DMWA Lab – 5

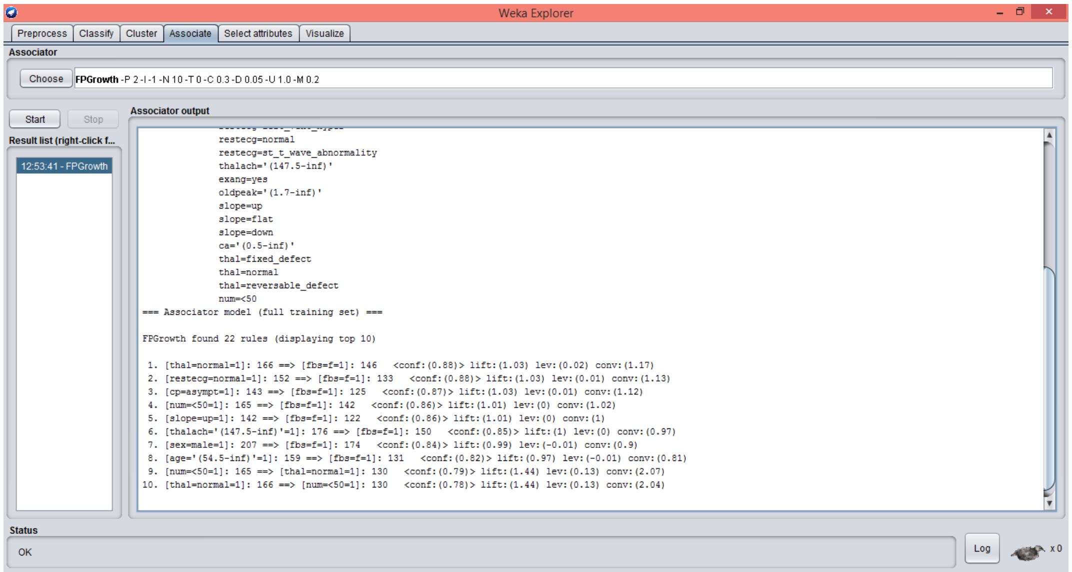
Tanya Pandhi

18104064

B12

Q1

FP_growth Rule in weka



FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.3 -D 0.05 -U 1.0 -M 0.2

```
               restecg=normal
               restecg=st_t_wave_abnormality
               thalach='(147.5-inf)'
               exang=yes
               oldpeak='(1.7-inf)'
               slope=up
               slope=flat
               slope=down
               ca='(0.5-inf)'
               thal=fixed_defect
               thal=normal
               thal=reversable_defect
               num=<50
=== Associator model (full training set) ===

FPGrowth found 22 rules (displaying top 10)

 1. [thal=normal=1]: 166 ==> [fbs=f=1]: 146   <conf:(0.88)> lift:(1.03) lev:(0.02) conv:(1.17)
 2. [restecg=normal=1]: 152 ==> [fbs=f=1]: 133   <conf:(0.88)> lift:(1.03) lev:(0.01) conv:(1.13)
 3. [cp=asympt=1]: 143 ==> [fbs=f=1]: 125   <conf:(0.87)> lift:(1.03) lev:(0.01) conv:(1.12)
 4. [num=<50=1]: 165 ==> [fbs=f=1]: 142   <conf:(0.86)> lift:(1.01) lev:(0) conv:(1.02)
 5. [slope=up=1]: 142 ==> [fbs=f=1]: 122   <conf:(0.86)> lift:(1.01) lev:(0) conv:(1)
 6. [thalach='(147.5-inf)'=1]: 176 ==> [fbs=f=1]: 150   <conf:(0.85)> lift:(1) lev:(0) conv:(0.97)
 7. [sex=male=1]: 207 ==> [fbs=f=1]: 174   <conf:(0.84)> lift:(0.99) lev:(-0.01) conv:(0.9)
 8. [age='(54.5-inf)'=1]: 159 ==> [fbs=f=1]: 131   <conf:(0.82)> lift:(0.97) lev:(-0.01) conv:(0.81)
 9. [num=<50=1]: 165 ==> [thal=normal=1]: 130   <conf:(0.79)> lift:(1.44) lev:(0.13) conv:(2.07)
10. [thal=normal=1]: 166 ==> [num=<50=1]: 130   <conf:(0.78)> lift:(1.44) lev:(0.13) conv:(2.04)
```

