DATA MINING LAB ASSIGNMENT

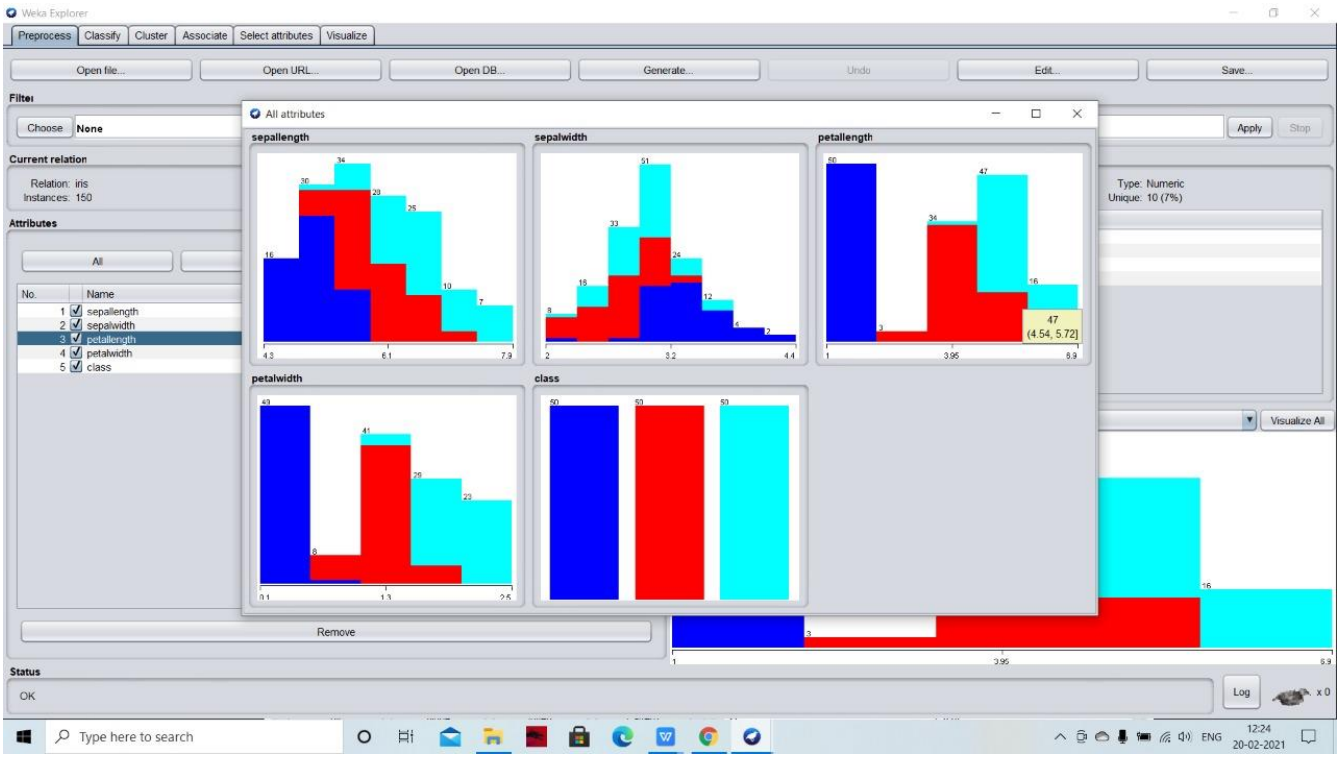
Name: P.Prasanna Kumar **Roll No:2451-17-733-096** **Section:CSE-2**

1.DATA PREPROCESSING

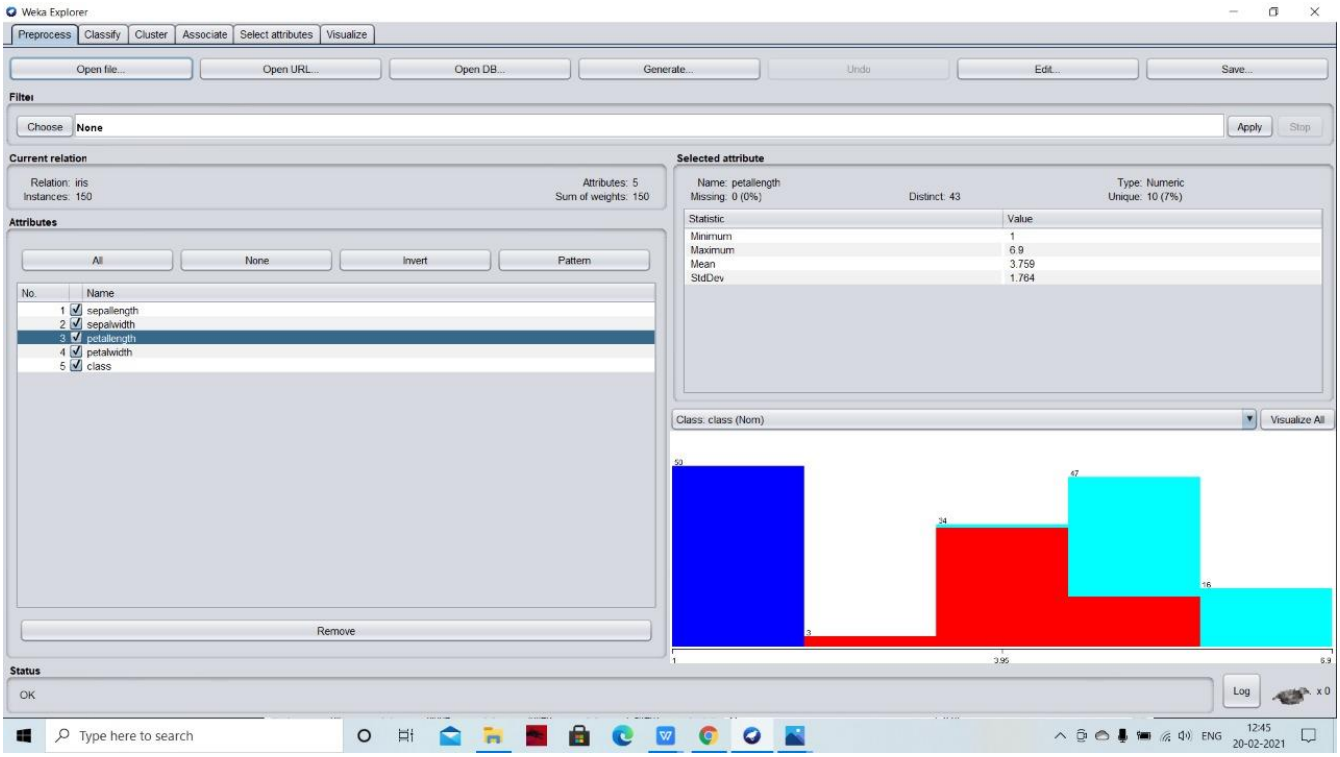
1.1Descriptive Stats and Visualization

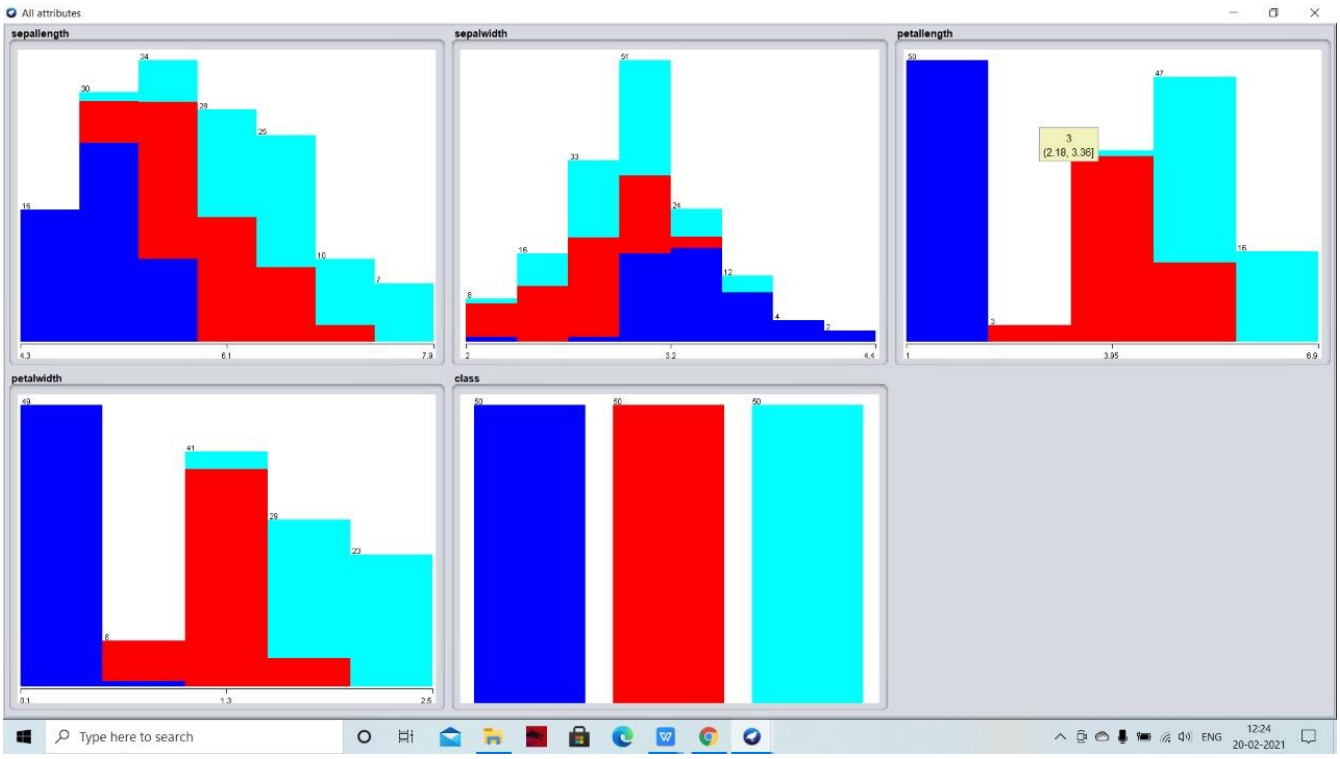
Weka allows you to review descriptive statistics calculated from your data. It also provides visualization tools.

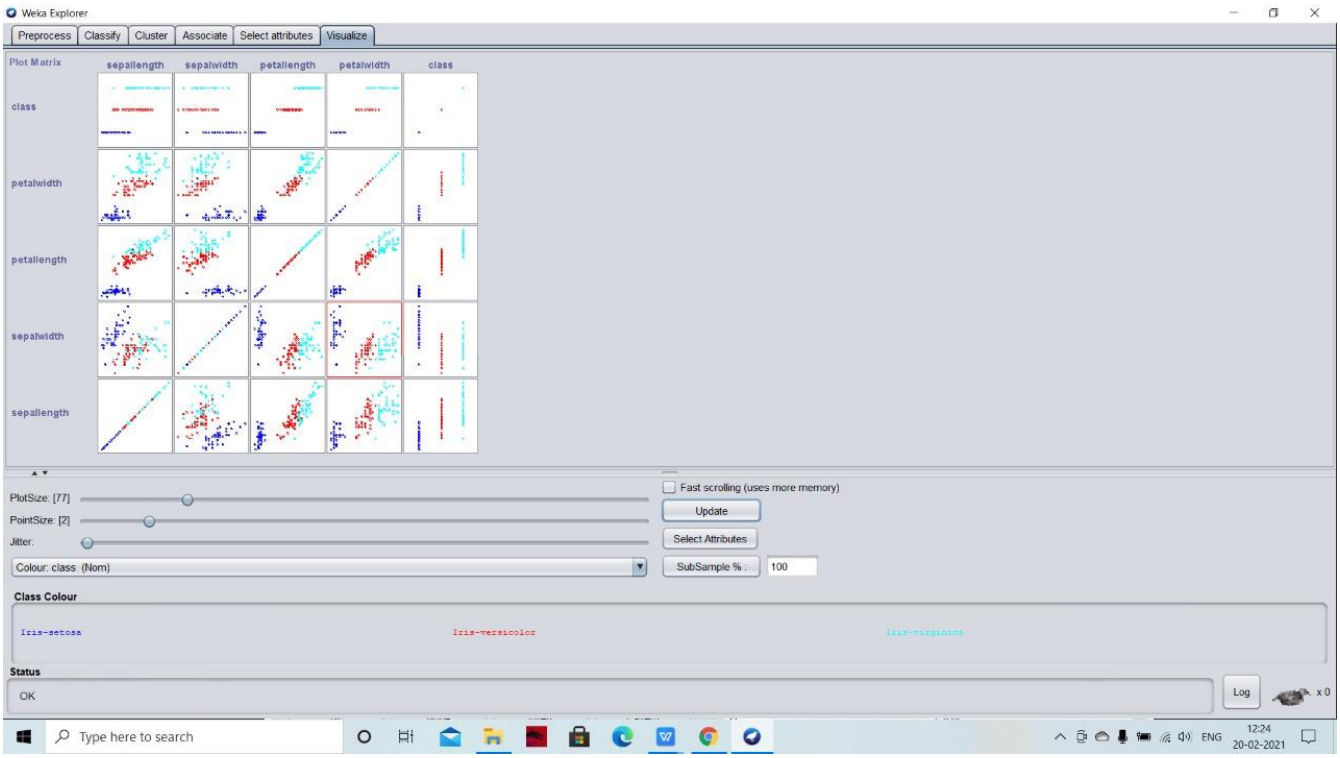
1. Open the Weka GUI Chooser.
2. Open the Weka Explorer.
3. Load the iris.arff dataset.
4. Click on different attributes in the Attributes list and review the details in the Selected attribute pane.
5. Click the Visualize All button to review all attribute distributions.
6. Click the Visualize tab and review the scatter plot matrix for all attributes.



Get comfortable reviewing the details for different attributes in the Preprocess tab and tuning the scatter plot matrix in the Visualize tab.


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
None
Apply
Stop
Current relation
Relation: iris
Instances: 150
Attributes: 5
Sum of weights: 150
Selected attribute
Name: petallength
Missing: 0 (0%)
Distinct: 43
Type: Numeric
Unique: 10 (7%)
Statistic
Value
Minimum
1
Maximum
6.9
Mean
3.759
StdDev
1.764
Attributes
All
None
Invert
Pattern
No.
Name
1 sepallength
2 sepalwidth
3 petallength
4 petalwidth
5 class
Remove
Class: class (Nom)
Visualize All
Status
OK
Log
x 0
Type here to search
ENG
12:45
20-02-2021


All attributes
sepallength
sepalwidth
petallength
petalwidth
class
3
(2.18, 3.36]
Type here to search
ENG
12:24
20-02-2021

