WEKA TOOL

T.HEMA VARSHITHA
192011258

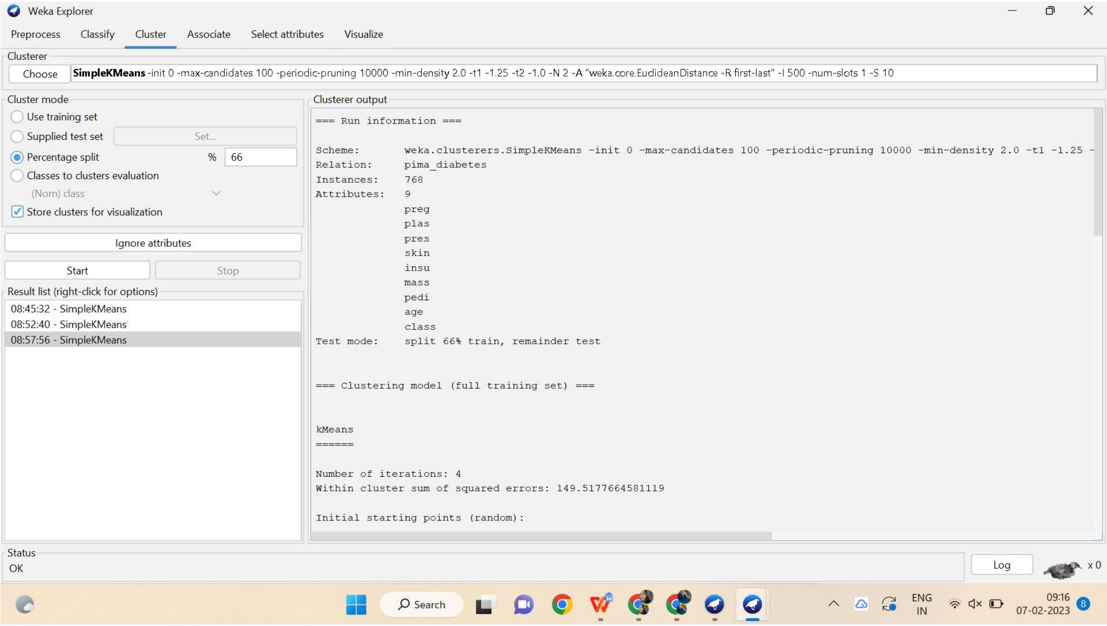
1. K MEANS ALGORITHM:

DATA SET: diabetes

ALGORITHM:

1. The k means clustering algorithm computes the centroids and repeats until optimal centroid is found.
2. Firstly,provide k number of clusters.
3. Choose k data points and assign to each clusters and divide data based on data points.
4. The cluster centroids will be constructed.
5. Iterate steps until ideal centroid is found.
6. The sum of squared distances between data points and clusters should be find.
7. Allocate each data point to cluster which is closest to centroid.
8. Construct the centroids for clusters by averaging all data points of clusters.

OUTPUT:



Clusterer: SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1 -S 10

Cluster mode

○ Use training set
○ Supplied test set
● Percentage split % 66
○ Classes to clusters evaluation
(Nom) class
☑ Store clusters for visualization

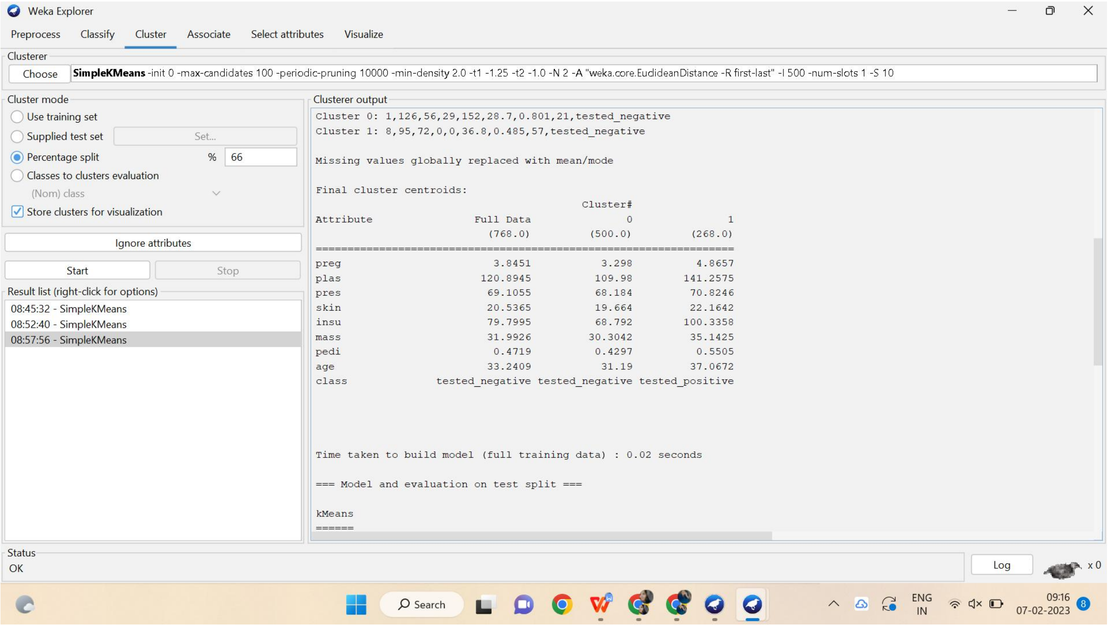
Result list (right-click for options)

08:45:32 - SimpleKMeans
08:52:40 - SimpleKMeans
08:57:56 - SimpleKMeans

Clusterer output

```
=== Run information ===

Scheme:       weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -
Relation:     pima_diabetes
Instances:    768
Attributes:   9
              preg
              plas
              pres
              skin
              insu
              mass
              pedi
              age
              class
Test mode:    split 66% train, remainder test


=== Clustering model (full training set) ===


kMeans
======

Number of iterations: 4
Within cluster sum of squared errors: 149.5177664581119

Initial starting points (random):
```