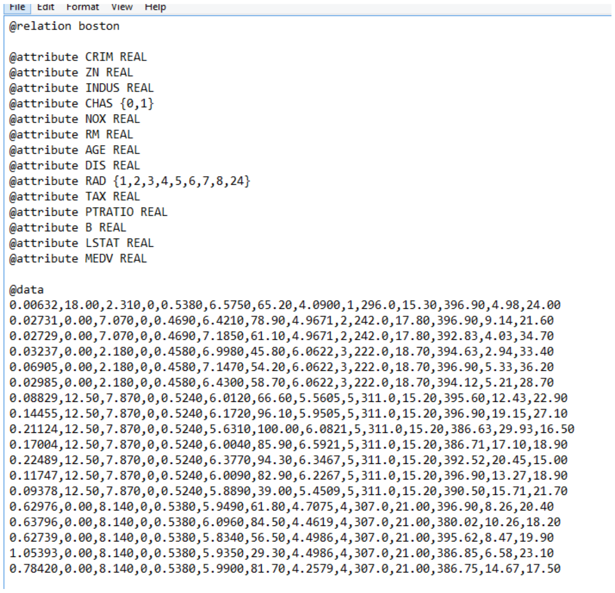
Q2

```
import pandas as pd
import numpy as np

import pyfpgrowth

df= pd.read_csv(“ transaction_data.csv”)

patterns = pyfpgrowth. find_frequent_patterns(transactions, 10)

rules = pyfpgrowth. generate_association_rules(patterns,0.8)


def support_count(rhs):

count=0
```

```
rhs= set(rhs)
for j in df[‘items’]:
j=set(j)
if(rhs.issubset(j)):
count=count+1
return count

rhs_support = []
for I in rules_df[‘Consequent’]:
a=support_count(i)
rhs_support.append(a/len(df))
rules_df[‘RHS_support’] = rhs_support
rules_df[‘lift’] = rules_df[‘Confidence’]/rules_df[‘RHS_support’]
rules_df[‘Conviction’] = (1-rules_df[‘RHS_support’])/(1-rules_df[‘Confidence’])
```