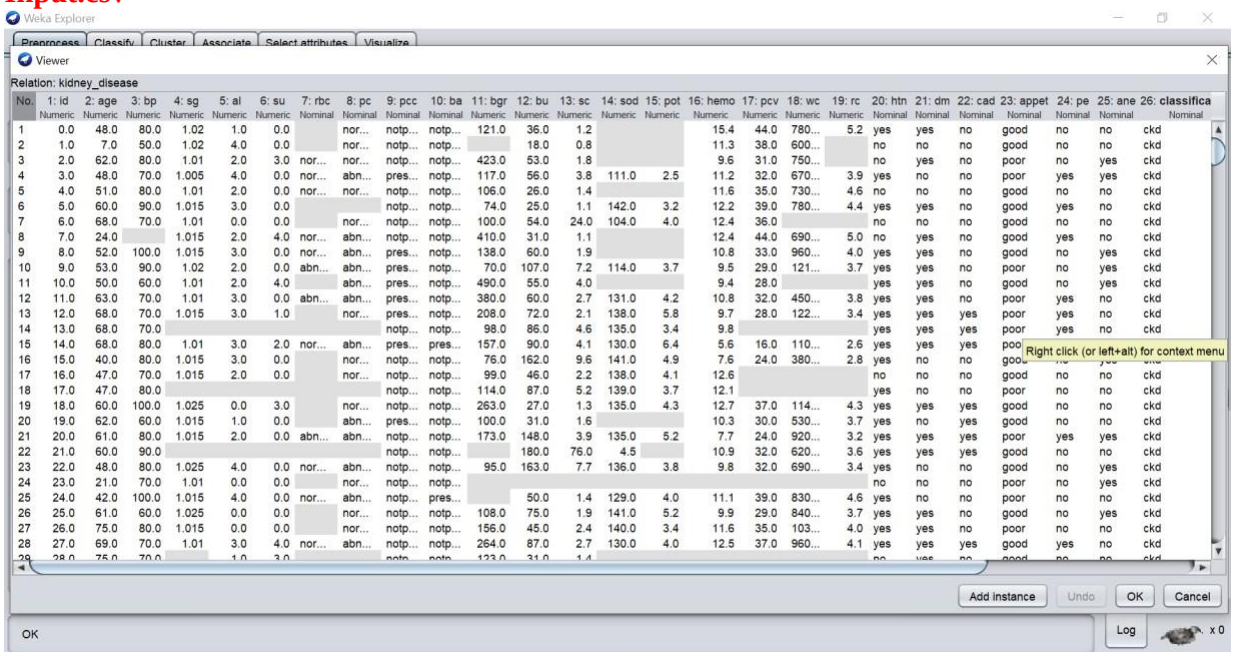


1.2 Replacing Missing Values

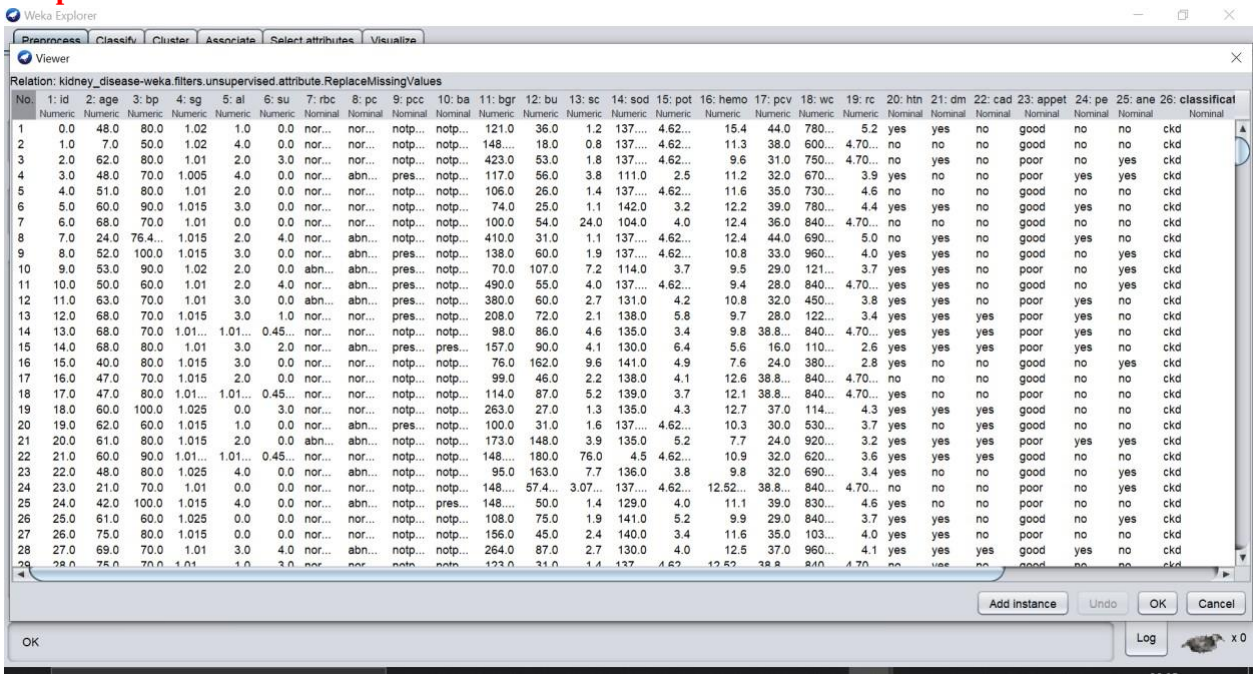
Open the input file input.csv apply ReplaceMissingValues and save output.csv

Paste your screen shots here:

Input.csv



Output.csv



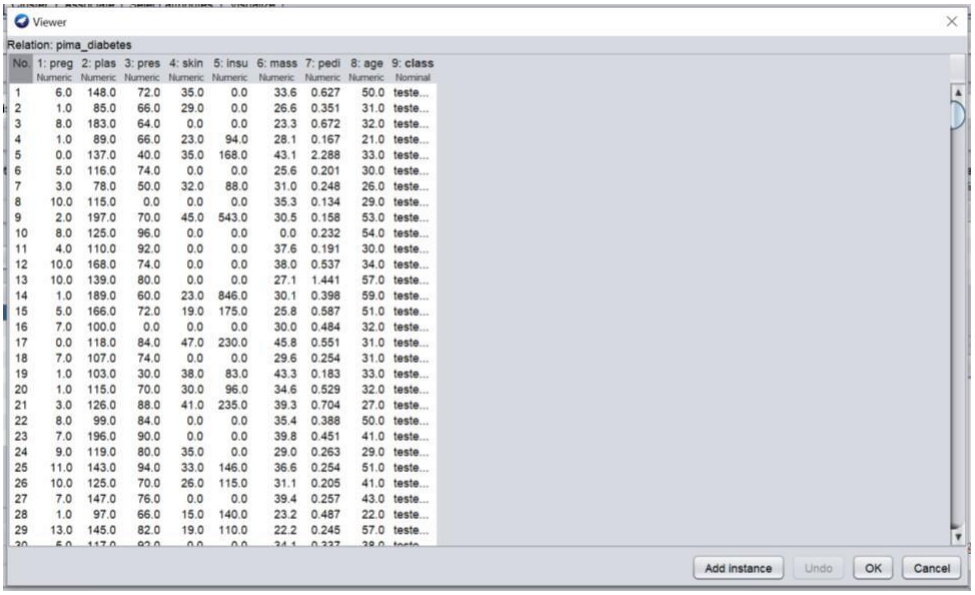
1.3 Discretization

An instance filter that discretizes a range of numeric attributes in the dataset into nominal attributes. Discretization is by simple binning. Skips the class attribute if set.

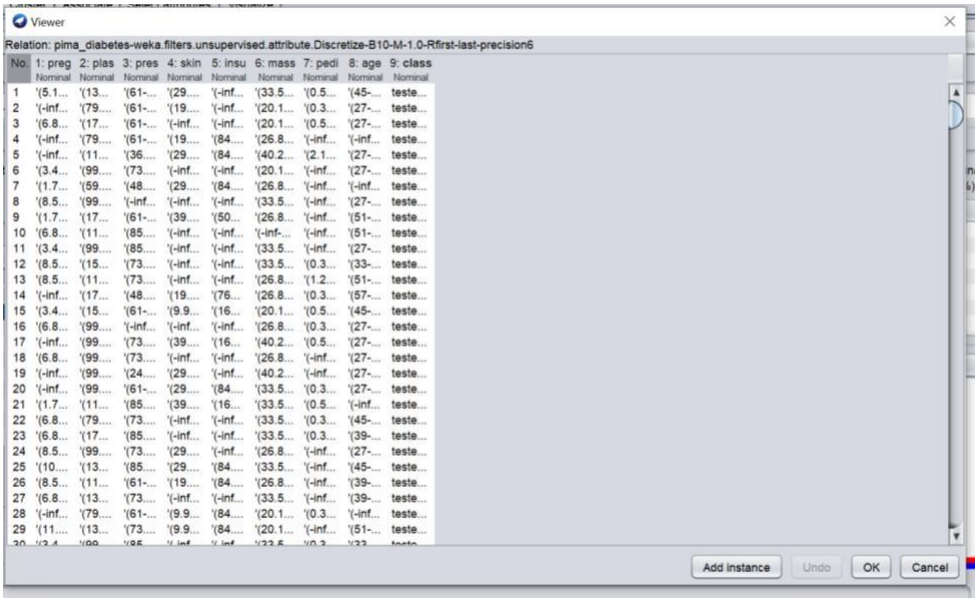
Apply weka.filters.unsupervised.attribute.Discretize on Iris dataset, take a screen shot of output.

Insert here your sample Input and Output

input.csv



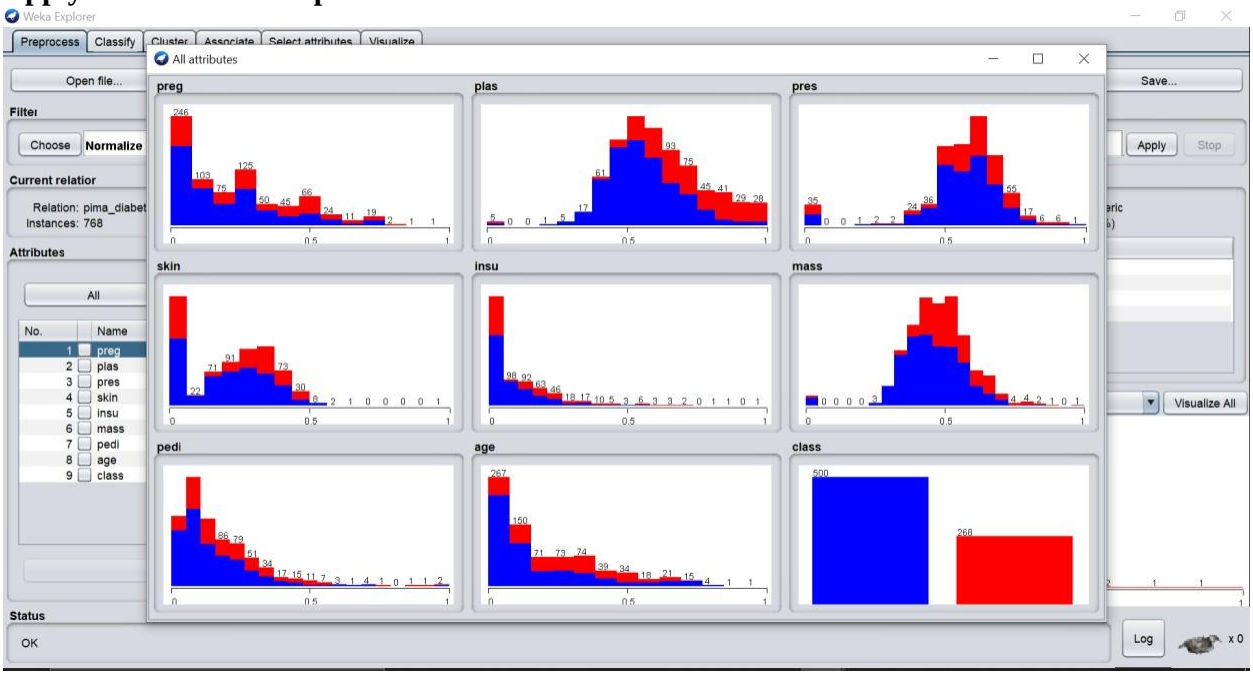
output.csv



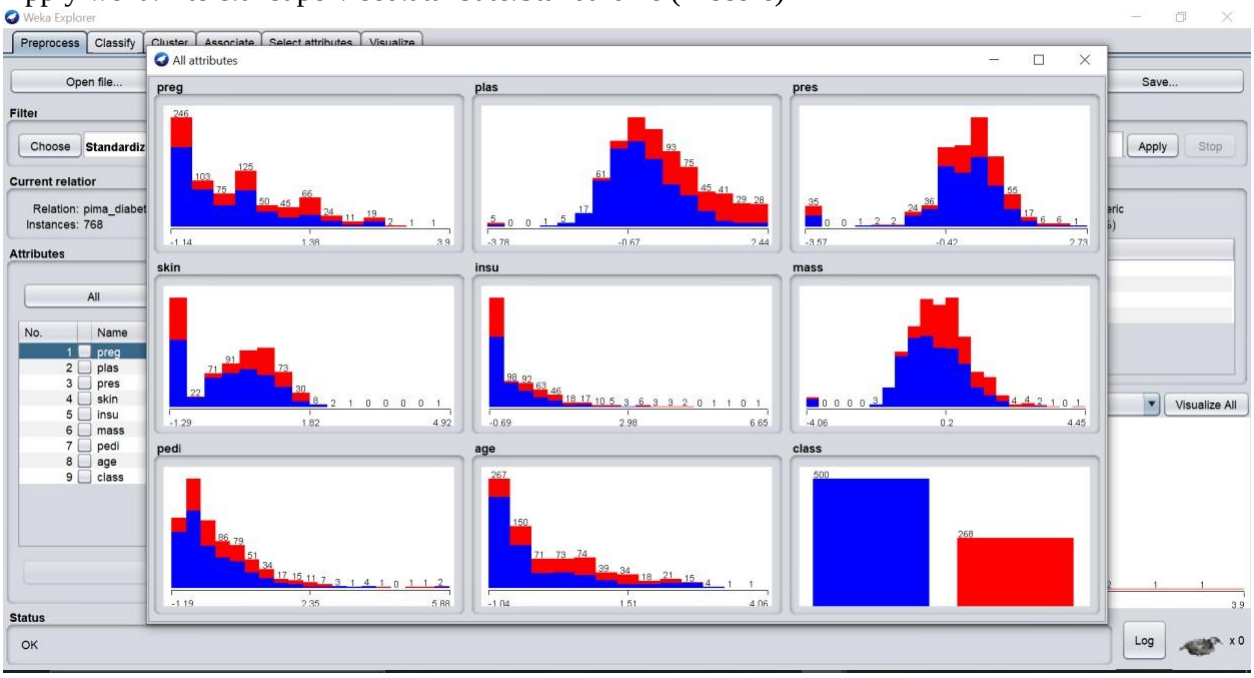
1.4 Normalisation

Where the attribute data are scaled so as to fall within a smaller range, such as 1.0 to 1.0, or 0.0 to 1.0.

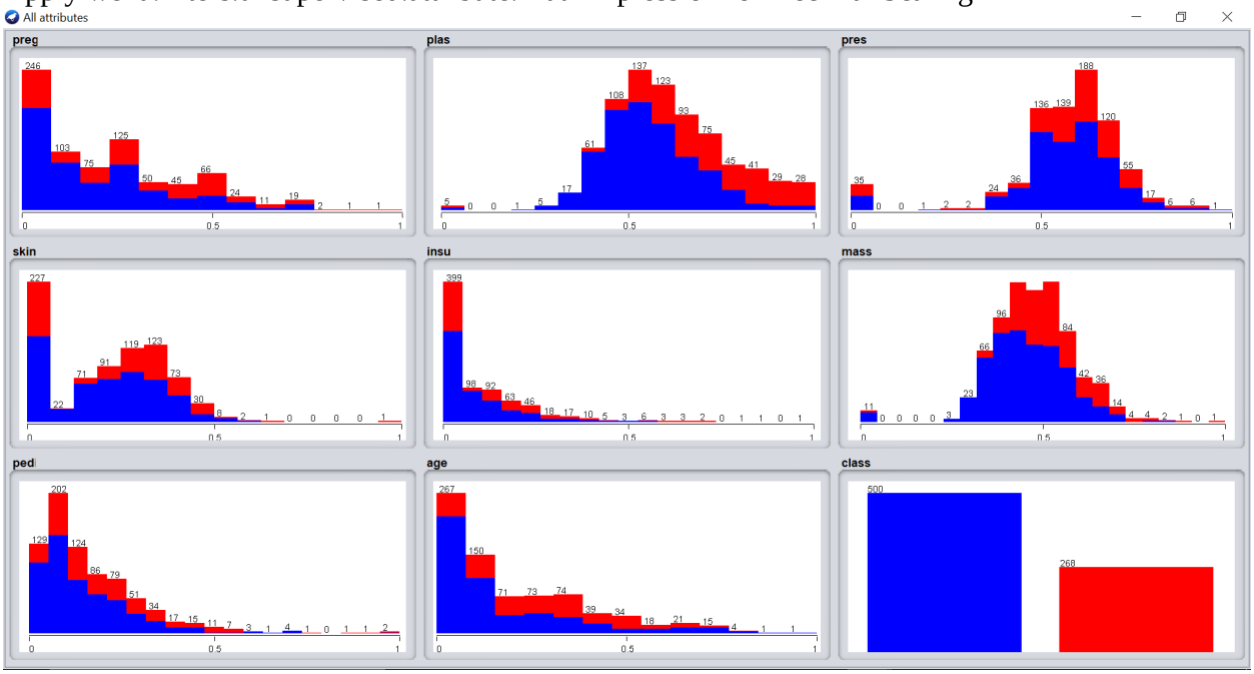
Apply weka.filters.unsupervised.attribute.Normalize for Min-Max Normalization



Apply weka.filters.unsupervised.attribute.Standardize (Z-score)



Apply weka.filters.unsupervised.attribute.MathExpression for Decimal Scaling



Fill the table below: (for diabetes.arff)

Attribute(Initial scale)	Min-Max	Standardize	Decimal Scaling
preg (0-17)	0-1	-1.141-3.904	0-1
plas(0-199)	0-1	-3.781-2.443	0-1
pres(0-122)	0-1	-3.57-2.733	0-1
skin(0-99)	0-1	-1.287-4.919	0-1
insu(0-846)	0-1	-0.692-6.649	0-1
mass(0-67.1)	0-1	-4.058-4.453	0-1
pedi(0.078-2.42)	0-1	-1.189-5.88	0-1
age(21-81)	0-1	-1.041-4.061	0-1

1.5 Sampling

Apply weka.filters.unsupervised.instance.Resample on iris dataset

Produces a random subsample of a dataset using either sampling with replacement or without replacement.

