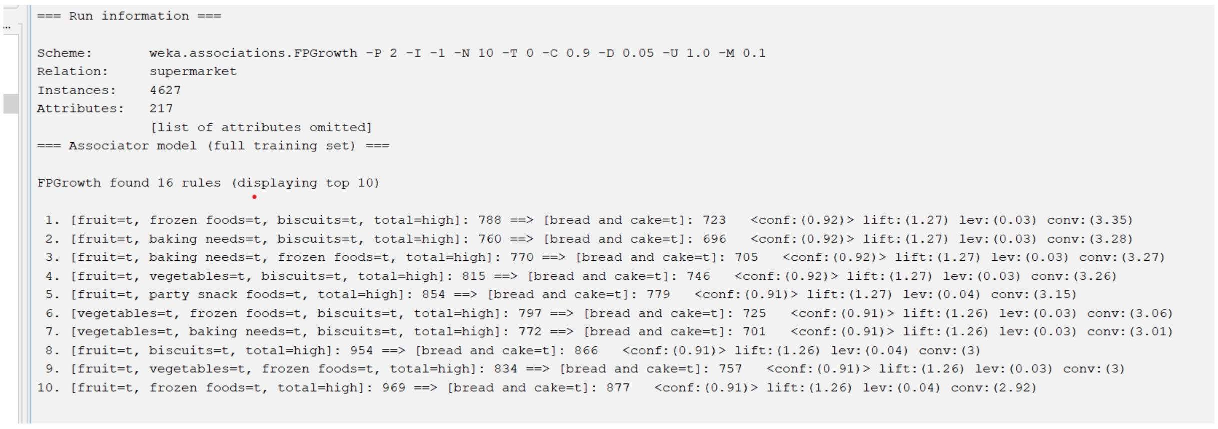
weka.gui.GenericObjectEditor

weka.associations.FPGrowth

About

Class implementing the FP-growth algorithm for finding large item sets without candidate generation.

delta	0.05
doNotCheckCapabilities	False
findAllRulesForSupportLevel	False
lowerBoundMinSupport	0.1
maxNumberOfItems	-1
metricType	Confidence
minMetric	0.9
numRulesToFind	10
positiveIndex	2
rulesMustContain	
transactionsMustContain	
upperBoundMinSupport	1.0
useORForMustContainList	False



```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1
Relation:     supermarket
Instances:    4627
Attributes:   217
              [list of attributes omitted]
=== Associator model (full training set) ===

FPGrowth found 16 rules (displaying top 10)

 1. [fruit=t, frozen foods=t, biscuits=t, total=high]: 788 ==> [bread and cake=t]: 723   <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.35)
 2. [fruit=t, baking needs=t, biscuits=t, total=high]: 760 ==> [bread and cake=t]: 696   <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.28)
 3. [fruit=t, baking needs=t, frozen foods=t, total=high]: 770 ==> [bread and cake=t]: 705   <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.27)
 4. [fruit=t, vegetables=t, biscuits=t, total=high]: 815 ==> [bread and cake=t]: 746   <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.26)
 5. [fruit=t, party snack foods=t, total=high]: 854 ==> [bread and cake=t]: 779   <conf:(0.91)> lift:(1.27) lev:(0.04) conv:(3.15)
 6. [vegetables=t, frozen foods=t, biscuits=t, total=high]: 797 ==> [bread and cake=t]: 725   <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3.06)
 7. [vegetables=t, baking needs=t, biscuits=t, total=high]: 772 ==> [bread and cake=t]: 701   <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3.01)
 8. [fruit=t, biscuits=t, total=high]: 954 ==> [bread and cake=t]: 866   <conf:(0.91)> lift:(1.26) lev:(0.04) conv:(3)
 9. [fruit=t, vegetables=t, frozen foods=t, total=high]: 834 ==> [bread and cake=t]: 757   <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3)
10. [fruit=t, frozen foods=t, total=high]: 969 ==> [bread and cake=t]: 877   <conf:(0.91)> lift:(1.26) lev:(0.04) conv:(2.92)
```

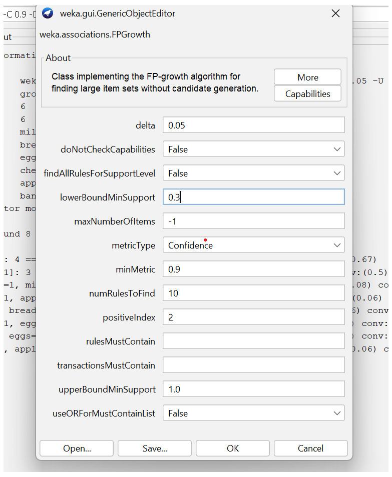
Description:

We tried to set a new value for the minsup to 0.1 and this time it did show out the top ten rules.