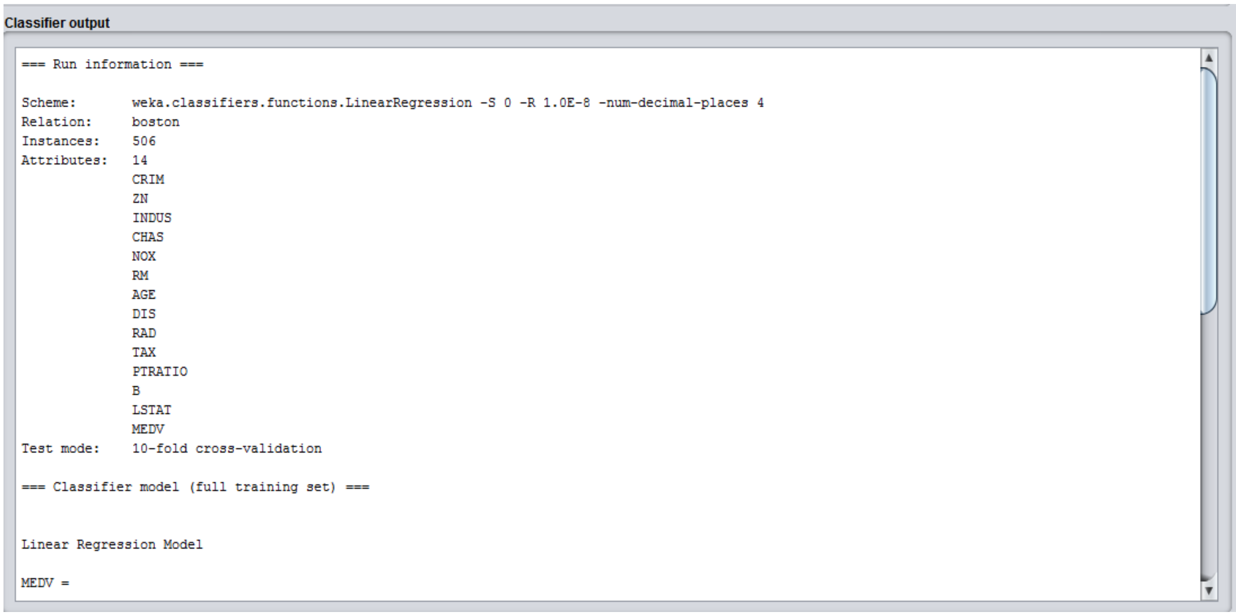
Q3

Bostonhousing.arff

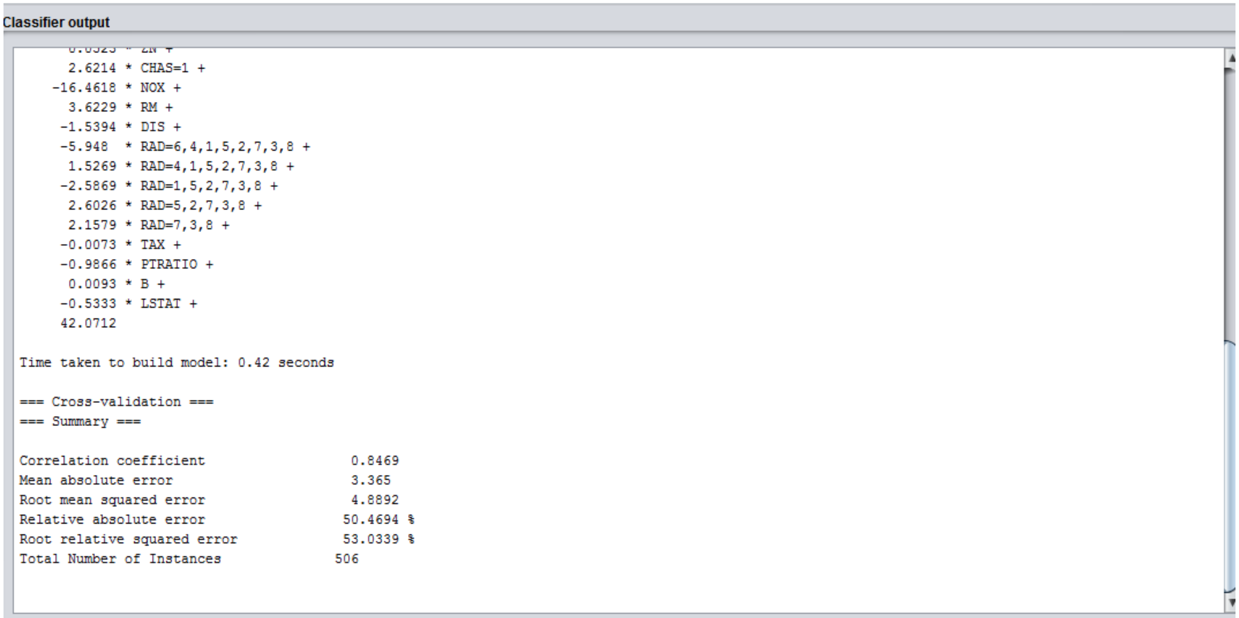


```
@relation boston

@attribute CRIM REAL
@attribute ZN REAL
@attribute INDUS REAL
@attribute CHAS {0,1}
@attribute NOX REAL
@attribute RM REAL
@attribute AGE REAL
@attribute DIS REAL
@attribute RAD {1,2,3,4,5,6,7,8,24}
@attribute TAX REAL
@attribute PTRATIO REAL
@attribute B REAL
@attribute LSTAT REAL
@attribute MEDV REAL

@data
0.00632,18.00,2.310,0,0.5380,6.5750,65.20,4.0900,1,296.0,15.30,396.90,4.98,24.00
0.02731,0.00,7.070,0,0.4690,6.4210,78.90,4.9671,2,242.0,17.80,396.90,9.14,21.60
0.02729,0.00,7.070,0,0.4690,7.1850,61.10,4.9671,2,242.0,17.80,392.83,4.03,34.70
0.03237,0.00,2.180,0,0.4580,6.9980,45.80,6.0622,3,222.0,18.70,394.63,2.94,33.40
0.06905,0.00,2.180,0,0.4580,7.1470,54.20,6.0622,3,222.0,18.70,396.90,5.33,36.20
0.02985,0.00,2.180,0,0.4580,6.4300,58.70,6.0622,3,222.0,18.70,394.12,5.21,28.70
0.08829,12.50,7.870,0,0.5240,6.0120,66.60,5.5605,5,311.0,15.20,395.60,12.43,22.90
0.14455,12.50,7.870,0,0.5240,6.1720,96.10,5.9505,5,311.0,15.20,396.90,19.15,27.10
0.21124,12.50,7.870,0,0.5240,5.6310,100.00,6.0821,5,311.0,15.20,386.63,29.93,16.50
0.17004,12.50,7.870,0,0.5240,6.0040,85.90,6.5921,5,311.0,15.20,386.71,17.10,18.90
0.22489,12.50,7.870,0,0.5240,6.3770,94.30,6.3467,5,311.0,15.20,392.52,20.45,15.00
0.11747,12.50,7.870,0,0.5240,6.0090,82.90,6.2267,5,311.0,15.20,396.90,13.27,18.90
0.09378,12.50,7.870,0,0.5240,5.8890,39.00,5.4509,5,311.0,15.20,390.50,15.71,21.70
0.62976,0.00,8.140,0,0.5380,5.9490,61.80,4.7075,4,307.0,21.00,396.90,8.26,20.40
0.63796,0.00,8.140,0,0.5380,6.0960,84.50,4.4619,4,307.0,21.00,380.02,10.26,18.20
0.62739,0.00,8.140,0,0.5380,5.8340,56.50,4.4986,4,307.0,21.00,395.62,8.47,19.90
1.05393,0.00,8.140,0,0.5380,5.9350,29.30,4.4986,4,307.0,21.00,386.85,6.58,23.10
0.78420,0.00,8.140,0,0.5380,5.9900,81.70,4.2579,4,307.0,21.00,386.75,14.67,17.50
```