Status
OK



Clusterer
SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1 -S 10

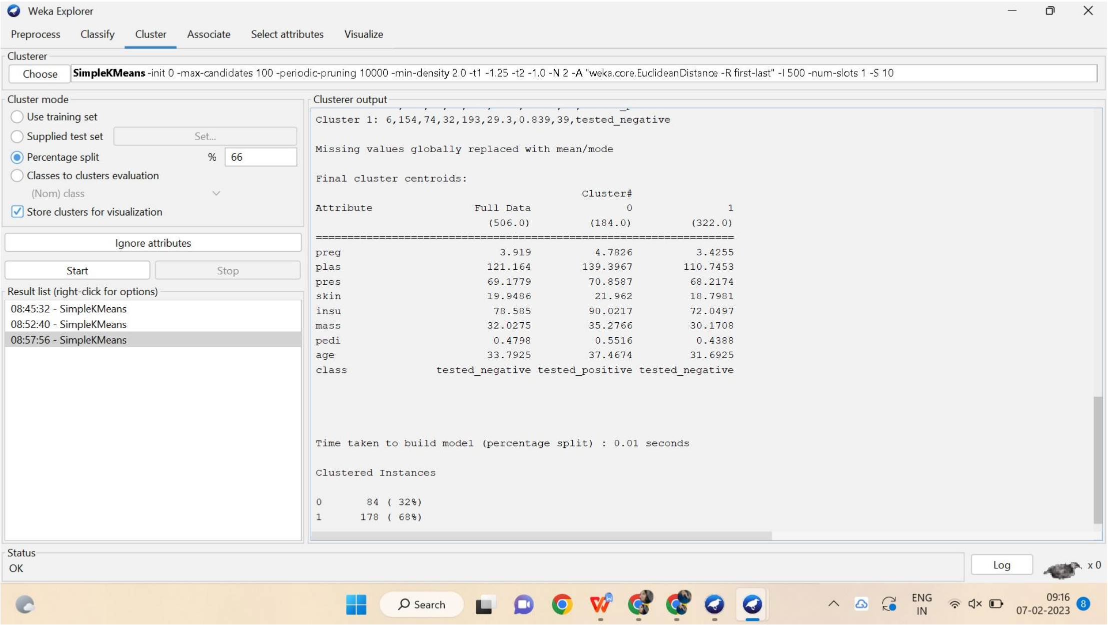
Cluster mode
○ Use training set
○ Supplied test set
● Percentage split % 66
○ Classes to clusters evaluation
(Nom) class
☑ Store clusters for visualization

Result list (right-click for options)
08:45:32 - SimpleKMeans
08:52:40 - SimpleKMeans
08:57:56 - SimpleKMeans

Clusterer output

```
Cluster 0: 1,126,56,29,152,28.7,0.801,21,tested_negative
Cluster 1: 8,95,72,0,0,36.8,0.485,57,tested_negative

Missing values globally replaced with mean/mode

Final cluster centroids:
                                        Cluster#
Attribute                  Full Data           0                1
                             (768.0)      (500.0)         (268.0)
=================================================================
preg                          3.8451       3.298           4.8657
plas                        120.8945      109.98         141.2575
pres                         69.1055      68.184          70.8246
skin                         20.5365      19.664          22.1642
insu                         79.7995      68.792         100.3358
mass                         31.9926     30.3042          35.1425
pedi                          0.4719      0.4297           0.5505
age                          33.2409       31.19          37.0672
class                 tested_negative tested_negative tested_positive




Time taken to build model (full training data) : 0.02 seconds

=== Model and evaluation on test split ===

kMeans
======
```

Status
OK



Clusterer
SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1 -S 10

Cluster mode
○ Use training set
○ Supplied test set
● Percentage split % 66
○ Classes to clusters evaluation
(Nom) class
☑ Store clusters for visualization

Result list (right-click for options)
08:45:32 - SimpleKMeans
08:52:40 - SimpleKMeans
08:57:56 - SimpleKMeans

Clusterer output

```
Cluster 1: 6,154,74,32,193,29.3,0.839,39,tested_negative

Missing values globally replaced with mean/mode

Final cluster centroids:
                                        Cluster#
Attribute                  Full Data           0                1
                             (506.0)      (184.0)         (322.0)
=================================================================
preg                           3.919      4.7826           3.4255
plas                         121.164    139.3967         110.7453
pres                         69.1779     70.8587          68.2174
skin                         19.9486      21.962          18.7981
insu                          78.585     90.0217          72.0497
mass                         32.0275     35.2766          30.1708
pedi                          0.4798      0.5516           0.4388
age                          33.7925     37.4674          31.6925
class                 tested_negative tested_positive tested_negative




Time taken to build model (percentage split) : 0.01 seconds

Clustered Instances

0       84 ( 32%)
1      178 ( 68%)
```

Status
OK

2. DECISION TREE:

DATA SET:diabetes

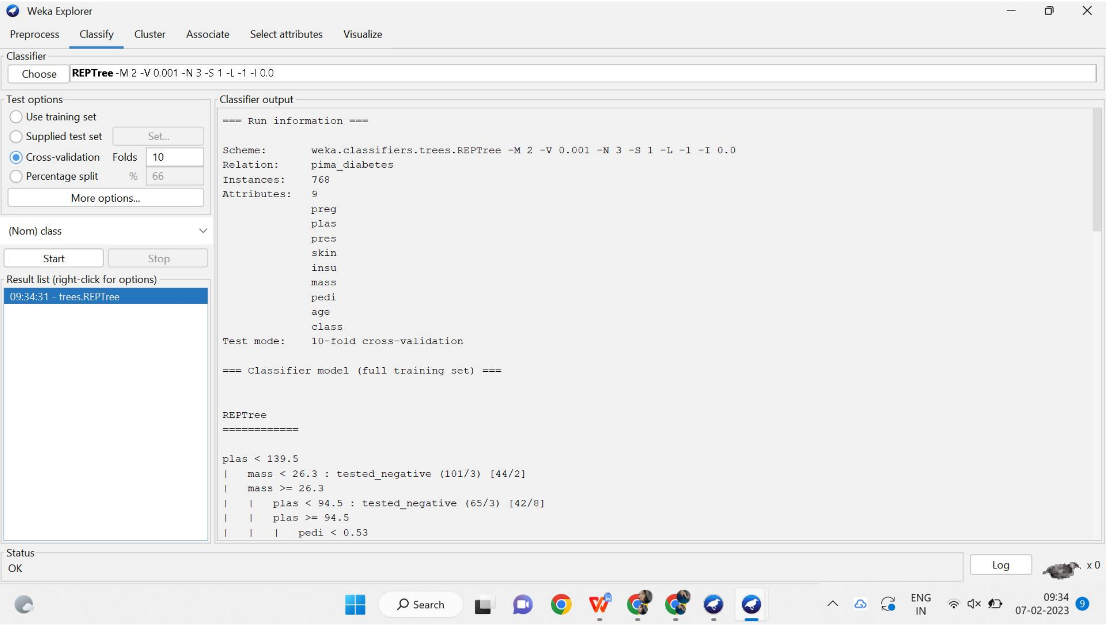
ALGORITHM:

1. determine the root node.
2. Calculate the entropy of classes.
3. Calculate the entropy of the split of the attribute.
4. Calculate the information gain.
5. Perform split.
6. Perform further split.
7. Compute decision tree.

Entropy= -Σ pi log2 pi

Information gain=entropy of parent node-sum of weights of entropy of child node.

Output:



Classifier

REPTree -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0

Test options

○ Use training set
○ Supplied test set
● Cross-validation Folds 10
○ Percentage split % 66

(Nom) class

Result list (right-click for options)

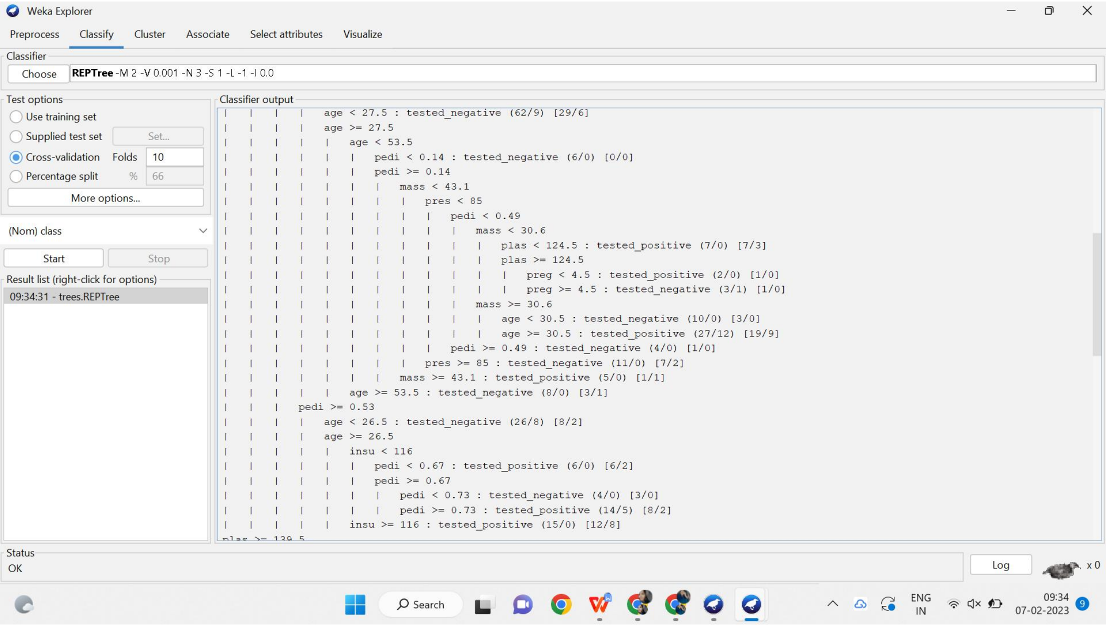
09:34:31 - trees.REPTree

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.trees.REPTree -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0
Relation:     pima_diabetes
Instances:    768
Attributes:   9
              preg
              plas
              pres
              skin
              insu
              mass
              pedi
              age
              class
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===


REPTree
============

plas < 139.5
|   mass < 26.3 : tested_negative (101/3) [44/2]
|   mass >= 26.3
|   |   plas < 94.5 : tested_negative (65/3) [42/8]
|   |   plas >= 94.5
|   |   |   pedi < 0.53
```

Status

OK



Classifier

REPTree -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0

Test options

○ Use training set
○ Supplied test set
● Cross-validation Folds 10
○ Percentage split % 66

(Nom) class

Result list (right-click for options)

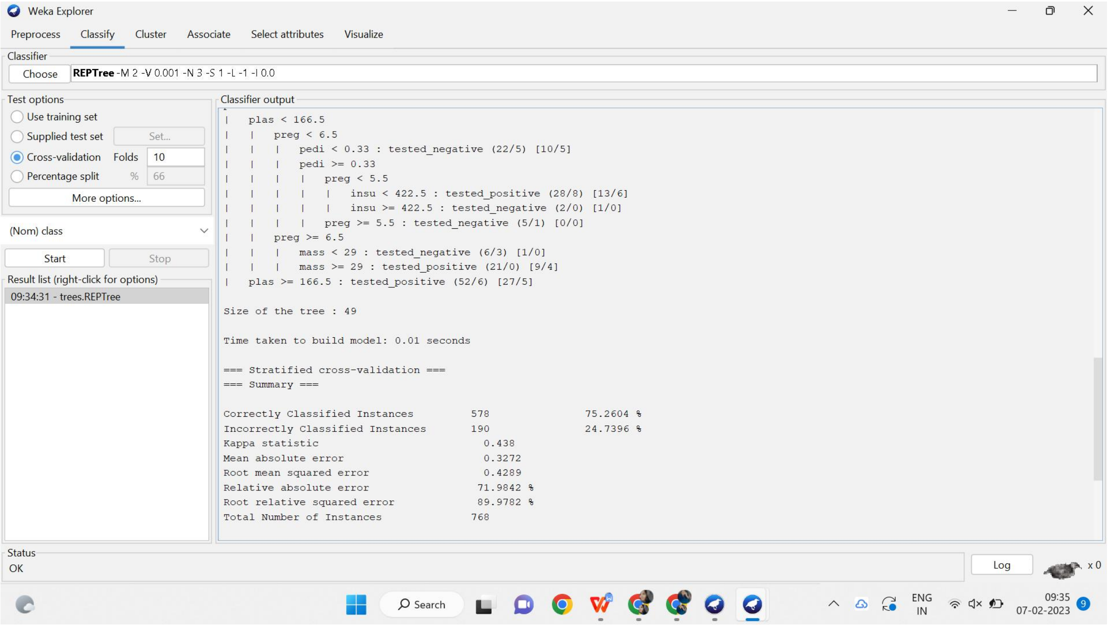
09:34:31 - trees.REPTree

Classifier output