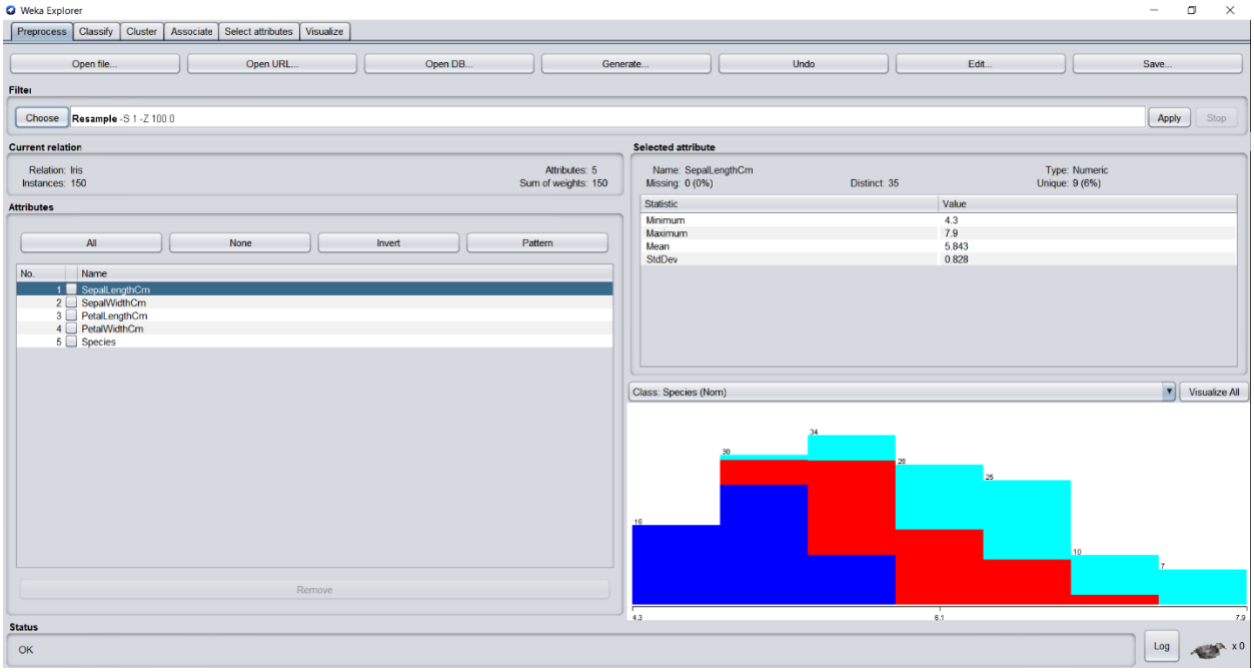
sampleSizePercent – Set Size of the subsample as a percentage of the original data say 50%.

Output: Fill the table below with No of tuples in your input and output .

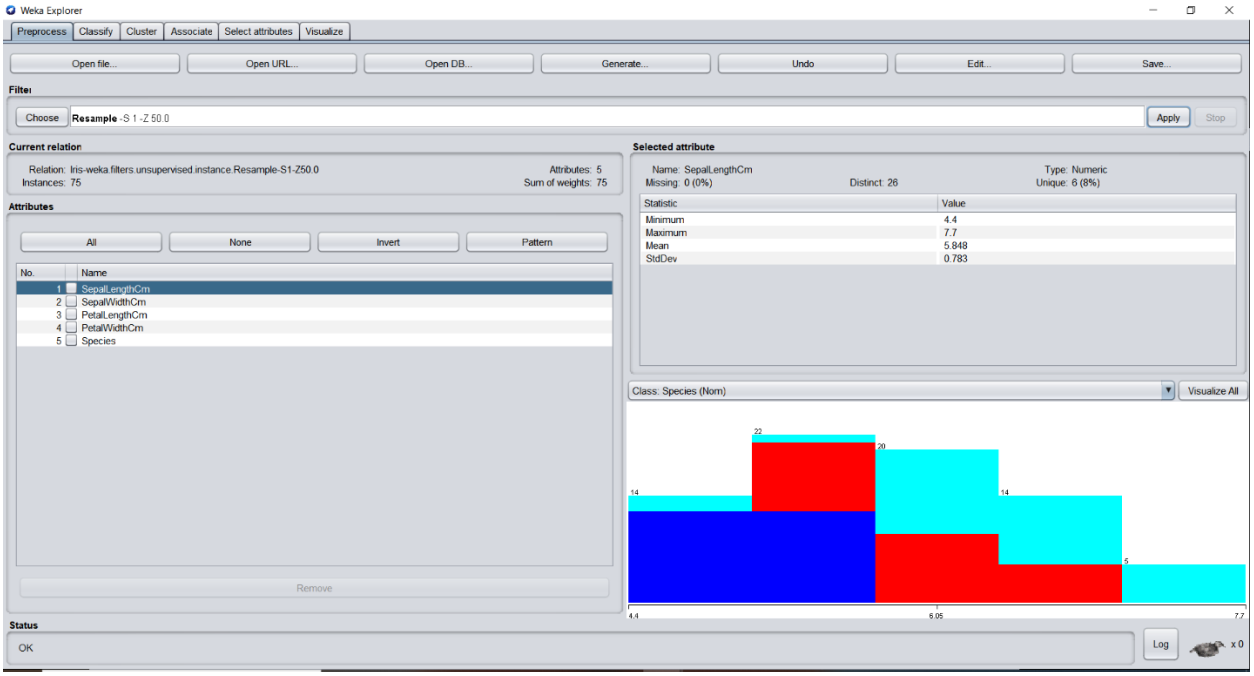
And take a scree shot of input and output and paste here

No. of tuples Before Sampling	No. of tuples After Sampling
150	

Input



Output



1.5 Perform Feature Selection

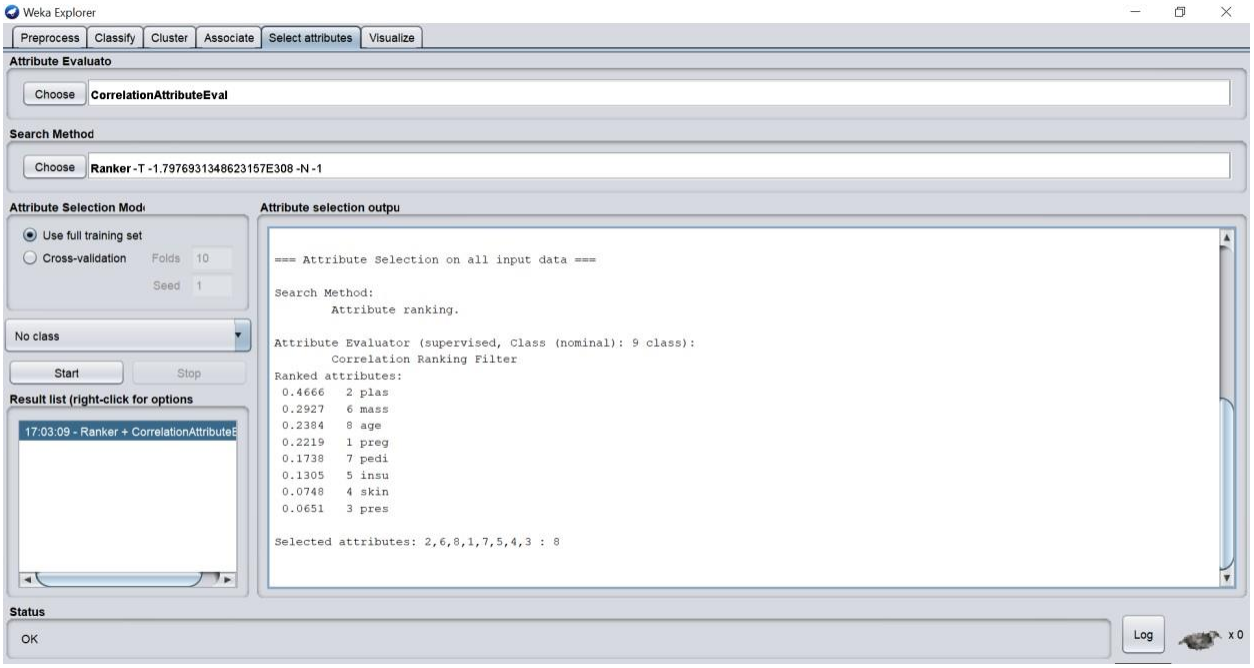
Not all of the attributes in your dataset may be relevant to the attribute you want to predict. You can use feature selection to identify those attributes that are most relevant to your output variable. The following procedure can be used for feature selection

1. Open the Weka GUI Chooser and then the Weka Explorer.
2. Load the iris.arff dataset.
3. Click the Select attributes tab.
4. Click the Choose button in the Attribute Evaluator pane and select the CorrelationAttributeEval.

(a) You will be presented with a dialog to change to the Ranker search method, needed when using this feature selection method. Click the Yes button.

5. Click the Start button to run the feature selection method.

Determining the Correlation score of attributes in diabetes dataset



Review the output in the Attribute selection output pane and note the correlation scores for each attribute, the larger numbers indicating the more relevant features. Explore other feature selection methods such as the use of information gain (entropy). Using feature selection, find out the important features and remove other features from dataset using remove button in Preprocess tab.

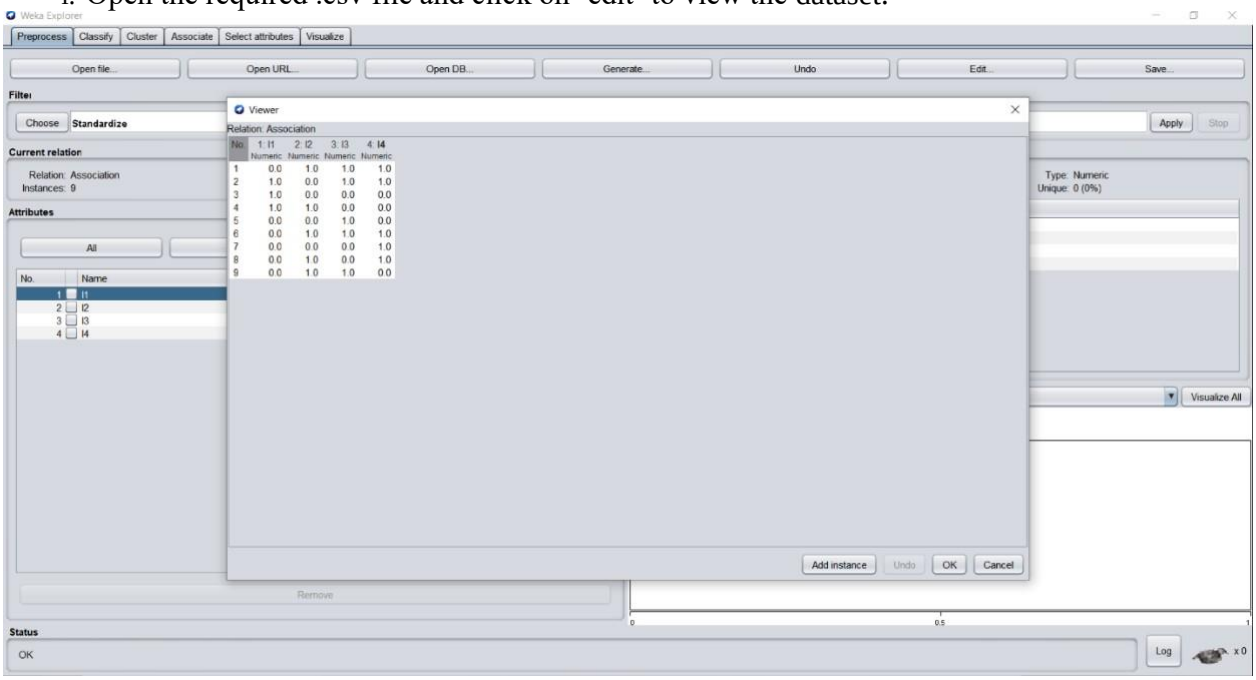
2. ASSOCIATION MINING

Step by Step Procedure

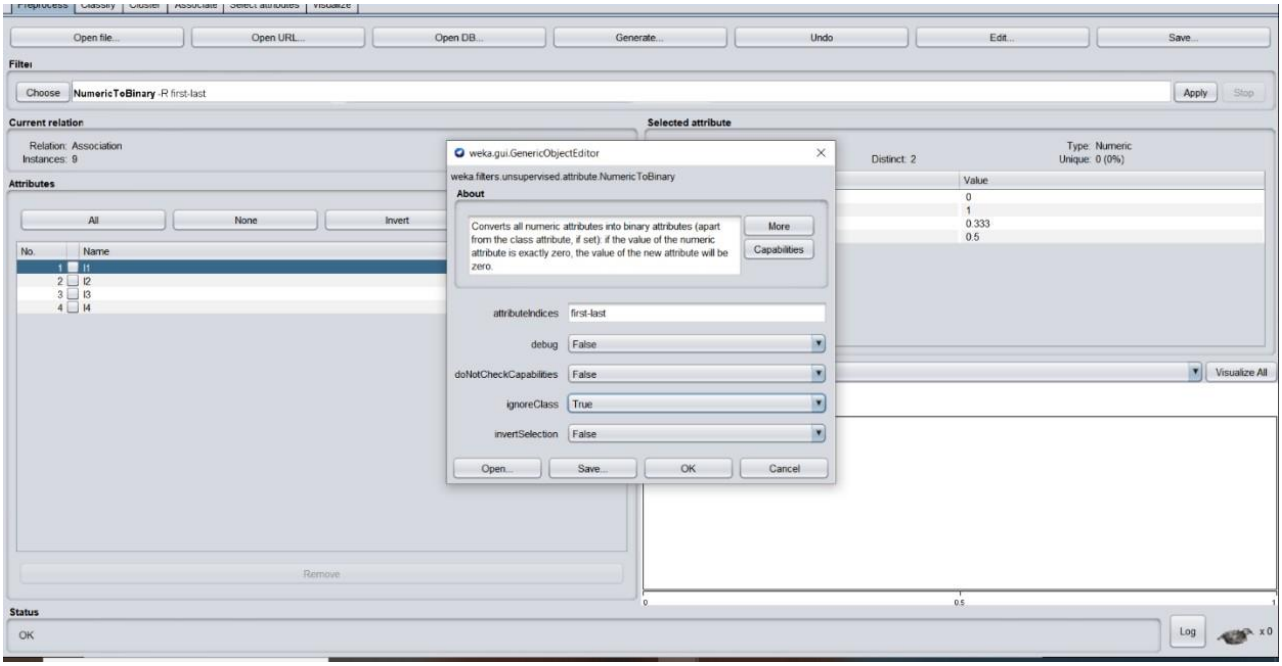
Practice these steps with your own dataset and upload the screen shots after each step here.