Weka Linear Regression



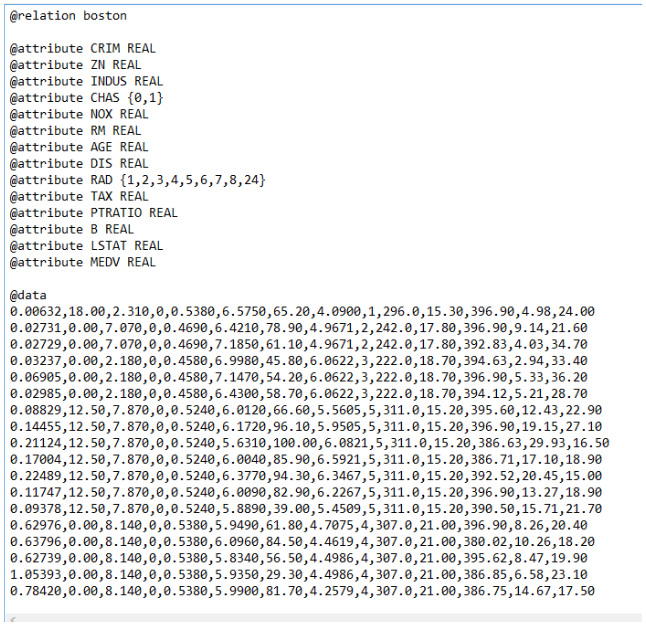
```
=== Run information ===

Scheme:       weka.classifiers.functions.LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4
Relation:     boston
Instances:    506
Attributes:   14
              CRIM
              ZN
              INDUS
              CHAS
              NOX
              RM
              AGE
              DIS
              RAD
              TAX
              PTRATIO
              B
              LSTAT
              MEDV
Test mode:    10-fold cross-validation

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


Linear Regression Model

MEDV =
```



```
      0.0523 * ZN +
      2.6214 * CHAS=1 +
    -16.4618 * NOX +
      3.6229 * RM +
     -1.5394 * DIS +
     -5.948  * RAD=6,4,1,5,2,7,3,8 +
      1.5269 * RAD=4,1,5,2,7,3,8 +
     -2.5869 * RAD=1,5,2,7,3,8 +
      2.6026 * RAD=5,2,7,3,8 +
      2.1579 * RAD=7,3,8 +
     -0.0073 * TAX +
     -0.9866 * PTRATIO +
      0.0093 * B +
     -0.5333 * LSTAT +
     42.0712

Time taken to build model: 0.42 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8469
Mean absolute error                      3.365
Root mean squared error                  4.8892
Relative absolute error                 50.4694 %
Root relative squared error             53.0339 %
Total Number of Instances              506
```