```
|   |   |   |   age < 27.5 : tested_negative (62/9) [29/6]
|   |   |   |   age >= 27.5
|   |   |   |   |   age < 53.5
|   |   |   |   |   |   pedi < 0.14 : tested_negative (6/0) [0/0]
|   |   |   |   |   |   pedi >= 0.14
|   |   |   |   |   |   |   mass < 43.1
|   |   |   |   |   |   |   |   pres < 85
|   |   |   |   |   |   |   |   |   pedi < 0.49
|   |   |   |   |   |   |   |   |   |   mass < 30.6
|   |   |   |   |   |   |   |   |   |   |   plas < 124.5 : tested_positive (7/0) [7/3]
|   |   |   |   |   |   |   |   |   |   |   plas >= 124.5
|   |   |   |   |   |   |   |   |   |   |   |   preg < 4.5 : tested_positive (2/0) [1/0]
|   |   |   |   |   |   |   |   |   |   |   |   preg >= 4.5 : tested_negative (3/1) [1/0]
|   |   |   |   |   |   |   |   |   |   mass >= 30.6
|   |   |   |   |   |   |   |   |   |   |   age < 30.5 : tested_negative (10/0) [3/0]
|   |   |   |   |   |   |   |   |   |   |   age >= 30.5 : tested_positive (27/12) [19/9]
|   |   |   |   |   |   |   |   |   pedi >= 0.49 : tested_negative (4/0) [1/0]
|   |   |   |   |   |   |   |   pres >= 85 : tested_negative (11/0) [7/2]
|   |   |   |   |   |   |   mass >= 43.1 : tested_positive (5/0) [1/1]
|   |   |   |   |   age >= 53.5 : tested_negative (8/0) [3/1]
|   |   |   pedi >= 0.53
|   |   |   |   age < 26.5 : tested_negative (26/8) [8/2]
|   |   |   |   age >= 26.5
|   |   |   |   |   insu < 116
|   |   |   |   |   |   pedi < 0.67 : tested_positive (6/0) [6/2]
|   |   |   |   |   |   pedi >= 0.67
|   |   |   |   |   |   |   pedi < 0.73 : tested_negative (4/0) [3/0]
|   |   |   |   |   |   |   pedi >= 0.73 : tested_positive (14/5) [8/2]
|   |   |   |   |   insu >= 116 : tested_positive (15/0) [12/8]
plas >= 139.5
```

Status

OK



Classifier: REPTree -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0

Test options: ○ Use training set; ○ Supplied test set; ● Cross-validation Folds 10; ○ Percentage split % 66

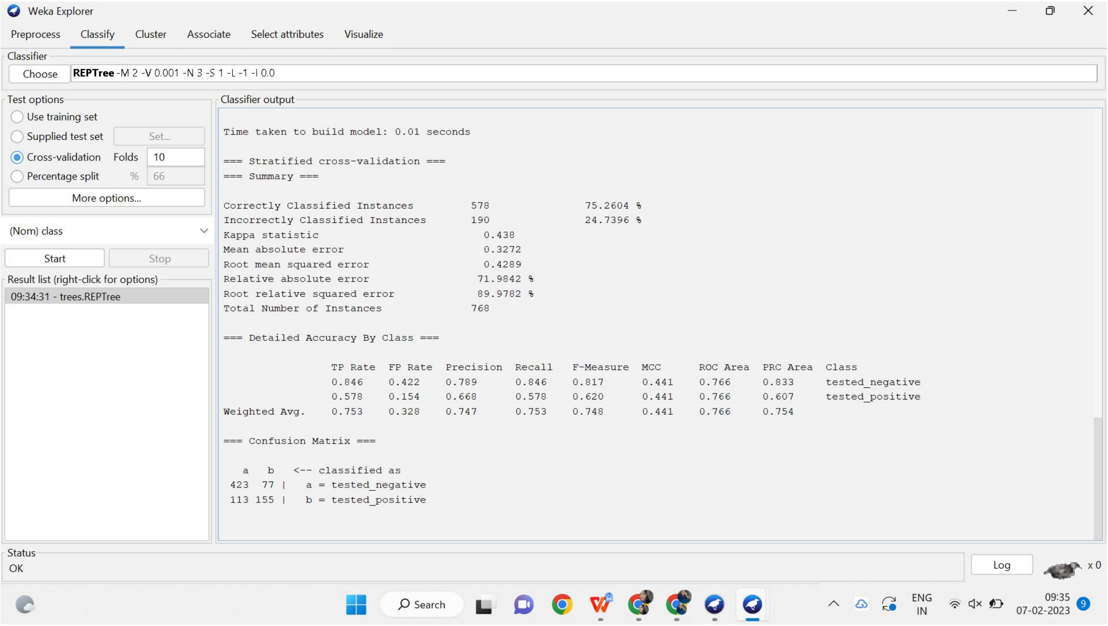
(Nom) class

Result list (right-click for options): 09:34:31 - trees.REPTree

Classifier output