Find Frequent patterns using Apriori and FP Growth algorithms and generate strong association rules using WEKA on the following dataset 2.1 Apriori algorithm

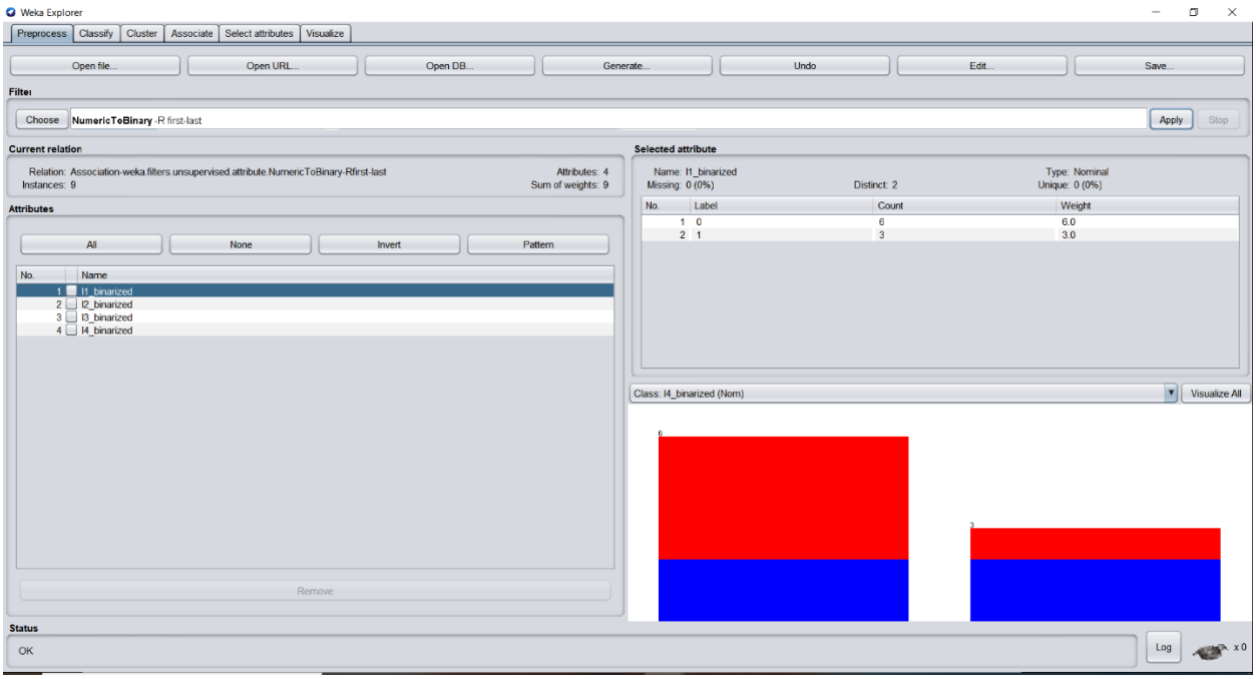
i. Open the required .csv file and click on 'edit' to view the dataset.



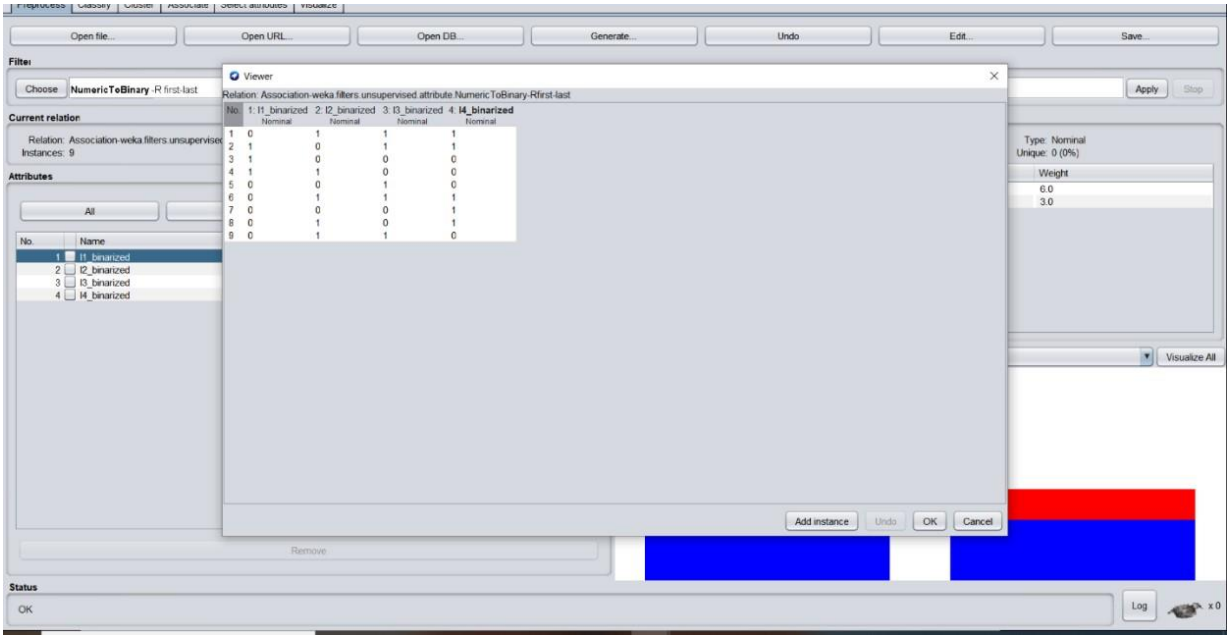
ii. Use NumericToBinary filter to convert the given numeric data to binary and click on that text box to see the description of that filter and change the settings of that filter.



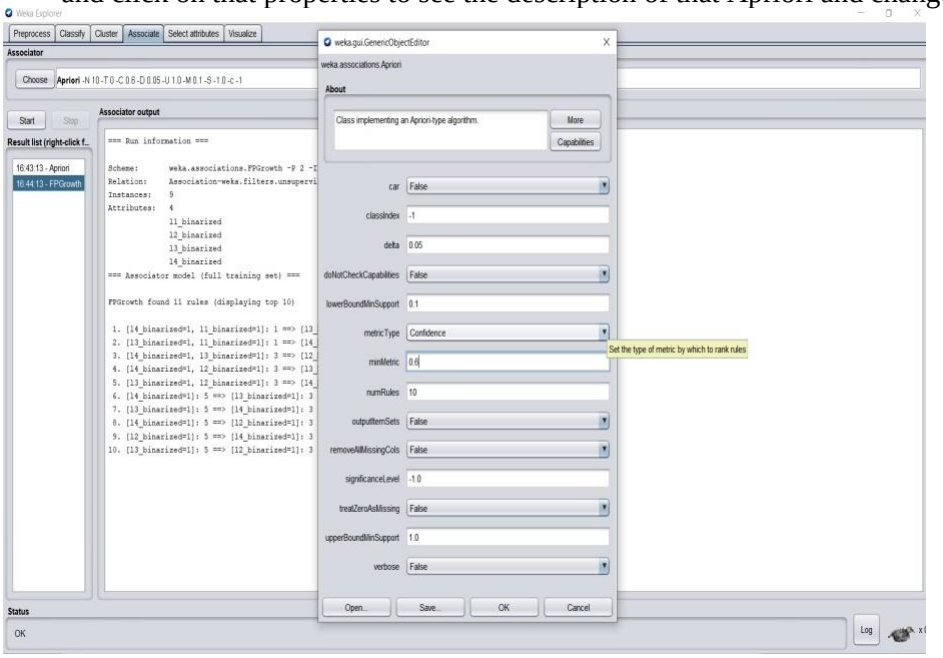
iii. In the below window we can see the binarized attributes.



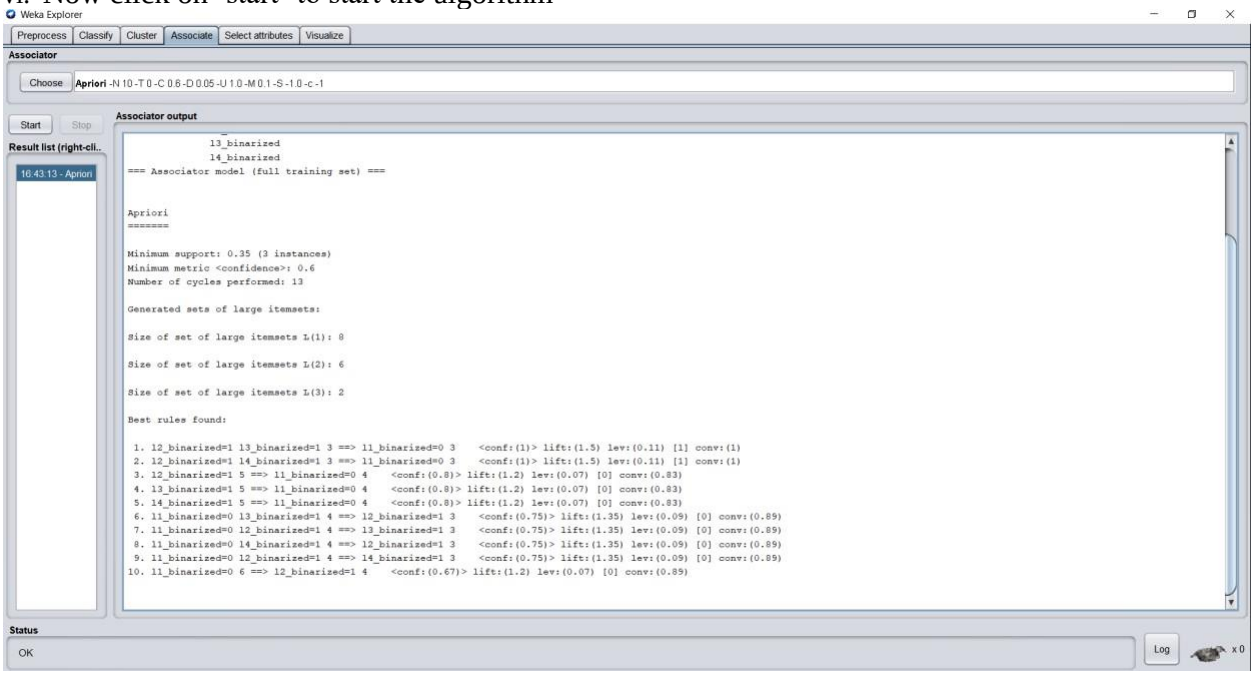
iv. Click on edit to check the dataset values after conversion.


Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
NumericToBinary -R first-last
Apply
Stop
Current relation
Relation: Association-weka.filters.unsupervised
Instances: 9
Attributes
All
No.
Name
1 I1_binarized
2 I2_binarized
3 I3_binarized
4 I4_binarized
Remove
Type: Nominal
Unique: 0 (0%)
Weight
6.0
3.0
Visualize All
Viewer
Relation: Association-weka.filters.unsupervised.attribute.NumericToBinary-Rfirst-last
No. 1: I1_binarized Nominal 2: I2_binarized Nominal 3: I3_binarized Nominal 4: I4_binarized Nominal
1 0 1 1 1
2 1 0 1 1
3 1 0 0 0
4 1 1 0 0
5 0 0 1 0
6 0 1 1 1
7 0 0 0 1
8 0 1 0 1
9 0 1 1 0
Add instance
Undo
OK
Cancel
Status
OK
Log
x 0

v. Go to Associate tab and click on choose to select Apriori Algorithm on the given data set and click on that properties to see the description of that Apriori and change the settings.



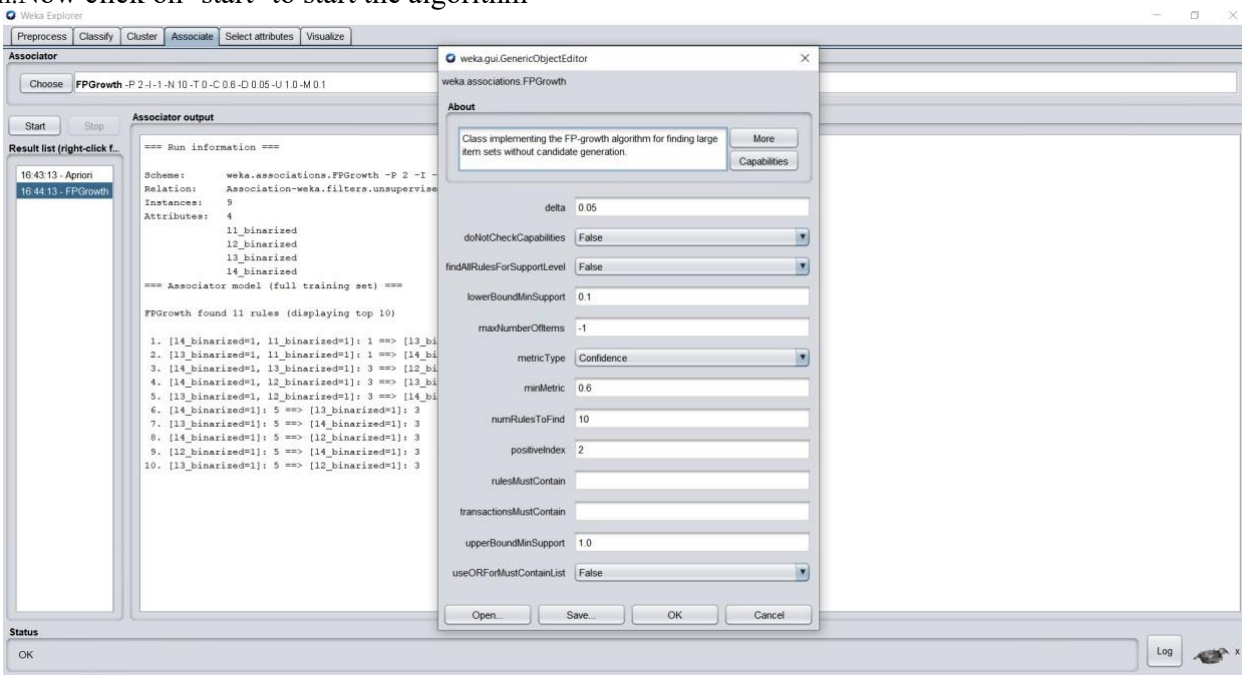
vi. Now click on 'start' to start the algorithm

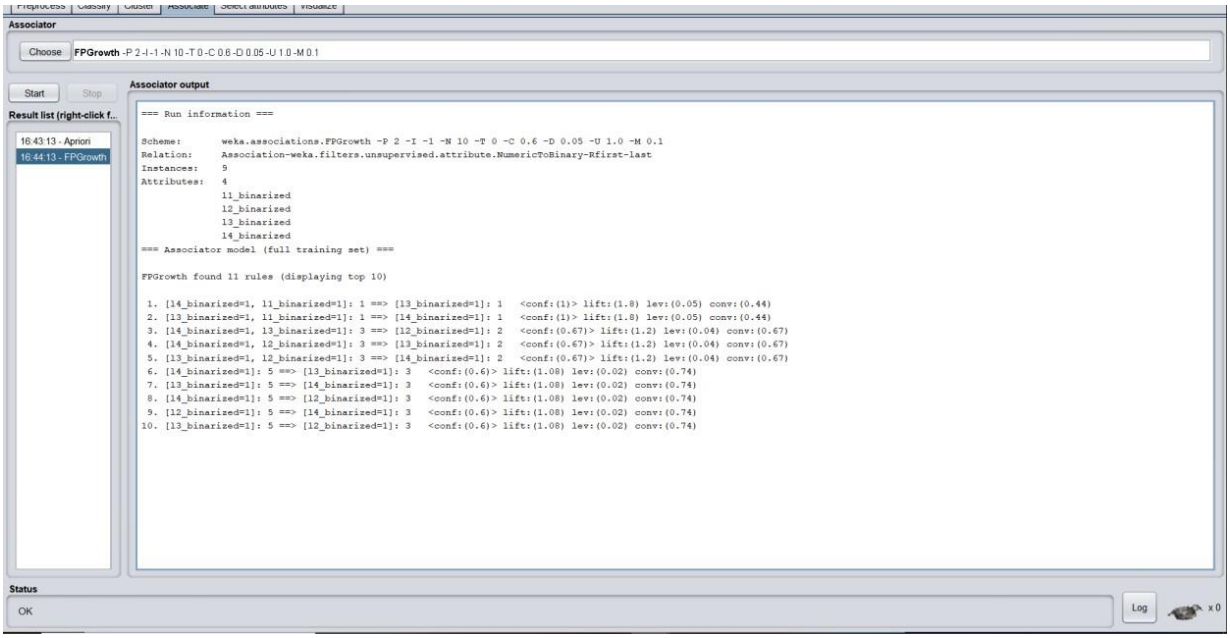


2.2 FPGrowth algorithm

i.Repeat above (i) to (iv) steps and go to Associate tab and choose to select FPGrowth Algorithm on the given data set and click on properties to see the description of that FP Growth and change the settings.

ii.Now click on ‘start’ to start the algorithm





3.CLASSIFICATION

Classification is a two step process, consisting of a learning step (where a classification model is constructed) and a classification step(where the model is used to predict class labels for given data).