It turned out that When you set minsup to 0.3, the algorithm will only consider itemsets that appear in at least 30 percent of the transactions. If there are no itemsets that appear in at least 30 percent of the transactions, then the algorithm will not find any rules.

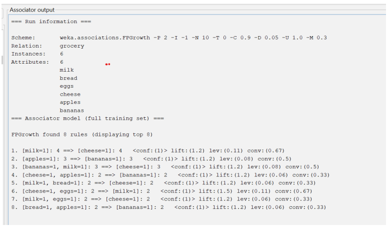
When you set minsup to 0.1, the algorithm will consider itemsets that appear in at least 10 percent of the transactions. This means that the algorithm is more likely to find rules, even if the rules are not very strong.

6 Exercise 2.17 & 2.18



Class implementing the FP-growth algorithm for finding large item sets without candidate generation.

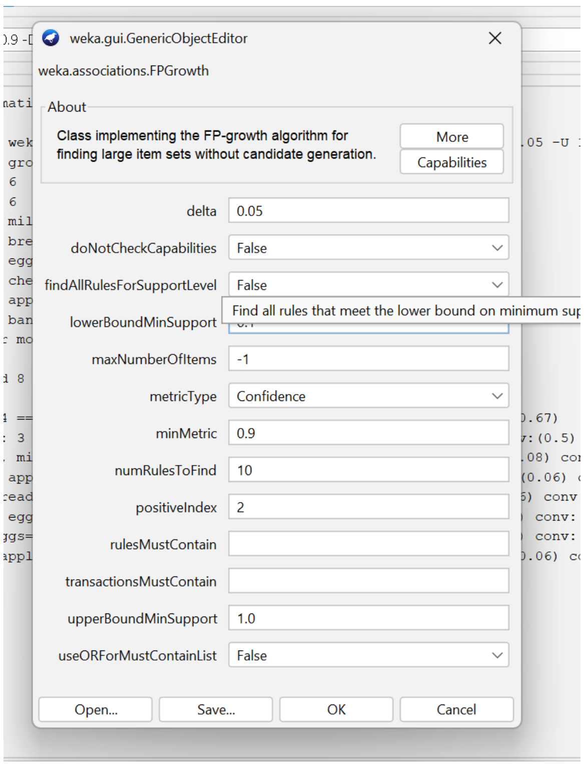
```
delta                          0.05
doNotCheckCapabilities         False
findAllRulesForSupportLevel    False
lowerBoundMinSupport           0.3
maxNumberOfItems               -1
metricType                     Confidence
minMetric                      0.9
numRulesToFind                 10
positiveIndex                  2
rulesMustContain
transactionsMustContain
upperBoundMinSupport           1.0
useORForMustContainList        False
```



```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.3
Relation:     grocery
Instances:    6
Attributes:   6
              milk
              bread
              eggs
              cheese
              apples
              bananas
=== Associator model (full training set) ===

FPGrowth found 8 rules (displaying top 8)

1. [milk=1]: 4 ==> [cheese=1]: 4   <conf:(1)> lift:(1.2) lev:(0.11) conv:(0.67)
2. [apples=1]: 3 ==> [bananas=1]: 3   <conf:(1)> lift:(1.2) lev:(0.08) conv:(0.5)
3. [bananas=1, milk=1]: 3 ==> [cheese=1]: 3   <conf:(1)> lift:(1.2) lev:(0.08) conv:(0.5)
4. [cheese=1, apples=1]: 2 ==> [bananas=1]: 2   <conf:(1)> lift:(1.2) lev:(0.06) conv:(0.33)
5. [milk=1, bread=1]: 2 ==> [cheese=1]: 2   <conf:(1)> lift:(1.2) lev:(0.06) conv:(0.33)
6. [cheese=1, eggs=1]: 2 ==> [milk=1]: 2   <conf:(1)> lift:(1.5) lev:(0.11) conv:(0.67)
7. [milk=1, eggs=1]: 2 ==> [cheese=1]: 2   <conf:(1)> lift:(1.2) lev:(0.06) conv:(0.33)
8. [bread=1, apples=1]: 2 ==> [bananas=1]: 2   <conf:(1)> lift:(1.2) lev:(0.06) conv:(0.33)
```

Description: We created our own dataset and the first one was with minsup =0.3

Description: We then tried it with other minsup =0.1



weka.gui.GenericObjectEditor

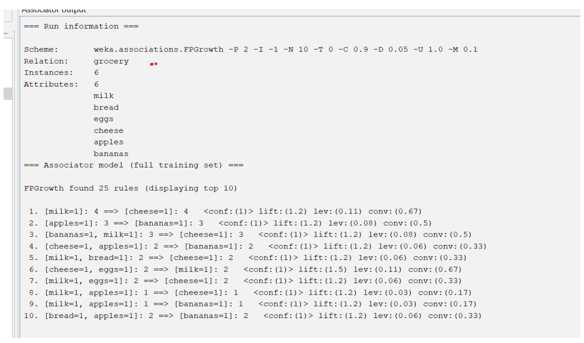
weka.associations.FPGrowth

About

Class implementing the FP-growth algorithm for finding large item sets without candidate generation.