```
|   plas < 166.5
|   |   preg < 6.5
|   |   |   pedi < 0.33 : tested_negative (22/5) [10/5]
|   |   |   pedi >= 0.33
|   |   |   |   preg < 5.5
|   |   |   |   |   insu < 422.5 : tested_positive (28/8) [13/6]
|   |   |   |   |   insu >= 422.5 : tested_negative (2/0) [1/0]
|   |   |   |   preg >= 5.5 : tested_negative (5/1) [0/0]
|   |   preg >= 6.5
|   |   |   mass < 29 : tested_negative (6/3) [1/0]
|   |   |   mass >= 29 : tested_positive (21/0) [9/4]
|   plas >= 166.5 : tested_positive (52/6) [27/5]

Size of the tree : 49

Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         578               75.2604 %
Incorrectly Classified Instances       190               24.7396 %
Kappa statistic                          0.438
Mean absolute error                      0.3272
Root mean squared error                  0.4289
Relative absolute error                 71.9842 %
Root relative squared error             89.9782 %
Total Number of Instances              768
```

Status
OK



Classifier: REPTree -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0

Test options: ○ Use training set; ○ Supplied test set; ● Cross-validation Folds 10; ○ Percentage split % 66

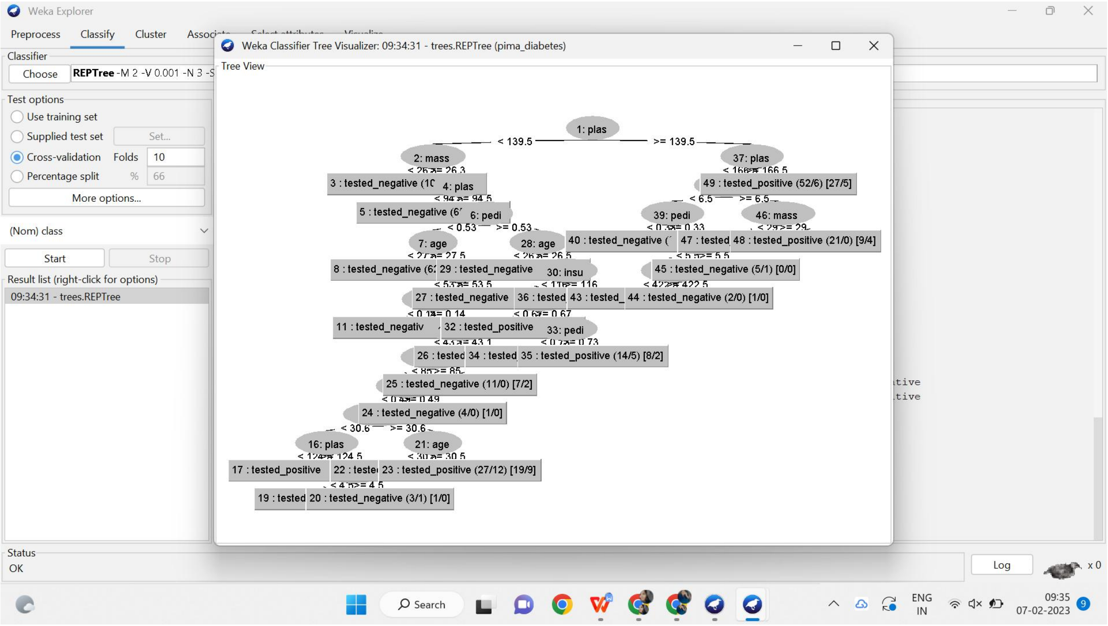
(Nom) class

Result list (right-click for options): 09:34:31 - trees.REPTree

Classifier output

```
Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         578               75.2604 %
Incorrectly Classified Instances       190               24.7396 %
Kappa statistic                          0.438
Mean absolute error                      0.3272
Root mean squared error                  0.4289
Relative absolute error                 71.9842 %
Root relative squared error             89.9782 %
Total Number of Instances              768

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.846    0.422    0.789      0.846    0.817      0.441    0.766     0.833     tested_negative
                 0.578    0.154    0.668      0.578    0.620      0.441    0.766     0.607     tested_positive
Weighted Avg.    0.753    0.328    0.747      0.753    0.748      0.441    0.766     0.754

=== Confusion Matrix ===

   a   b   <-- classified as
 423  77 |   a = tested_negative
 113 155 |   b = tested_positive
```

Status
OK



3. BAYESIAN CLASSIFICATION:

DATASET:diabetes

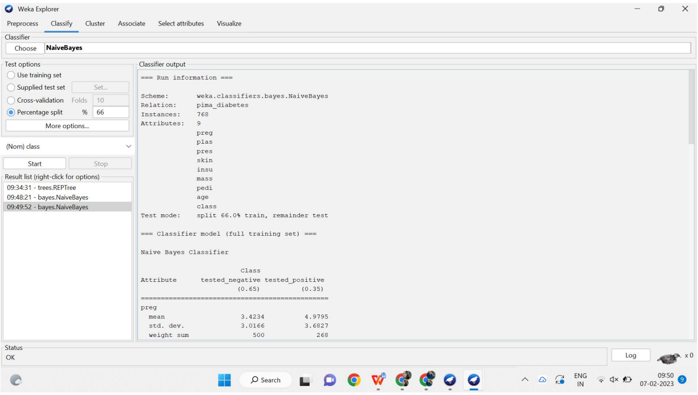
ALGORITHM:

1. convert given dataset into frequency table.
2. Construct livelihood tables by calculating the probabilities.
3. Use the bayes formula for calculating probabilities.

$$P(A|B) = [P(B|A)\ P(A)]/\ P(B), \text{ where } P(B) \neq 0$$

4. now calculate the probability for all possible choices.
5. Then compare all the outputs.
6. Determine the probability which is more efficient by checking outputs.
7. Finally,compute the probability using bayesian classification.

OUTPUT:



Classifier: NaiveBayes

Test options
○ Use training set
○ Supplied test set
○ Cross-validation Folds 10
● Percentage split % 66

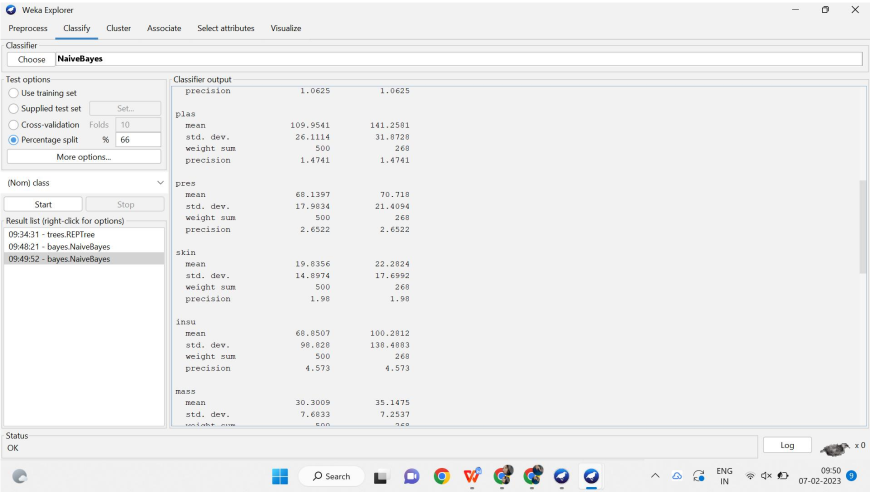
(Nom) class

Result list (right-click for options)
09:34:31 - trees.REPTree
09:48:21 - bayes.NaiveBayes
09:49:52 - bayes.NaiveBayes

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     pima_diabetes
Instances:    768
Attributes:   9
              preg
              plas
              pres
              skin
              insu
              mass
              pedi
              age
              class
Test mode:    split 66.0% train, remainder test

=== Classifier model (full training set) ===

Naive Bayes Classifier

                          Class
Attribute      tested_negative tested_positive
                        (0.65)          (0.35)
===============================================
preg
  mean                  3.4234          4.9795
  std. dev.             3.0166          3.6827
  weight sum               500             268
```