Dataset:iris.arff

Number of Instances: 150 (50 in each of three classes)
Number of Attributes: 4 numeric, predictive attributes and the class
Attribute Information:

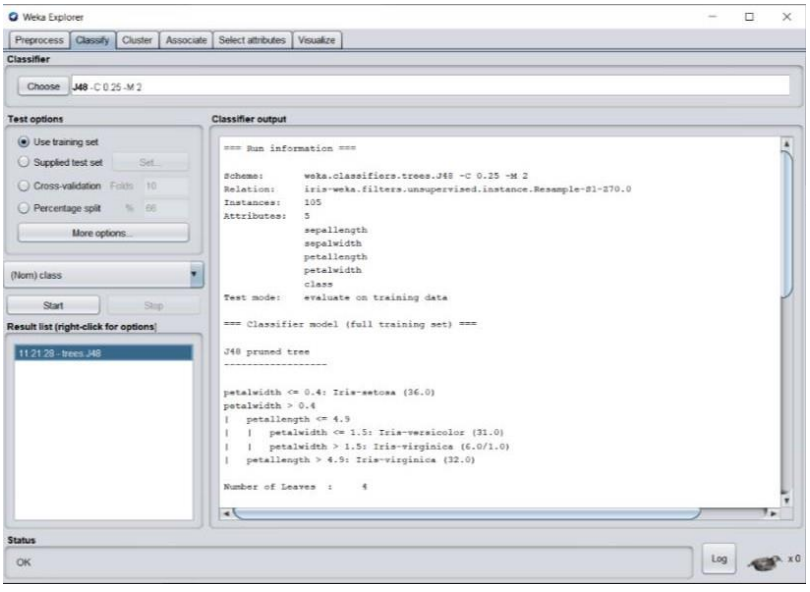
1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
   - Iris Setosa
   - Iris Versicolour
   - Iris Virginica

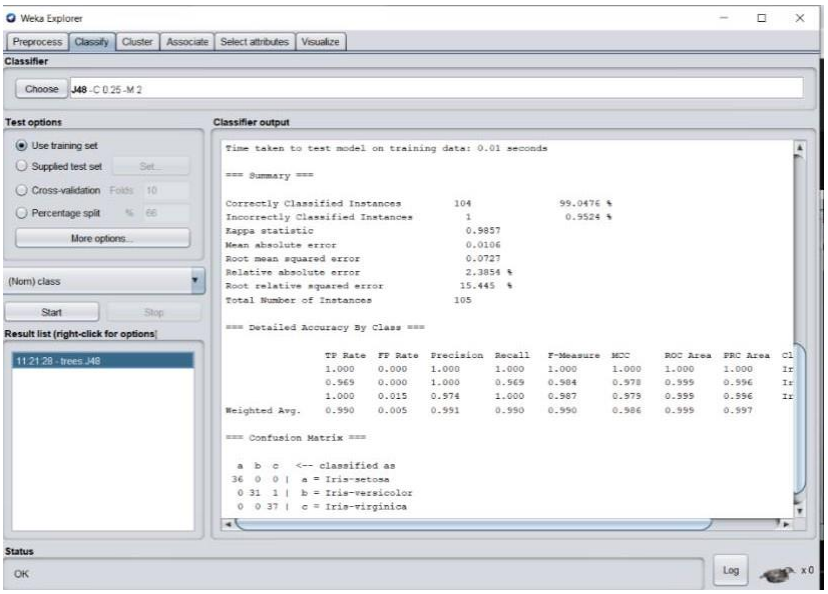
3.1 Desicion Tree Algorithm:

Decision tree is a flow chart like structure where each internal node denote test on attribute , each branch represents an outcome of test and leaf node represents classes.

Browse the iris.arff in weka apply Weka.filters.unsupervised.instance.Resample give samplesize%= 70 and apply Now save the file as iris_train.arff, ,again select iris.arff apply Resample and give samplesize%=30 and apply save the file as iris_test.arff Browse iris_train.arff apply weka.classifiers.trees.J48.

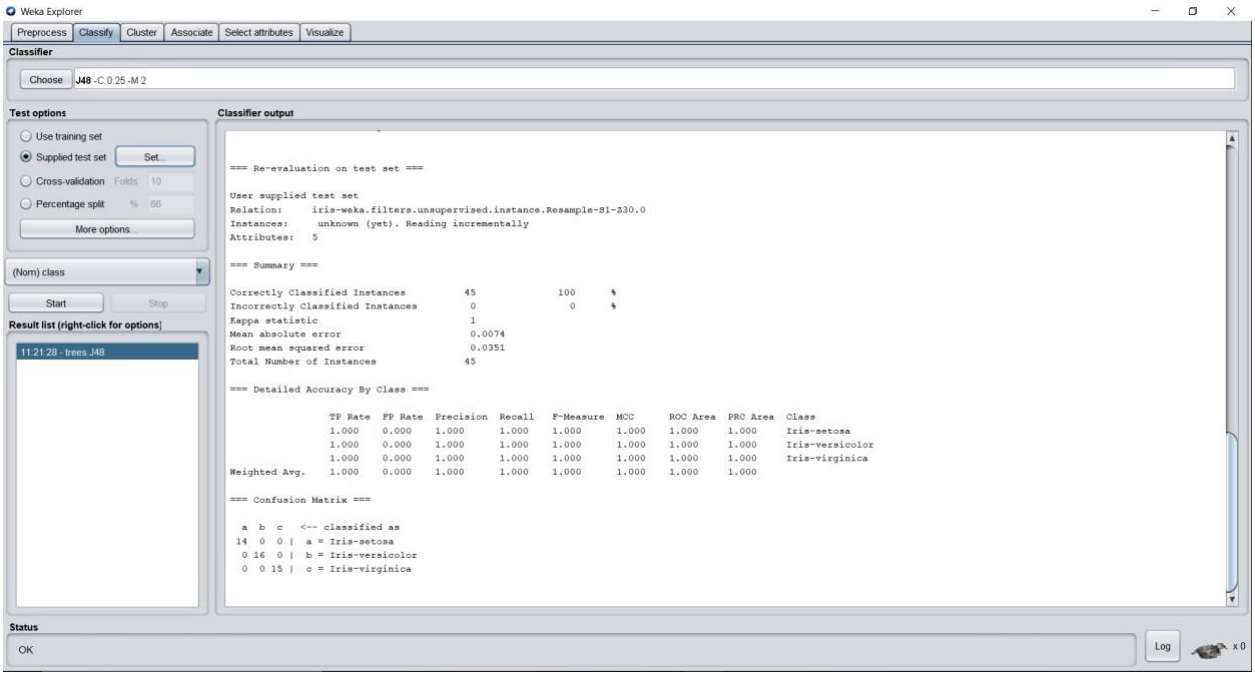
Take a screen shot of your output and paste here





Click on supplied set > set >open file > iris_test.arff. Right click & re-evaluate model on current test set

Take a screen shot of your output and paste here



And observe accuracy for given dataset.(in %).-100%

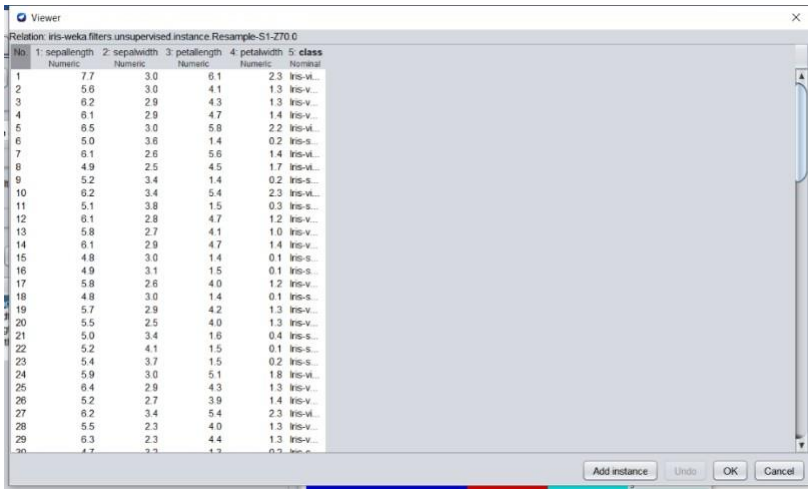
Prediction

Prediction is a process of predicting the values of data by making use of known results from a different set of sample data.Click on supplied set > set >open file > iris_prediction.arff

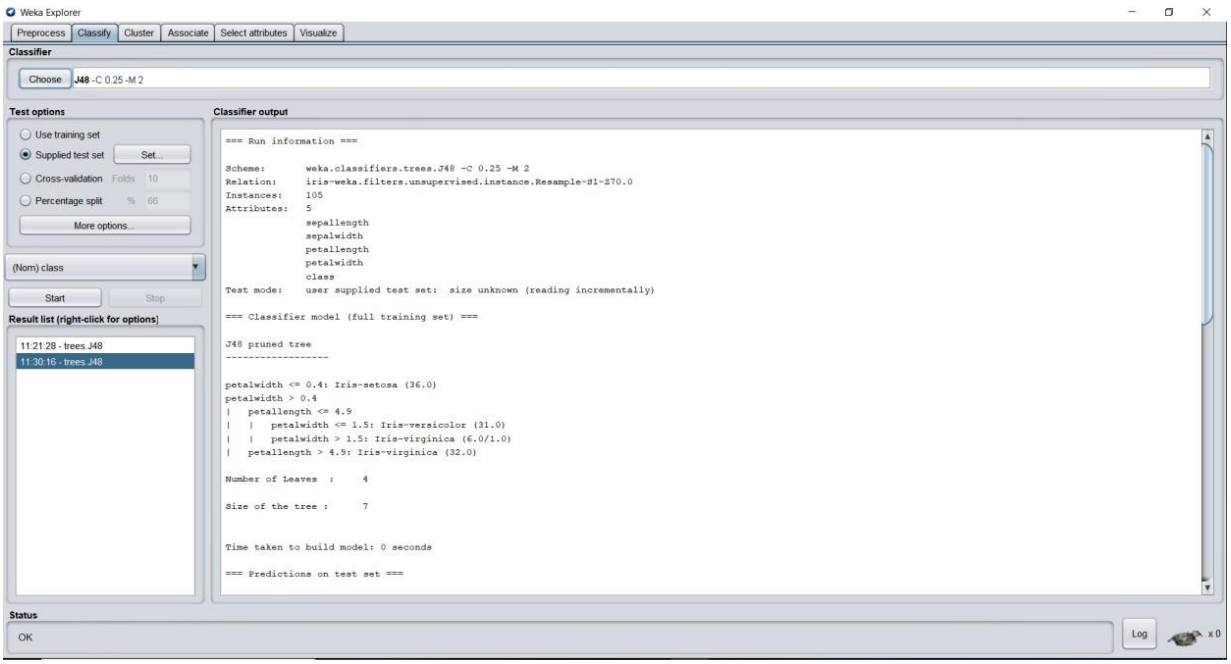
Click on more options >choose plain text .click ok and start

Take a screen shot of your output and paste here

Iris_prediction input



output

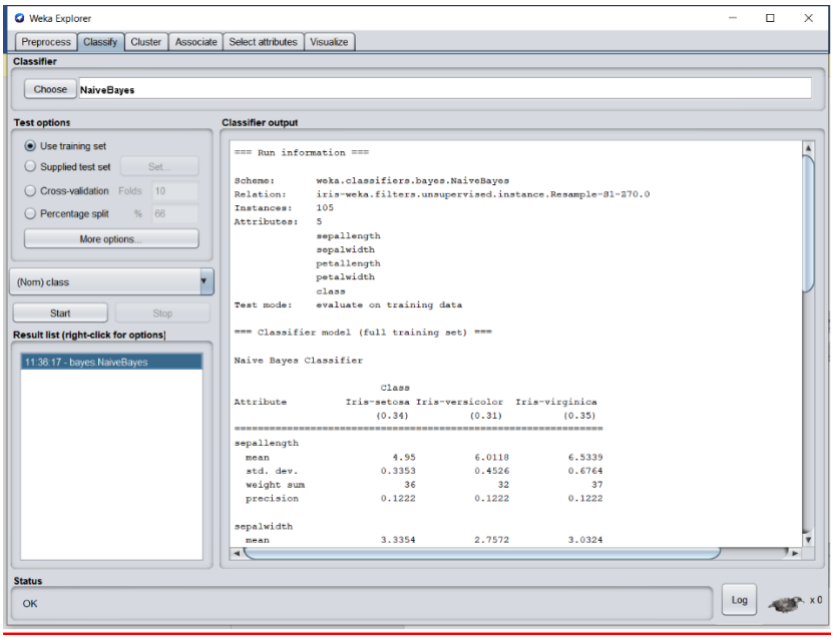


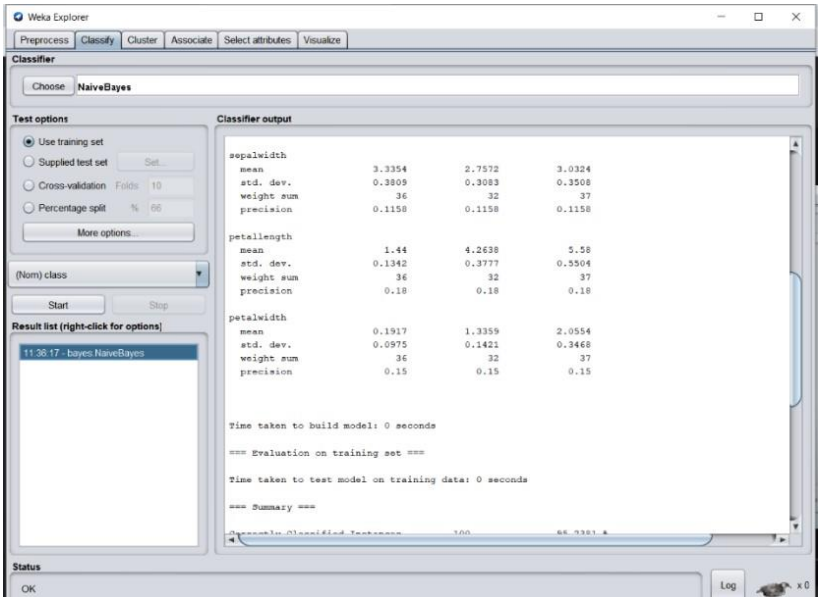
1. Naive Bayes Algorithm:

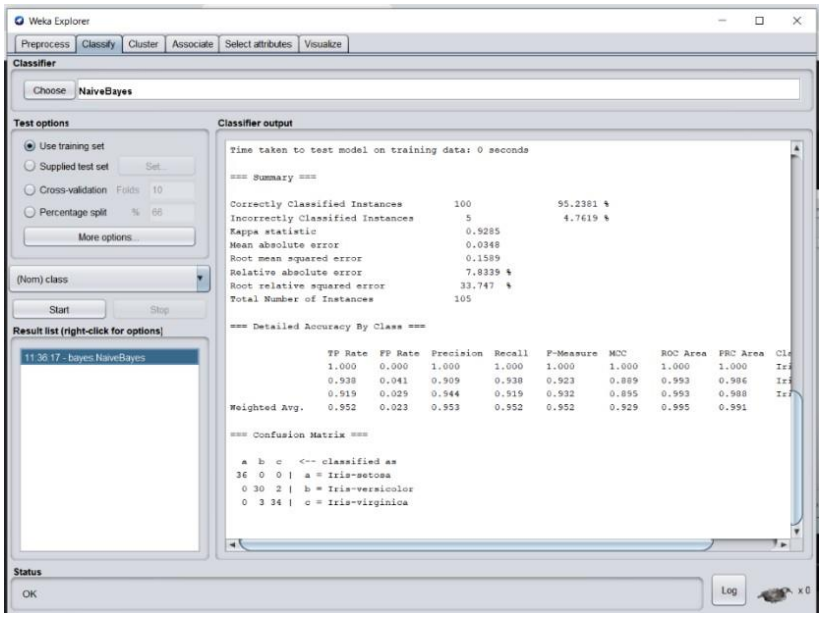
This algorithm is based on Bayes Theorem.

Browse the iris.arff in weka apply Weka.filters.unsupervised.instance.Resample give samplesize%= 70 and apply now save the file as iris_train.arff.Again select iris.arff apply Resample and give samplesize%=30 and apply save the file as iris_test.arff .Browse iris_train.arff apply weka.classifiers.bayes.naivebayes.