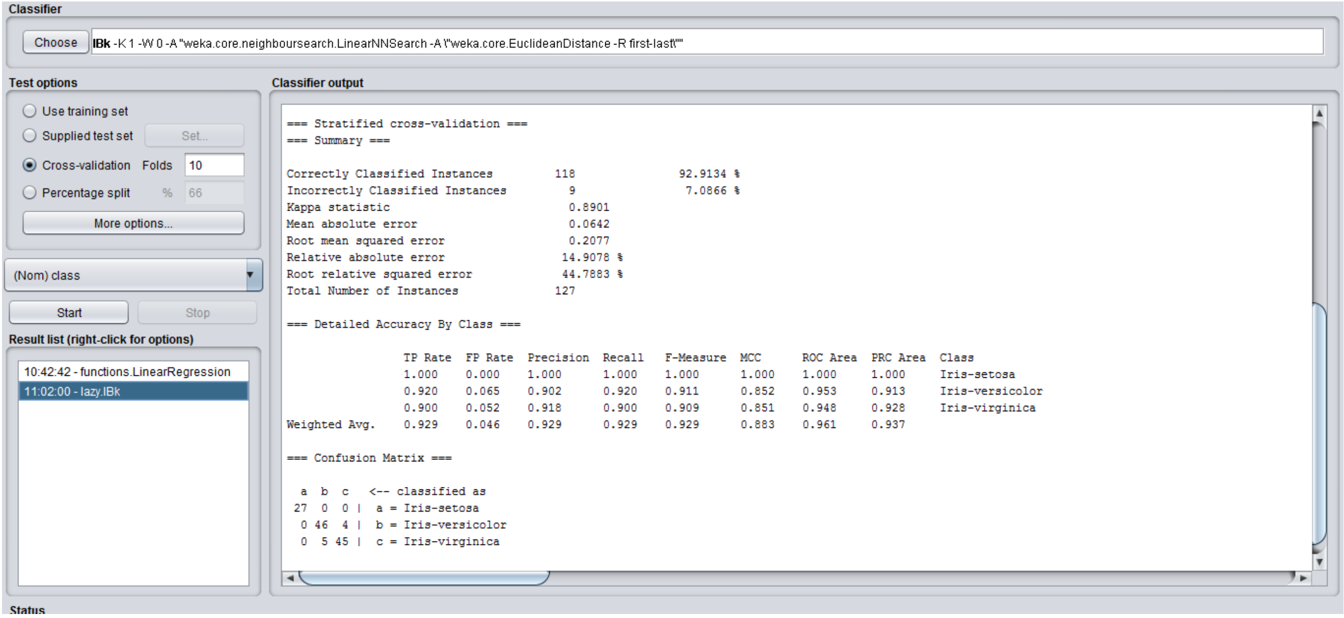
Q4

Iris.arff



```
@relation boston

@attribute CRIM REAL
@attribute ZN REAL
@attribute INDUS REAL
@attribute CHAS {0,1}
@attribute NOX REAL
@attribute RM REAL
@attribute AGE REAL
@attribute DIS REAL
@attribute RAD {1,2,3,4,5,6,7,8,24}
@attribute TAX REAL
@attribute PTRATIO REAL
@attribute B REAL
@attribute LSTAT REAL
@attribute MEDV REAL

@data
0.00632,18.00,2.310,0,0.5380,6.5750,65.20,4.0900,1,296.0,15.30,396.90,4.98,24.00
0.02731,0.00,7.070,0,0.4690,6.4210,78.90,4.9671,2,242.0,17.80,396.90,9.14,21.60
0.02729,0.00,7.070,0,0.4690,7.1850,61.10,4.9671,2,242.0,17.80,392.83,4.03,34.70
0.03237,0.00,2.180,0,0.4580,6.9980,45.80,6.0622,3,222.0,18.70,394.63,2.94,33.40
0.06905,0.00,2.180,0,0.4580,7.1470,54.20,6.0622,3,222.0,18.70,396.90,5.33,36.20
0.02985,0.00,2.180,0,0.4580,6.4300,58.70,6.0622,3,222.0,18.70,394.12,5.21,28.70
0.08829,12.50,7.870,0,0.5240,6.0120,66.60,5.5605,5,311.0,15.20,395.60,12.43,22.90
0.14455,12.50,7.870,0,0.5240,6.1720,96.10,5.9505,5,311.0,15.20,396.90,19.15,27.10