Status
OK



Classifier: NaiveBayes

Test options:
- ○ Use training set
- ○ Supplied test set
- ○ Cross-validation Folds 10
- ● Percentage split % 66

(Nom) class

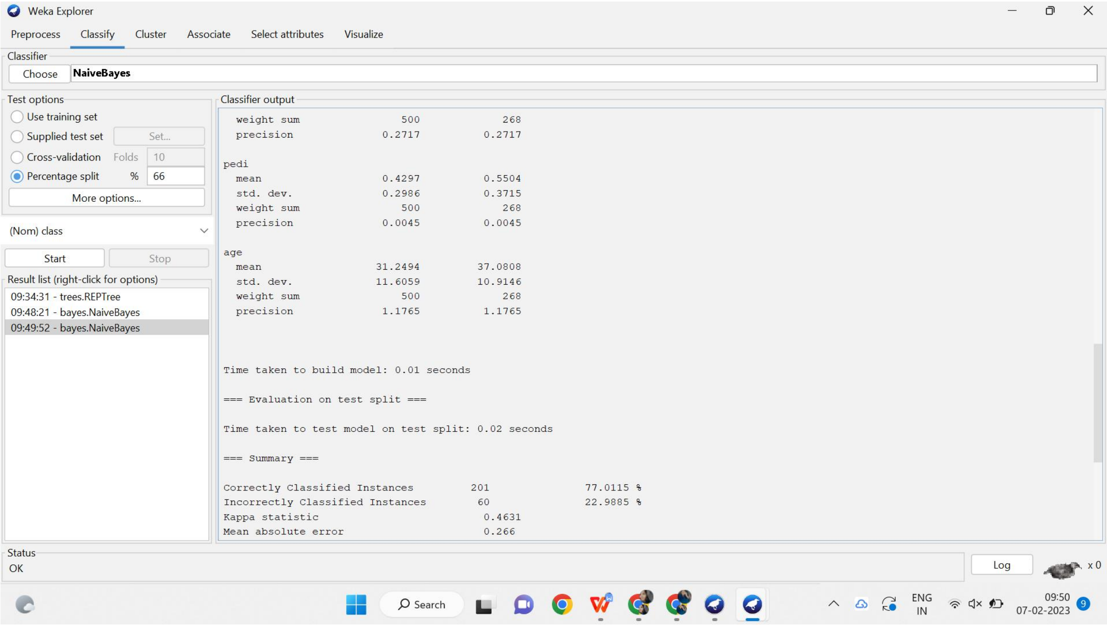
Result list (right-click for options)
- 09:34:31 - trees.REPTree
- 09:48:21 - bayes.NaiveBayes
- 09:49:52 - bayes.NaiveBayes

Classifier output

```
  precision           1.0625         1.0625

plas
  mean              109.9541       141.2581
  std. dev.          26.1114        31.8728
  weight sum             500            268
  precision           1.4741         1.4741

pres
  mean               68.1397         70.718
  std. dev.          17.9834        21.4094
  weight sum             500            268
  precision           2.6522         2.6522

skin
  mean               19.8356        22.2824
  std. dev.          14.8974        17.6992
  weight sum             500            268
  precision             1.98           1.98

insu
  mean               68.8507       100.2812
  std. dev.           98.828       138.4883
  weight sum             500            268
  precision            4.573          4.573

mass
  mean               30.3009        35.1475
  std. dev.           7.6833         7.2537
  weight sum             500            268
```

Status
OK

ENG IN 09:50 07-02-2023



Classifier: NaiveBayes

Test options:
- ○ Use training set
- ○ Supplied test set
- ○ Cross-validation Folds 10
- ● Percentage split % 66