Take a screen shot of your output and paste here





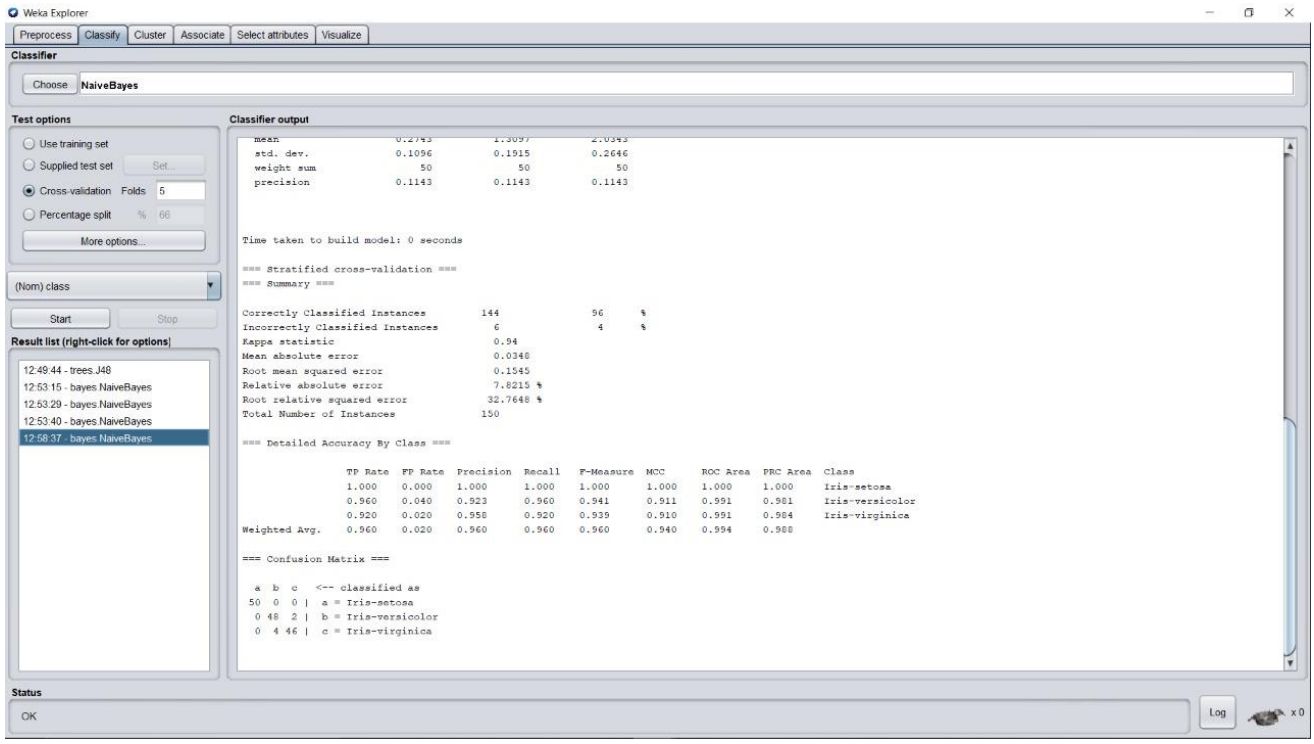


Practice these additional Exercises and observe the output

Cross Validation:

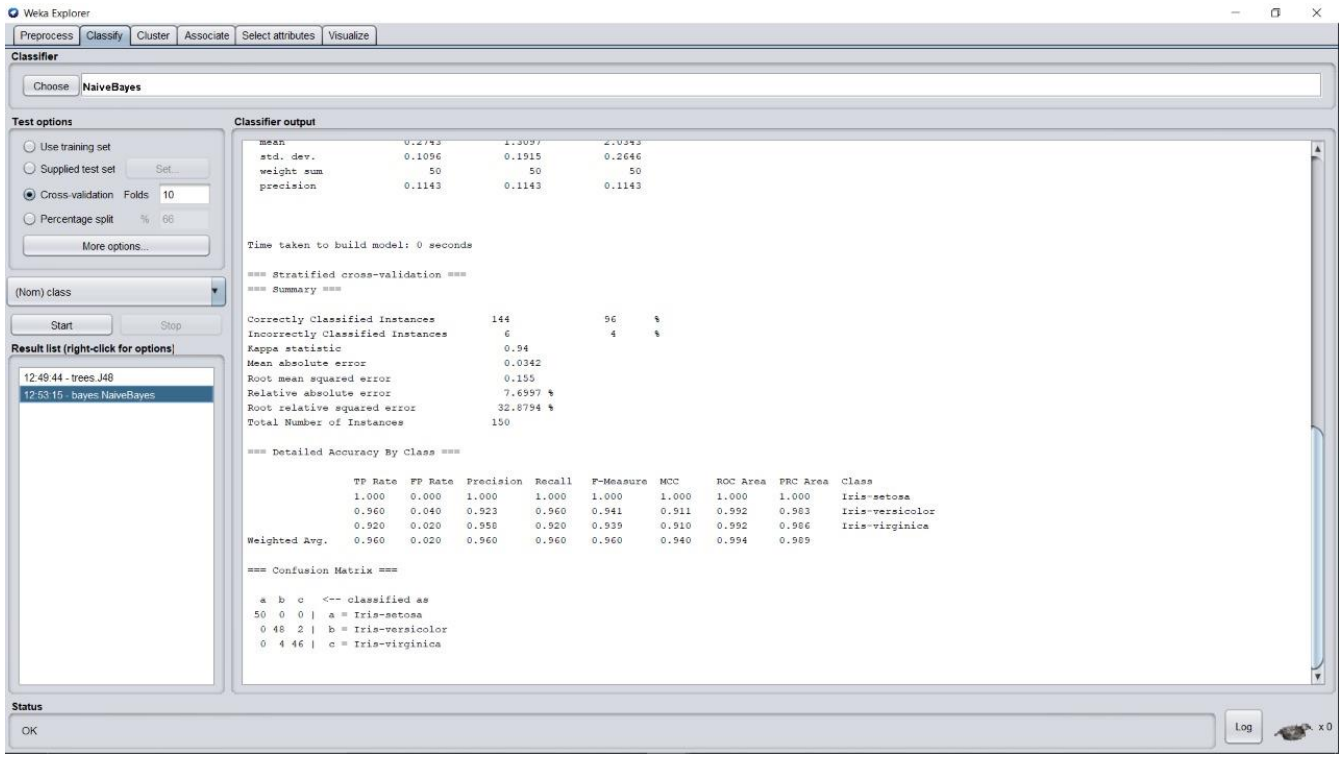
In ***K*-fold cross-validation**, the initial data are randomly partitioned into *K* mutually exclusive subsets or "folds," *D*1, *D*2, : : : , *DK*, each of approximately equal size. Training and testing is performed *K* times. In iteration *I*, partition *DI* is reserved as the test set, and the remaining partitions are collectively used to train the model. That is, in the first iteration, subsets *D*2, : :, *DK* collectively serve as the training set to obtain a first model, which is tested on *D*1; the second iteration is trained on subsets *D*1, *D*3, : : : , *DK* and tested on *D*2; and so on. the accuracy estimate is the overall number of correct classifications from the *K* iterations, divided by the total number of tuples in the initial data.

No of folds	**Accuracy**
5	96 %
10	96 %
15	95.33 %

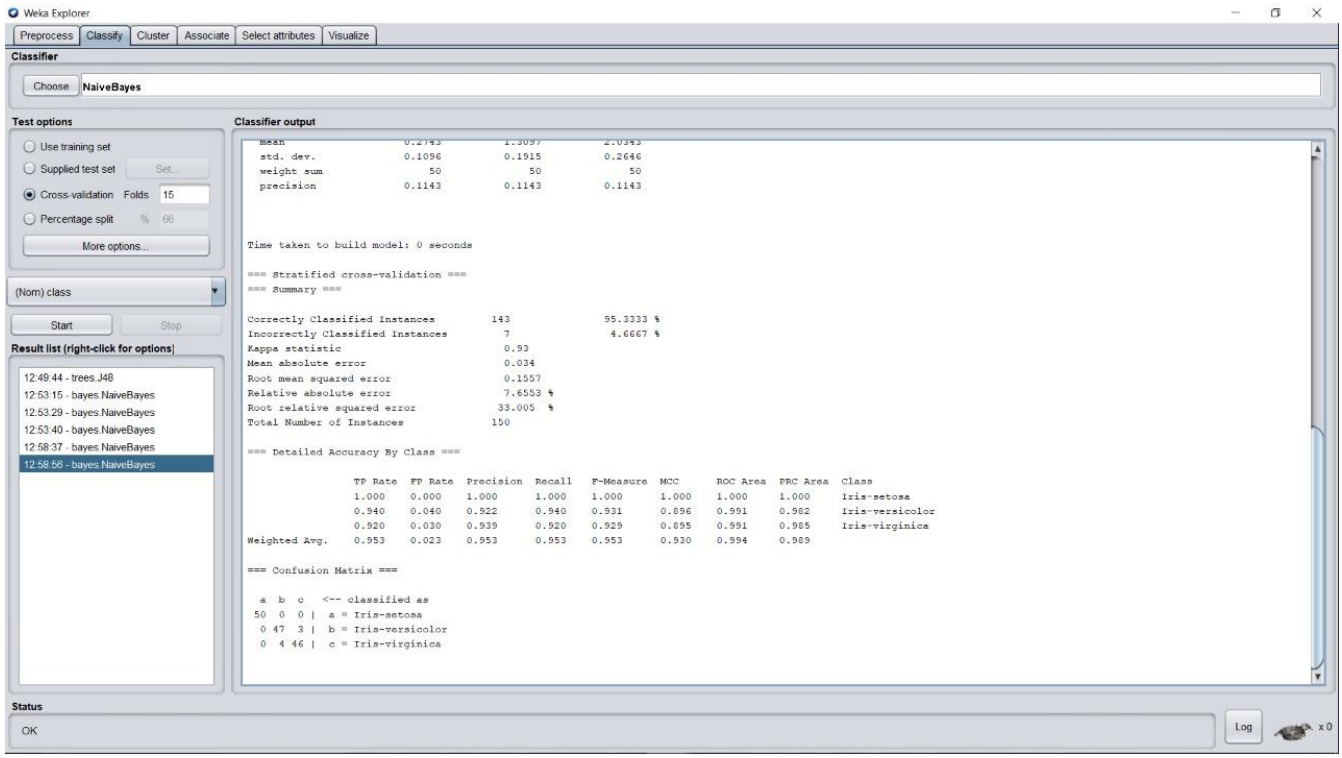


5-folds:

10-folds:


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
NaiveBayes
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
12:49:44 - trees.J48
12:53:15 - bayes.NaiveBayes
Classifier output
mean 0.2743 1.3097 2.0343
std. dev. 0.1096 0.1915 0.2646
weight sum 50 50 50
precision 0.1143 0.1143 0.1143
Time taken to build model: 0 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 144 96 %
Incorrectly Classified Instances 6 4 %
Kappa statistic 0.94
Mean absolute error 0.0342
Root mean squared error 0.155
Relative absolute error 7.6997 %
Root relative squared error 32.8794 %
Total Number of Instances 150
=== 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
0.960 0.040 0.923 0.960 0.941 0.911 0.992 0.983 Iris-versicolor
0.920 0.020 0.958 0.920 0.939 0.910 0.992 0.986 Iris-virginica
Weighted Avg. 0.960 0.020 0.960 0.960 0.960 0.940 0.994 0.989
=== Confusion Matrix ===
a b c <-- classified as
50 0 0 | a = Iris-setosa
0 48 2 | b = Iris-versicolor
0 4 46 | c = Iris-virginica
Status
OK
Log
x 0

15-folds:



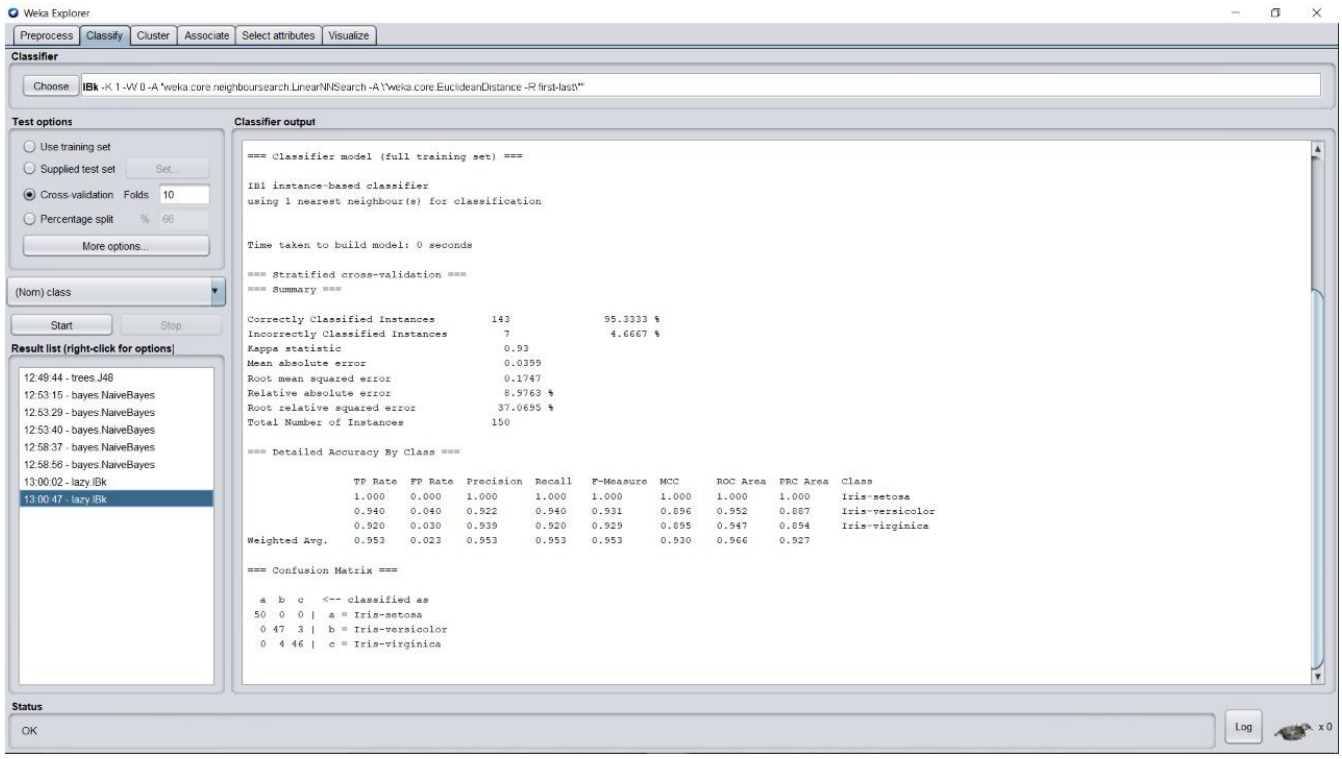
3.3KNN

KNN searches the pattern space for the k training tuples that are closest to the unknown tuple. These k training tuples are the k nearest neighbours of the unknown tuple.

Browse iris.arff in weka apply Weka.filters.unsupervised.instance.Resample give samplesize%= 70 and apply Now save the file as **iris_train.arff**, Again select iris.arff apply Resample and give samplesize%=30 and apply save the file as **iris_test.arff**

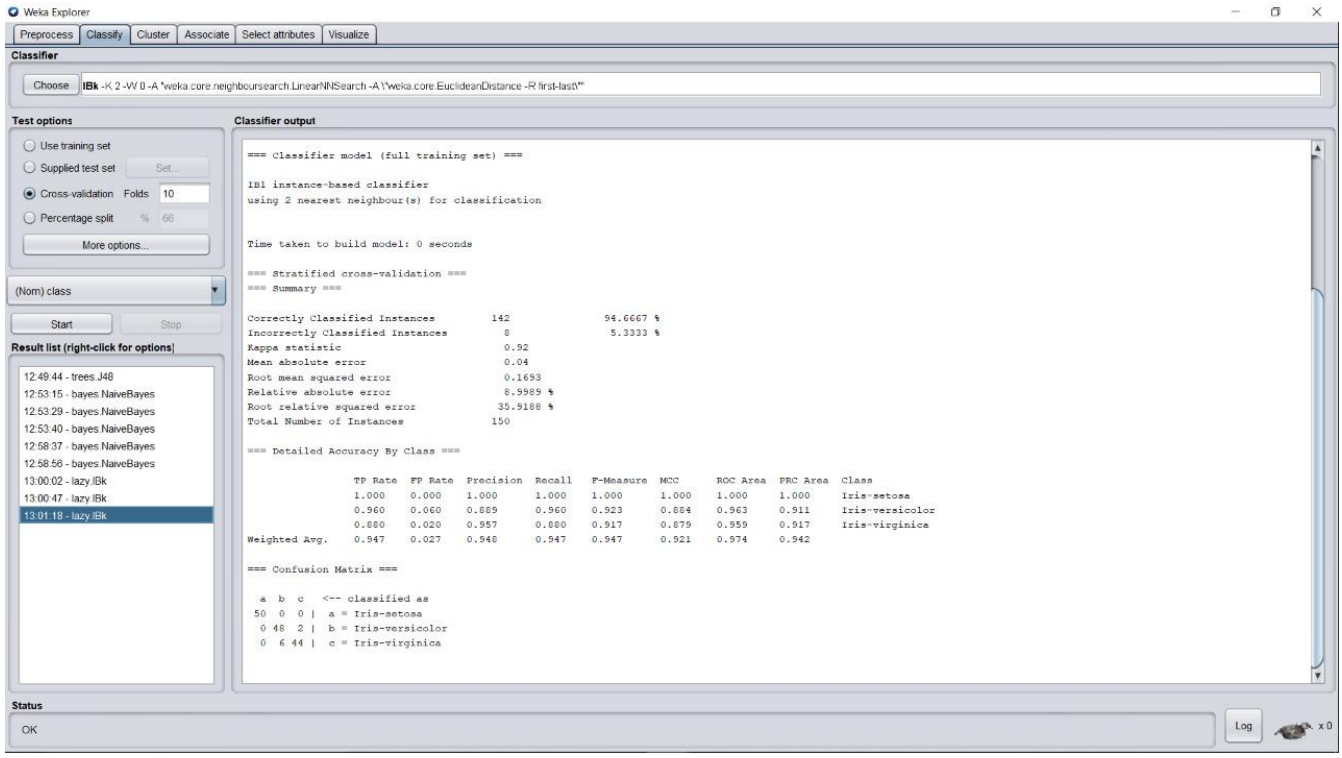
Browse iris_train.arff apply weka.classifiers.lazy.IBK