0.21124,12.50,7.870,0,0.5240,5.6310,100.00,6.0821,5,311.0,15.20,386.63,29.93,16.50
0.17004,12.50,7.870,0,0.5240,6.0040,85.90,6.5921,5,311.0,15.20,386.71,17.10,18.90
0.22489,12.50,7.870,0,0.5240,6.3770,94.30,6.3467,5,311.0,15.20,392.52,20.45,15.00
0.11747,12.50,7.870,0,0.5240,6.0090,82.90,6.2267,5,311.0,15.20,396.90,13.27,18.90
0.09378,12.50,7.870,0,0.5240,5.8890,39.00,5.4509,5,311.0,15.20,390.50,15.71,21.70
0.62976,0.00,8.140,0,0.5380,5.9490,61.80,4.7075,4,307.0,21.00,396.90,8.26,20.40
0.63796,0.00,8.140,0,0.5380,6.0960,84.50,4.4619,4,307.0,21.00,380.02,10.26,18.20
0.62739,0.00,8.140,0,0.5380,5.8340,56.50,4.4986,4,307.0,21.00,395.62,8.47,19.90
1.05393,0.00,8.140,0,0.5380,5.9350,29.30,4.4986,4,307.0,21.00,386.85,6.58,23.10
0.78420,0.00,8.140,0,0.5380,5.9900,81.70,4.2579,4,307.0,21.00,386.75,14.67,17.50
```

KNN classifier using Weka



Classifier

IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""

Test options

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

(Nom) class

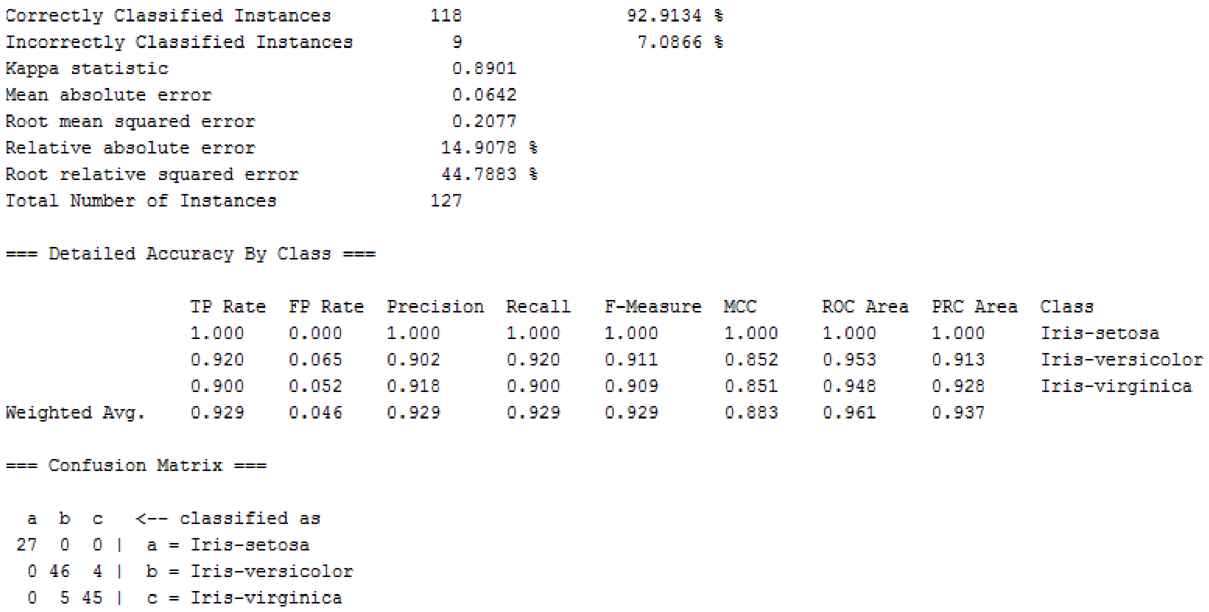
Result list (right-click for options)