(Nom) class

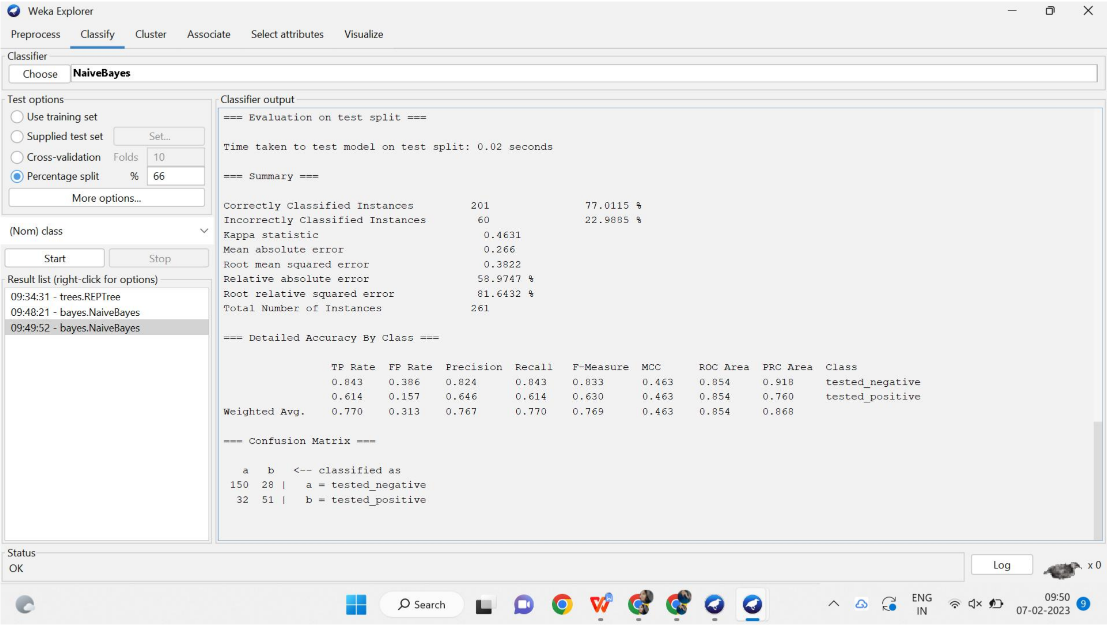
Result list (right-click for options)
- 09:34:31 - trees.REPTree
- 09:48:21 - bayes.NaiveBayes
- 09:49:52 - bayes.NaiveBayes

Classifier output

```
  weight sum             500            268
  precision           0.2717         0.2717

pedi
  mean                0.4297         0.5504
  std. dev.           0.2986         0.3715
  weight sum             500            268
  precision           0.0045         0.0045

age
  mean               31.2494        37.0808
  std. dev.          11.6059        10.9146
  weight sum             500            268
  precision           1.1765         1.1765



Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.02 seconds

=== Summary ===

Correctly Classified Instances         201               77.0115 %
Incorrectly Classified Instances        60               22.9885 %
Kappa statistic                          0.4631
Mean absolute error                      0.266
```

Status
OK

ENG IN 09:50 07-02-2023



Classifier: NaiveBayes

Test options:
- ○ Use training set
- ○ Supplied test set
- ○ Cross-validation Folds 10
- ● Percentage split % 66

(Nom) class

Result list (right-click for options)
- 09:34:31 - trees.REPTree
- 09:48:21 - bayes.NaiveBayes
- 09:49:52 - bayes.NaiveBayes

Classifier output

```
=== Evaluation on test split ===

Time taken to test model on test split: 0.02 seconds

=== Summary ===

Correctly Classified Instances         201               77.0115 %
Incorrectly Classified Instances        60               22.9885 %
Kappa statistic                          0.4631
Mean absolute error                      0.266
Root mean squared error                  0.3822
Relative absolute error                 58.9747 %
Root relative squared error             81.6432 %
Total Number of Instances              261

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.843    0.386    0.824      0.843    0.833      0.463    0.854     0.918     tested_negative
                 0.614    0.157    0.646      0.614    0.630      0.463    0.854     0.760     tested_positive
Weighted Avg.    0.770    0.313    0.767      0.770    0.769      0.463    0.854     0.868

=== Confusion Matrix ===

   a   b   <-- classified as
 150  28 |   a = tested_negative
  32  51 |   b = tested_positive
```

Status
OK

ENG IN 09:50 07-02-2023

4. APRIORI ALGORITHM:

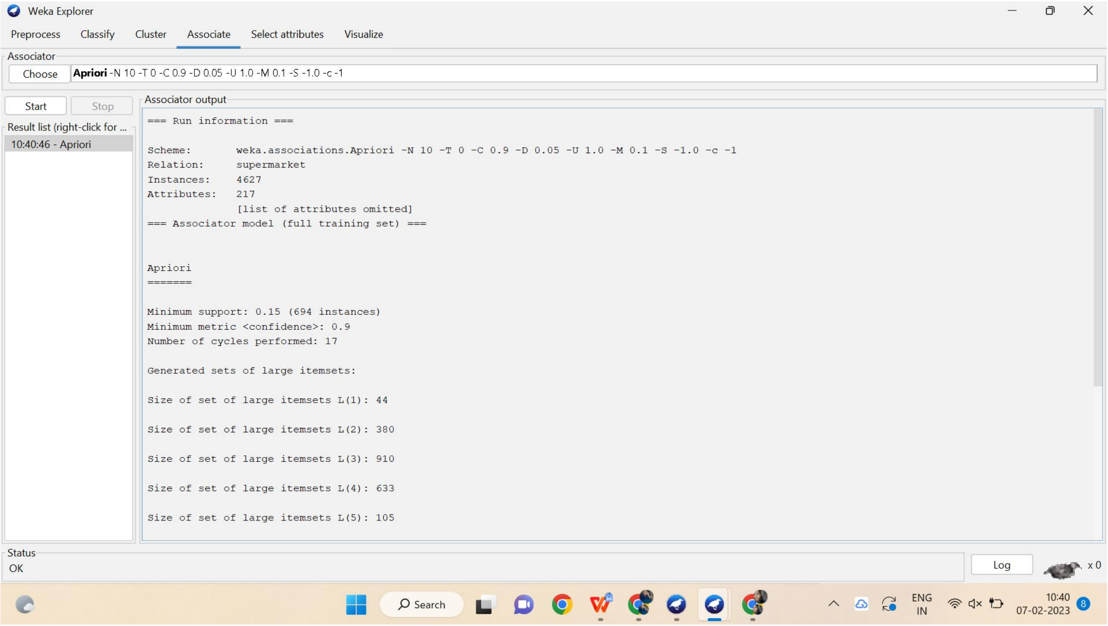
DATASET: supermarket

ALGORITHM:

1. firstly,convert the given transactional database into an frequency table.
2. Assign any minimum support to the frequency table,in which contains item sets and suppor count.
3. The item sets and support count is combinely called as candidate set.
4. Now,check the support count with the minimum support.
5. Remove the support count which is less than minimum support and write the remaining item sets in descending order.
6. Again checking by combining two itemsets.
7. iterate the steps until the support count should be equal to minimum support.