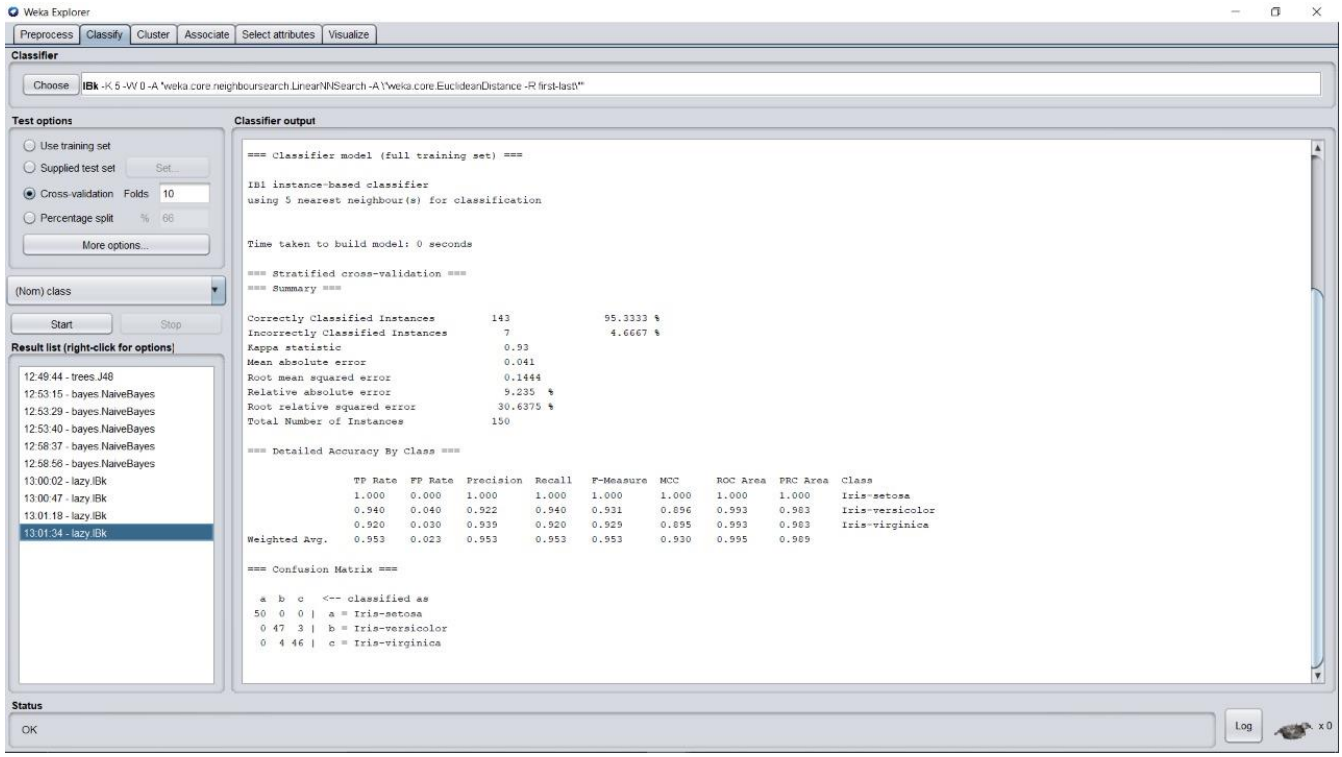
Distance Measure	K value	Time Taken(sec)	Accuracy %
LinearNNsearch	1	0.02	100%
LinearNNsearch	2	0.02	97.33%
LinearNNsearch	3	0.02	96.6667 %
LinearNNsearch	5	0.02	96%


Weka Explorer
Preprocess Classify Cluster Associate Select attributes Visualize
Classifier
Choose 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 Set...
Cross-validation Folds 10
Percentage split % 66
More options...
(Nom) class
Start Stop
Result list (right-click for options)
12:49:44 - trees.J48
12:53:15 - bayes.NaiveBayes
12:53:29 - bayes.NaiveBayes
12:53:40 - bayes.NaiveBayes
12:58:37 - bayes.NaiveBayes
12:58:56 - bayes.NaiveBayes
13:00:02 - lazy.IBk
13:00:47 - lazy.IBk
Classifier output
=== Classifier model (full training set) ===
IB1 instance-based classifier
using 1 nearest neighbour(s) for classification
Time taken to build model: 0 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 143 95.3333 %
Incorrectly Classified Instances 7 4.6667 %
Kappa statistic 0.93
Mean absolute error 0.0399
Root mean squared error 0.1747
Relative absolute error 8.9763 %
Root relative squared error 37.0695 %
Total Number of Instances 150
=== 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
0.940 0.040 0.922 0.940 0.931 0.896 0.952 0.887 Iris-versicolor
0.920 0.030 0.939 0.920 0.929 0.895 0.947 0.894 Iris-virginica
Weighted Avg. 0.953 0.023 0.953 0.953 0.953 0.930 0.966 0.927
=== Confusion Matrix ===
a b c <-- classified as
50 0 0 | a = Iris-setosa
0 47 3 | b = Iris-versicolor
0 4 46 | c = Iris-virginica
Status
OK
Log
x 0

k=1

k=2


Weka Explorer
Preprocess Classify Cluster Associate Select attributes Visualize
Classifier
Choose IBk -K 2 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""
Test options
Use training set
Supplied test set Set...
Cross-validation Folds 10
Percentage split % 66
More options...
(Nom) class
Start Stop
Result list (right-click for options)
12:49:44 - trees.J48
12:53:15 - bayes.NaiveBayes
12:53:29 - bayes.NaiveBayes
12:53:40 - bayes.NaiveBayes
12:58:37 - bayes.NaiveBayes
12:58:56 - bayes.NaiveBayes
13:00:02 - lazy.IBk
13:00:47 - lazy.IBk
13:01:18 - lazy.IBk
Classifier output
=== Classifier model (full training set) ===
IB1 instance-based classifier
using 2 nearest neighbour(s) for classification
Time taken to build model: 0 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 142 94.6667 %
Incorrectly Classified Instances 8 5.3333 %
Kappa statistic 0.92
Mean absolute error 0.04
Root mean squared error 0.1693
Relative absolute error 8.9989 %
Root relative squared error 35.9188 %
Total Number of Instances 150
=== 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
0.960 0.060 0.889 0.960 0.923 0.884 0.963 0.911 Iris-versicolor
0.880 0.020 0.957 0.880 0.917 0.879 0.959 0.917 Iris-virginica
Weighted Avg. 0.947 0.027 0.948 0.947 0.947 0.921 0.974 0.942
=== Confusion Matrix ===
a b c <-- classified as
50 0 0 | a = Iris-setosa
0 48 2 | b = Iris-versicolor
0 6 44 | c = Iris-virginica
Status
OK
Log
x 0



k=5

3.4 Multilayer Perceptron

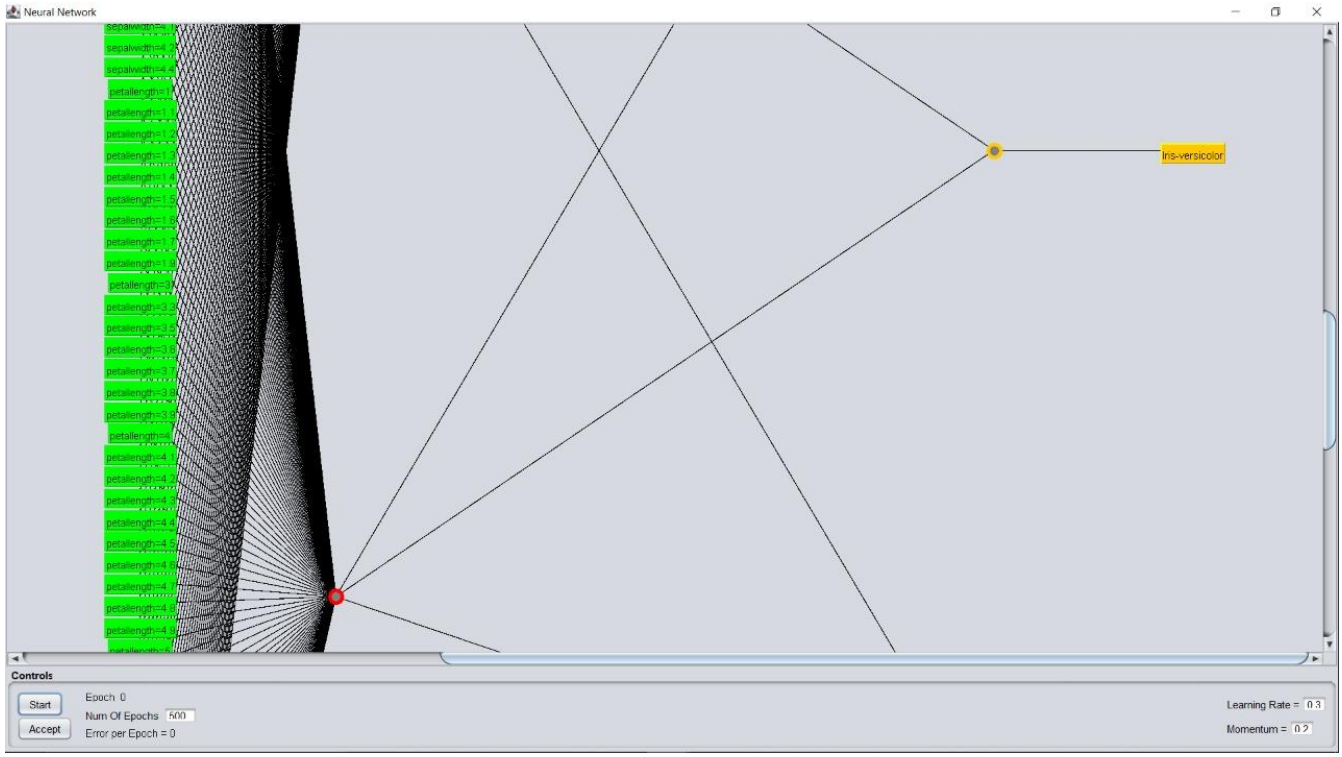
Apply weka.filters.unsupervised.attribute.NumericToNominal

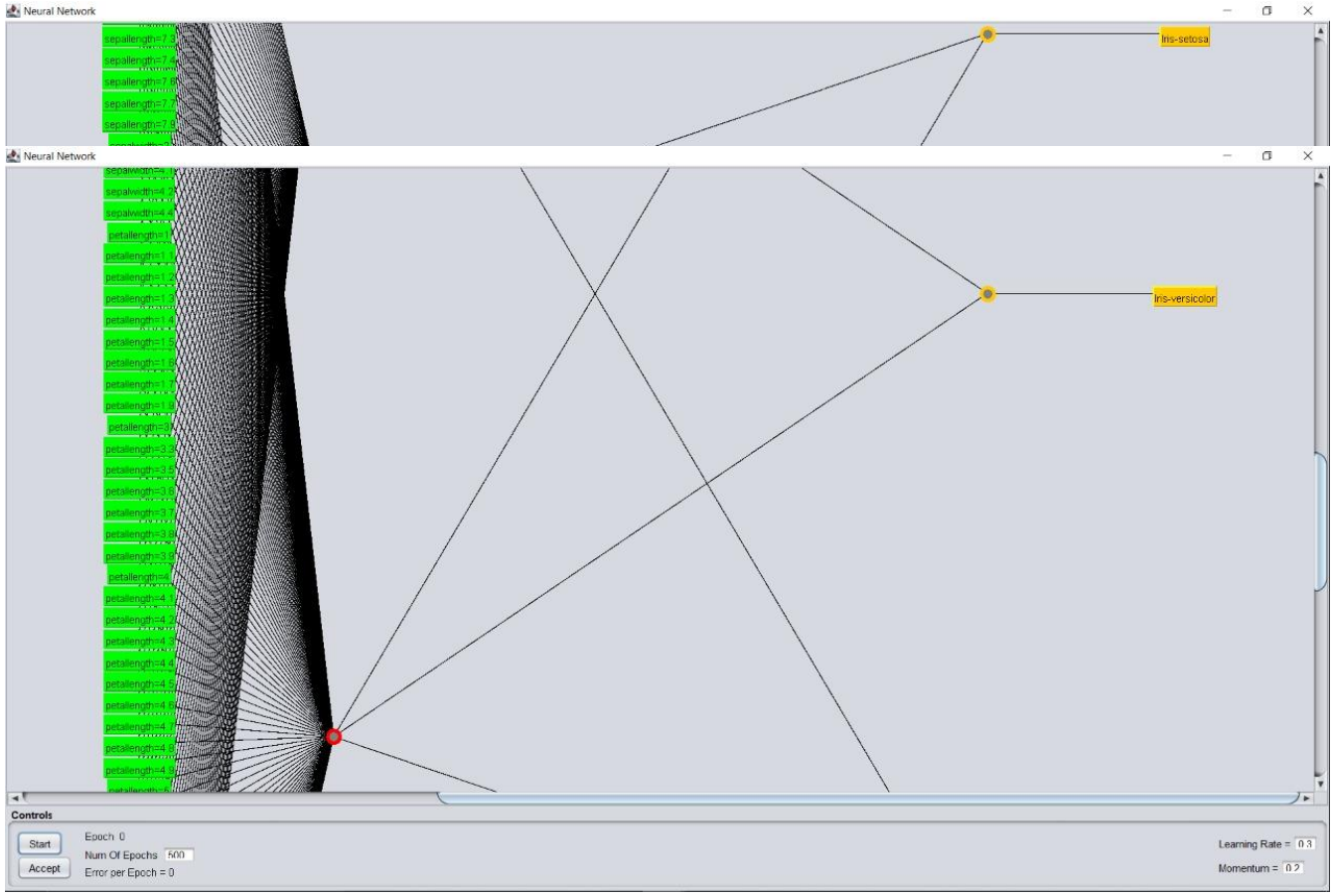
Goto classify ->functions->multilayer preceptron

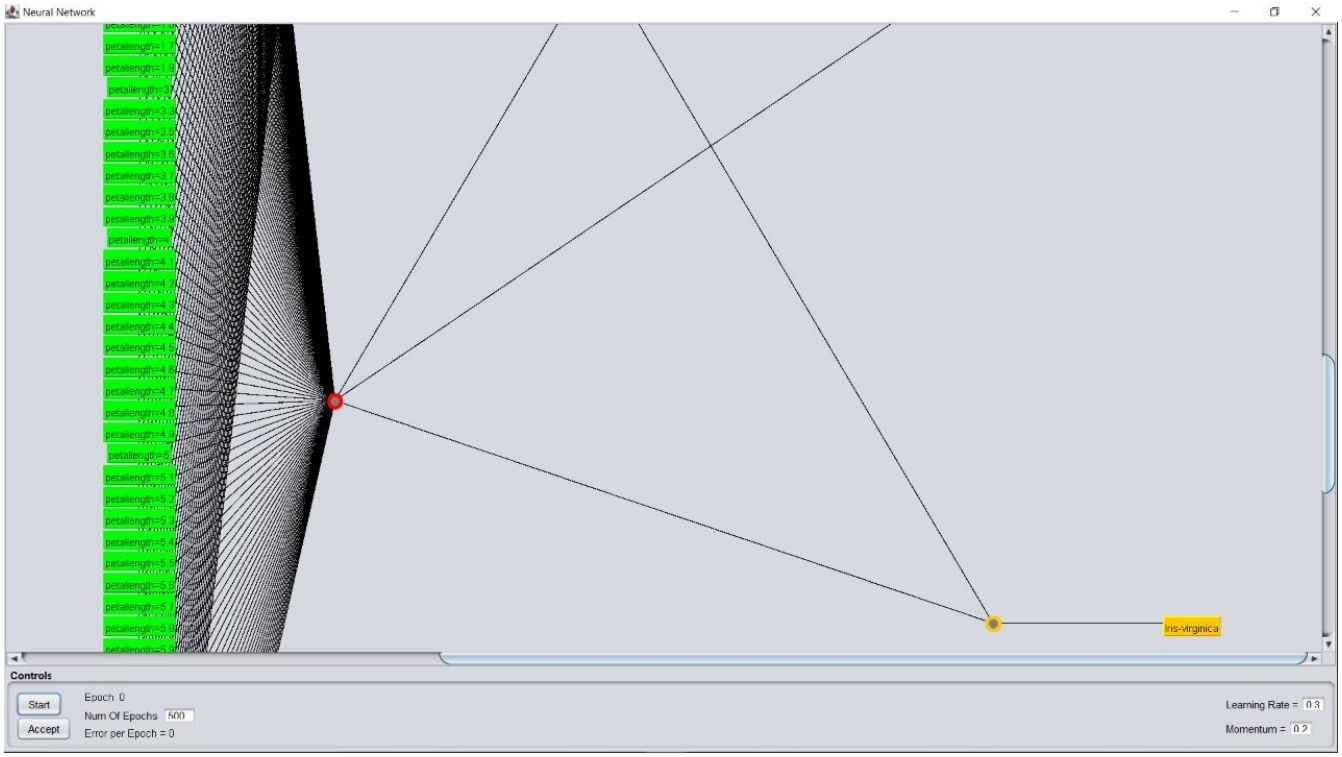
Click on properties tab and make GUI as true, hidden layers as 2

Click on start and accept and observe the output.