10:42:42 - functions.LinearRegression
11:02:00 - lazy.IBk

Classifier output

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

Correctly Classified Instances         118               92.9134 %
Incorrectly Classified Instances         9                7.0866 %
Kappa statistic                          0.8901
Mean absolute error                      0.0642
Root mean squared error                  0.2077
Relative absolute error                 14.9078 %
Root relative squared error             44.7883 %
Total Number of Instances              127

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Iris-setosa
                 0.920    0.065    0.902      0.920    0.911      0.852    0.953     0.913     Iris-versicolor
                 0.900    0.052    0.918      0.900    0.909      0.851    0.948     0.928     Iris-virginica
Weighted Avg.    0.929    0.046    0.929      0.929    0.929      0.883    0.961     0.937

=== Confusion Matrix ===

  a  b  c   <-- classified as
 27  0  0 |  a = Iris-setosa
  0 46  4 |  b = Iris-versicolor
  0  5 45 |  c = Iris-virginica
```

Confusion Matrix of Iris.arff using KNN



```
Correctly Classified Instances         118               92.9134 %
Incorrectly Classified Instances         9                7.0866 %
Kappa statistic                          0.8901
Mean absolute error                      0.0642
Root mean squared error                  0.2077
Relative absolute error                 14.9078 %
Root relative squared error             44.7883 %
Total Number of Instances              127

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Iris-setosa
                 0.920    0.065    0.902      0.920    0.911      0.852    0.953     0.913     Iris-versicolor
                 0.900    0.052    0.918      0.900    0.909      0.851    0.948     0.928     Iris-virginica
Weighted Avg.    0.929    0.046    0.929      0.929    0.929      0.883    0.961     0.937

=== Confusion Matrix ===

  a  b  c   <-- classified as
 27  0  0 |  a = Iris-setosa
  0 46  4 |  b = Iris-versicolor
  0  5 45 |  c = Iris-virginica
```

Code Of KNN using python-:

```
from math import sqrt


def euclidean_distance(row1, row2):
distance = 0.0
for i in range(len(row1)-1):
distance += (row1[i] - row2[i])**2
return sqrt(distance)

def get_neighbors(train, test_row, num_neighbors):
distances = list()
for train_row in train:
dist = euclidean_distance(test_row, train_row)
distances.append((train_row, dist))
distances.sort(key=lambda tup: tup[1])
neighbors = list()
for i in range(num_neighbors):
neighbors.append(distances[i][0])
return neighbors


def predict_classification(train, test_row, num_neighbors):
neighbors = get_neighbors(train, test_row, num_neighbors)
output_values = [row[-1] for row in neighbors]
```

```
prediction = max(set(output_values), key=output_values.count)
return prediction

dataset = [….]
prediction = predict_classification(dataset, dataset[0], 3)
print('Expected %d, Got %d.' % (dataset[0][-1], prediction))
```