   Confidence=support(A⋂B)/support(A)
8. calculate the confidence and convert it into percentage.
9. Finally,check which is more efficient.

OUTPUT:

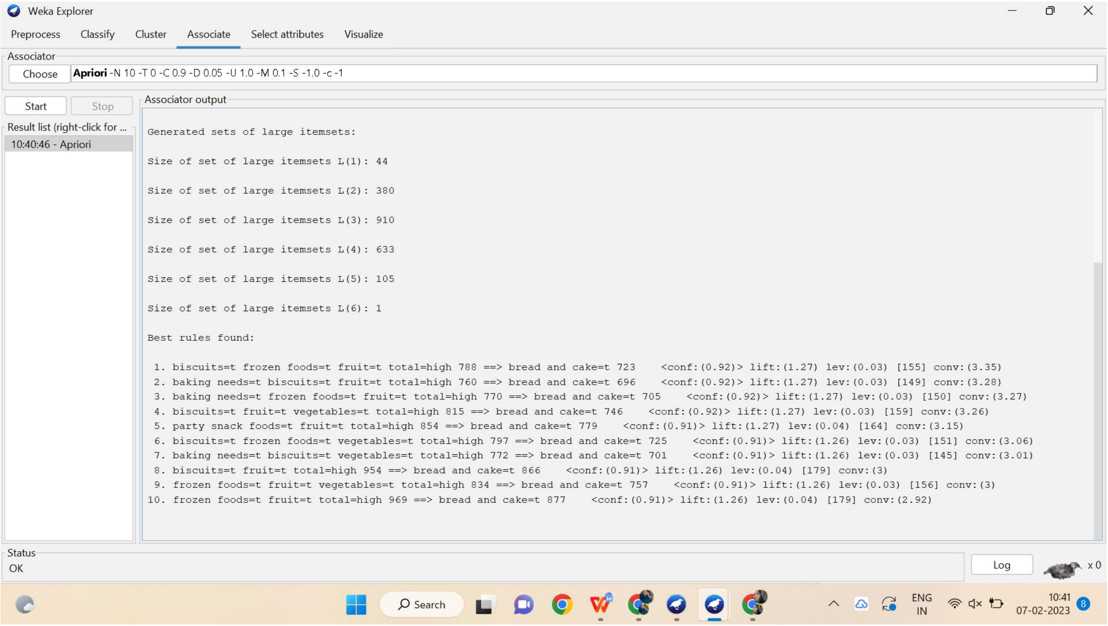


Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

10:40:46 - 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
```

OK



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

```
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)
```

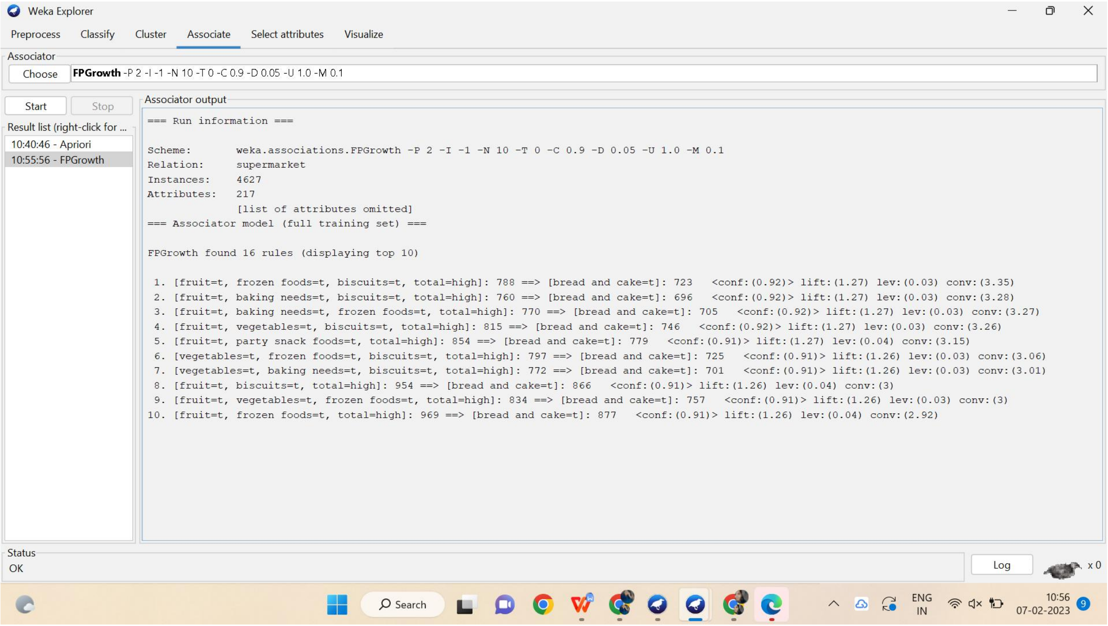
5. FP GROWTH ALGORITHM:

DATASET: supermarket

ALGORITHM:

1. firstly,convert the given transactional database into an frequency table.
2. Assign any minimum support to the frequency table,in which contains itemsets and support count.
3. The item sets and support count is combinely called as candidate set.
4. Now,check the support count with the minimum support.
5. Remove the support count which is less than minimum support and write remaining items in descending order.
6. Find the ordered item set using frequency table.
7. Construct the FP gowth using the ordered item set.
8. Then compute the conditionally pattern using FP grpowth.
9. Again find the conditionally frequency pattern.
10. Finally compute the FP gtowth algorithm.

OUTPUT:



FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1

10:40:46 - Apriori
10:55:56 - FPGrowth

```
=== 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)
```

OK