delta	0.05
doNotCheckCapabilities	False
findAllRulesForSupportLevel	False
lowerBoundMinSupport	0.1
maxNumberOfItems	-1
metricType	Confidence
minMetric	0.9
numRulesToFind	10
positiveIndex	2
rulesMustContain	
transactionsMustContain	
upperBoundMinSupport	1.0
useORForMustContainList	False

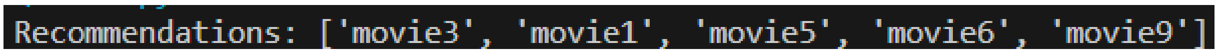
Find all rules that meet the lower bound on minimum sup



```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1
Relation:     grocery
Instances:    6
Attributes:   6
              milk
              bread
              eggs
              cheese
              apples
              bananas
=== Associator model (full training set) ===

FPGrowth found 25 rules (displaying top 10)

 1. [milk=1]: 4 ==> [cheese=1]: 4   <conf:(1)> lift:(1.2) lev:(0.11) conv:(0.67)
 2. [apples=1]: 3 ==> [bananas=1]: 3   <conf:(1)> lift:(1.2) lev:(0.08) conv:(0.5)
 3. [bananas=1, milk=1]: 3 ==> [cheese=1]: 3   <conf:(1)> lift:(1.2) lev:(0.08) conv:(0.5)
 4. [cheese=1, apples=1]: 2 ==> [bananas=1]: 2   <conf:(1)> lift:(1.2) lev:(0.06) conv:(0.33)
 5. [milk=1, bread=1]: 2 ==> [cheese=1]: 2   <conf:(1)> lift:(1.2) lev:(0.06) conv:(0.33)
 6. [cheese=1, eggs=1]: 2 ==> [milk=1]: 2   <conf:(1)> lift:(1.5) lev:(0.11) conv:(0.67)
 7. [milk=1, eggs=1]: 2 ==> [cheese=1]: 2   <conf:(1)> lift:(1.2) lev:(0.06) conv:(0.33)
 8. [milk=1, apples=1]: 1 ==> [cheese=1]: 1   <conf:(1)> lift:(1.2) lev:(0.03) conv:(0.17)
 9. [milk=1, apples=1]: 1 ==> [bananas=1]: 1   <conf:(1)> lift:(1.2) lev:(0.03) conv:(0.17)
10. [bread=1, apples=1]: 2 ==> [bananas=1]: 2   <conf:(1)> lift:(1.2) lev:(0.06) conv:(0.33)
```

7 Exercise 2.19 & 2.20

```python
# Sample dataset for movie recommendations
dataset = {
    'movie1': {'action': 1, 'thriller': 1, 'comedy': 0, 'romance': 0,'adventure':0},
    'movie2': {'action': 0, 'thriller': 0, 'comedy': 1, 'romance': 1,'adventure':0},
    'movie3': {'action': 1, 'thriller': 0, 'comedy': 0, 'romance': 0,'adventure':1},
    'movie4': {'action': 0, 'thriller': 0, 'comedy': 1, 'romance': 0,'adventure':0},
    'movie5': {'action': 0, 'thriller': 1, 'comedy': 0, 'romance': 0,'adventure':1},
    'movie6': {'action': 1, 'thriller': 0, 'comedy': 1, 'romance': 0,'adventure':0},
    'movie7': {'action': 0, 'thriller': 0, 'comedy': 0, 'romance': 1,'adventure':0},
    'movie8': {'action': 0, 'thriller': 1, 'comedy': 0, 'romance': 0,'adventure':0},
    'movie9': {'action': 1, 'thriller': 0, 'comedy': 0, 'romance': 0,'adventure':0},
    'movie10': {'action': 0, 'thriller': 0, 'comedy': 1, 'romance': 1,'adventure':0}
}

def content_based_recommender(dataset, user_preferences, num_recommendations=5):
    # Compute weighted sums
    scores = {}
    for movie in dataset:
        features = dataset[movie]
        weighted_sum = sum(features[genre] for genre in user_preferences)
        scores[movie] = weighted_sum

    # Sort the scores in descending order
    sorted_scores = sorted(scores.items(), key=lambda x: x[1], reverse=True)

    # Generate recommendations
    recommendations = []
    for movie, score in sorted_scores:
        if movie not in user_preferences:
            recommendations.append(movie)
        if len(recommendations) == num_recommendations:
            break

    return recommendations

# Example usage
user_preferences = ['action', 'adventure']
recommendations = content_based_recommender(dataset, user_preferences)
print("Recommendations:", recommendations)
```



```
Recommendations: ['movie3', 'movie1', 'movie5', 'movie6', 'movie9']
```