Correlation coefficient	0.7048
Mean absolute error	0.3038
Root mean squared error	0.3734
Relative absolute error	60.7581 %
Root relative squared error	74.6824 %
Total Number of Instances	4


Neural Network
Iris-versicolor
Controls
Start
Accept
Epoch 0
Num Of Epochs 500
Error per Epoch = 0
Learning Rate = 0.3
Momentum = 0.2

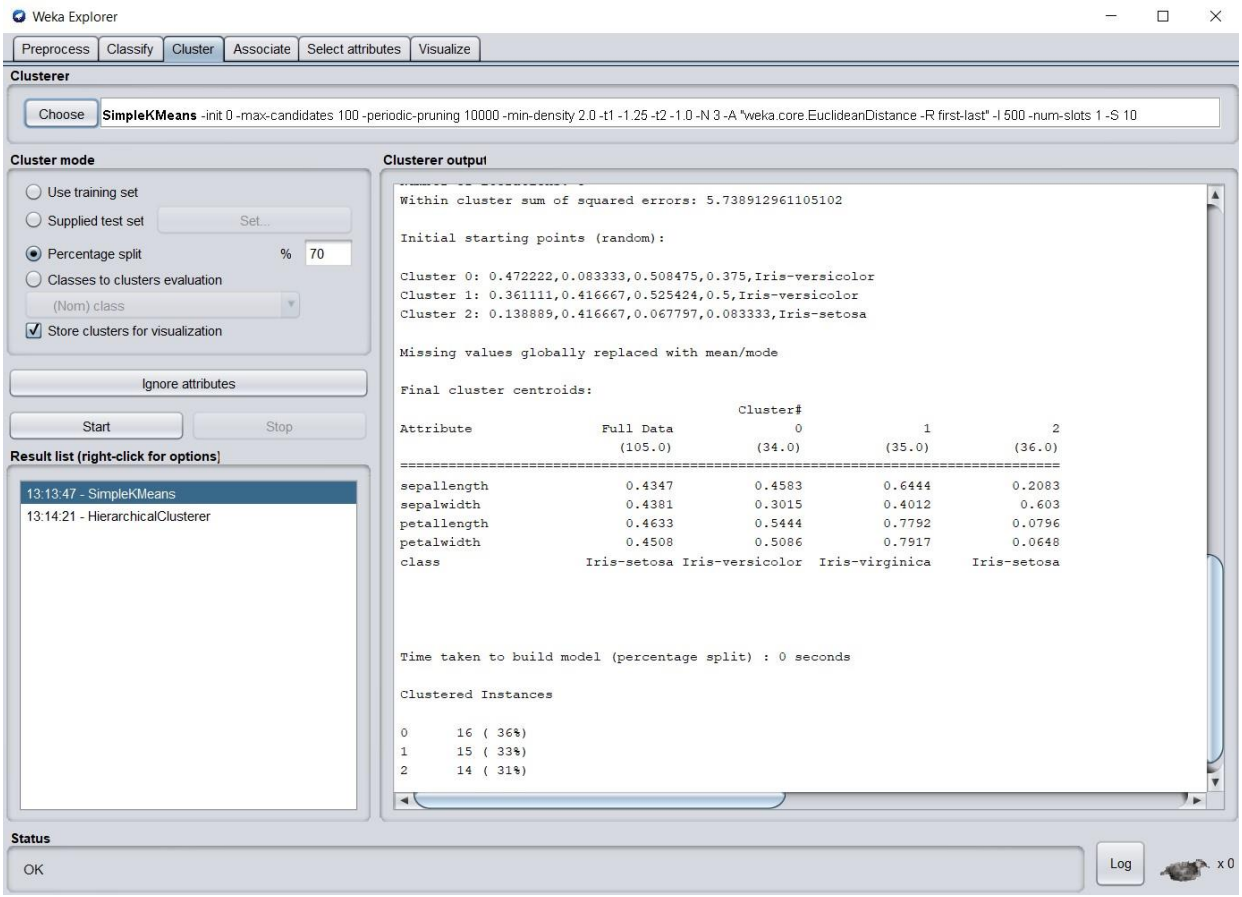

Neural Network
Iris-setosa
Neural Network
Iris-versicolor
Controls
Start
Accept
Epoch 0
Num Of Epochs 500
Error per Epoch = 0
Learning Rate = 0.3
Momentum = 0.2


Neural Network
Iris-virginica
Controls
Start
Accept
Epoch 0
Num Of Epochs 500
Error per Epoch = 0
Learning Rate = 0.3
Momentum = 0.2

4.CLUSTERING

4.1 SimpleKMeans

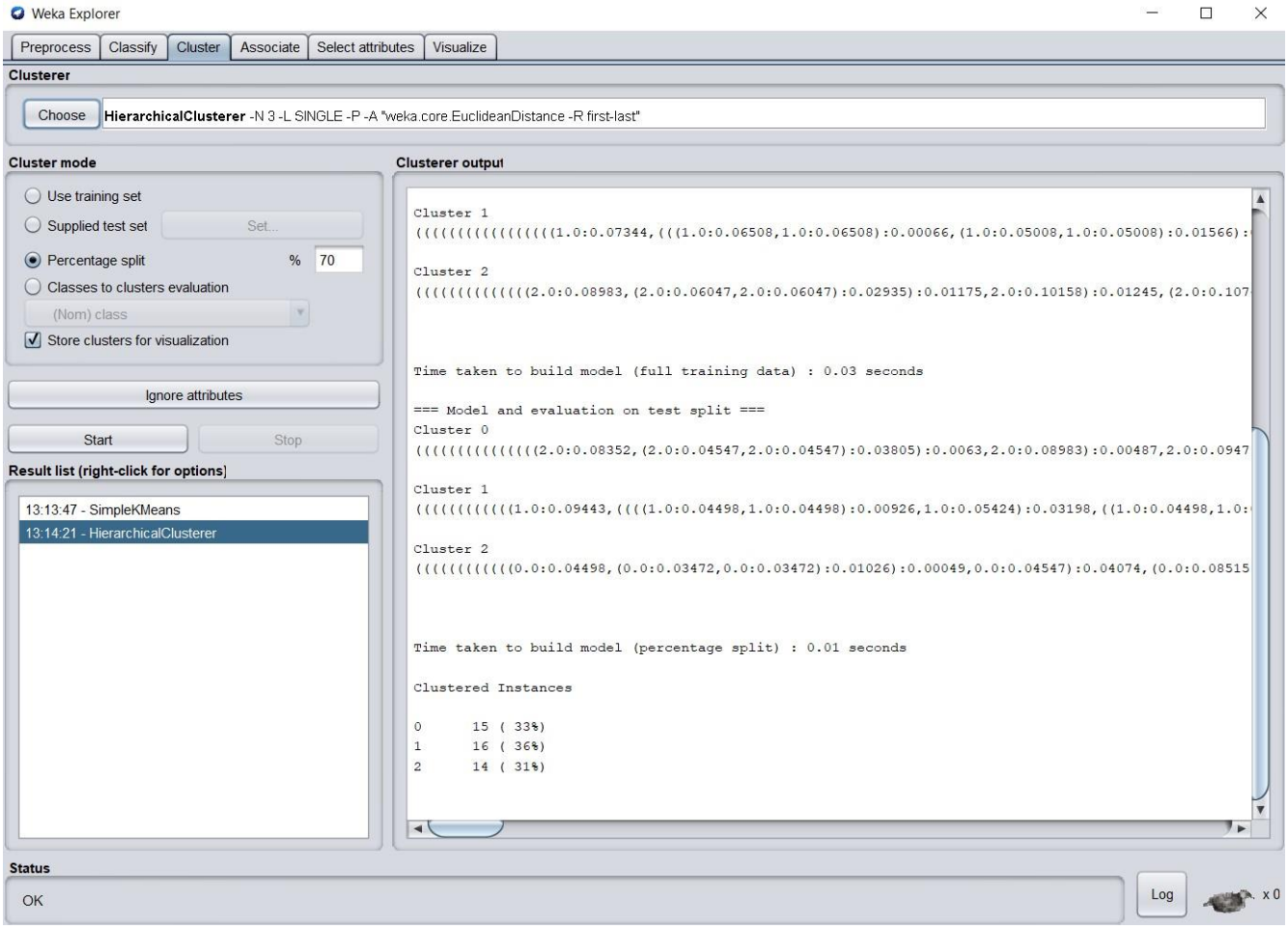
Apply weka.clusterers.SimpleKMeans
Set NumCluster as 3.
Cluster data using the k means algorithm. Can use either the Euclidean distance



(default) or the Manhattan distance.

4.2 Hierarchical Clutering

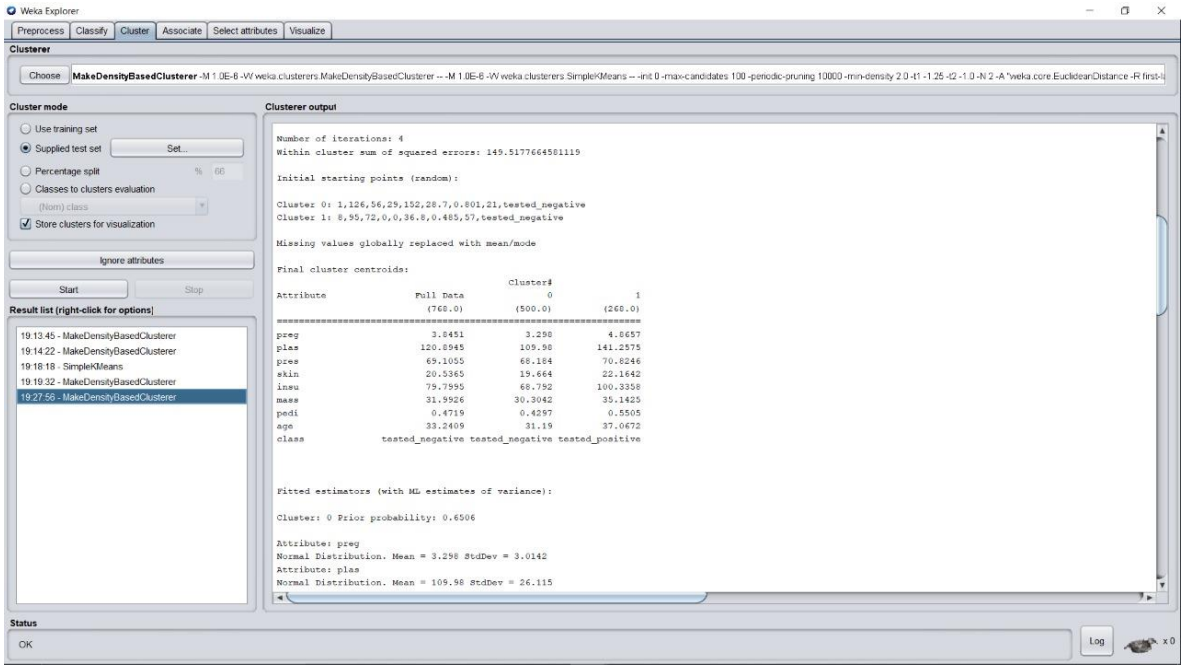
Apply weka.clusterers.HierarchicalClusterer
Hierarchical clustering class.
Implements a number of classic agglomorative (i.e. bottom up) hierarchical clustering methods
Set linkType as SINGLE.

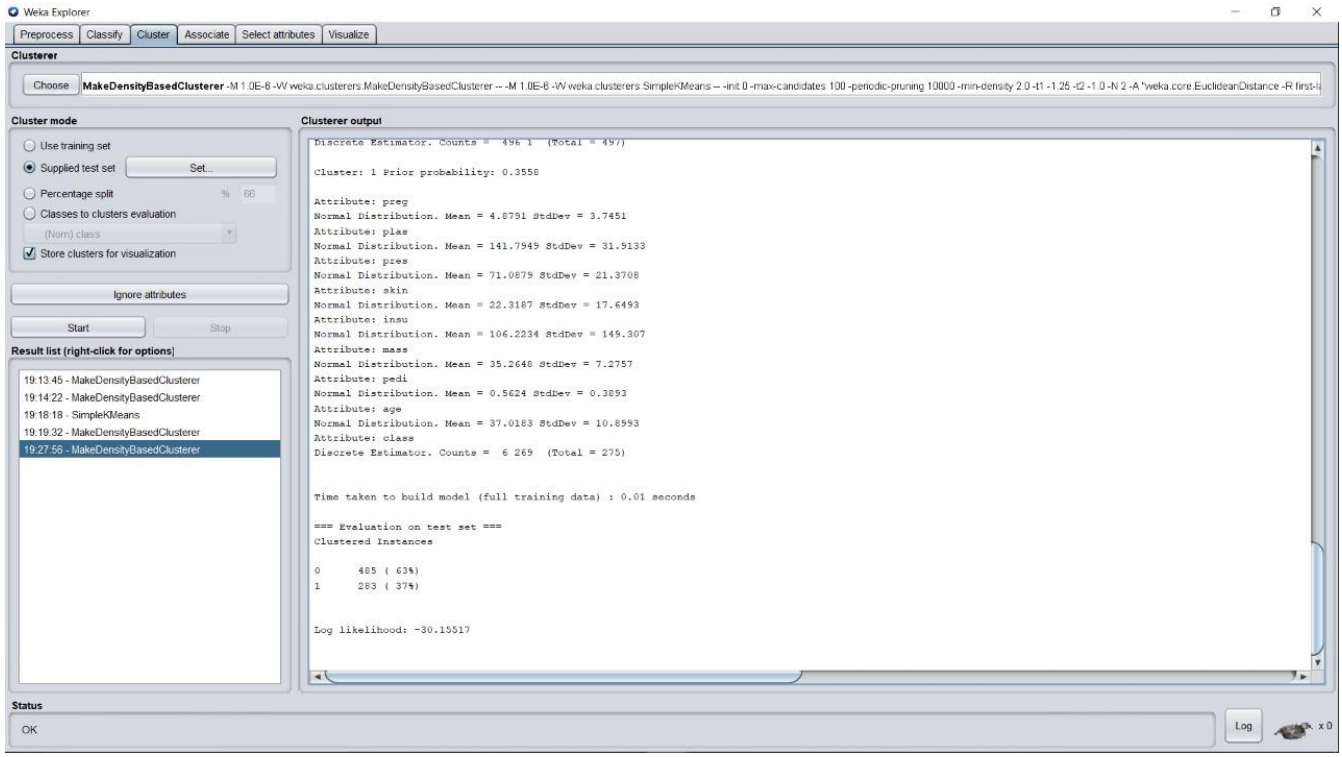


4.3 MakeDensityBasedClusterer

Apply weka.clusterers.MakeDensityBasedClusterer

Class for wrapping a Clusterer to make it return a distribution and density. Fits normal distributions and discrete distributions within each cluster produced by the wrapped clusterer. Supports the NumberOfClustersRequestable interface only if the wrapped

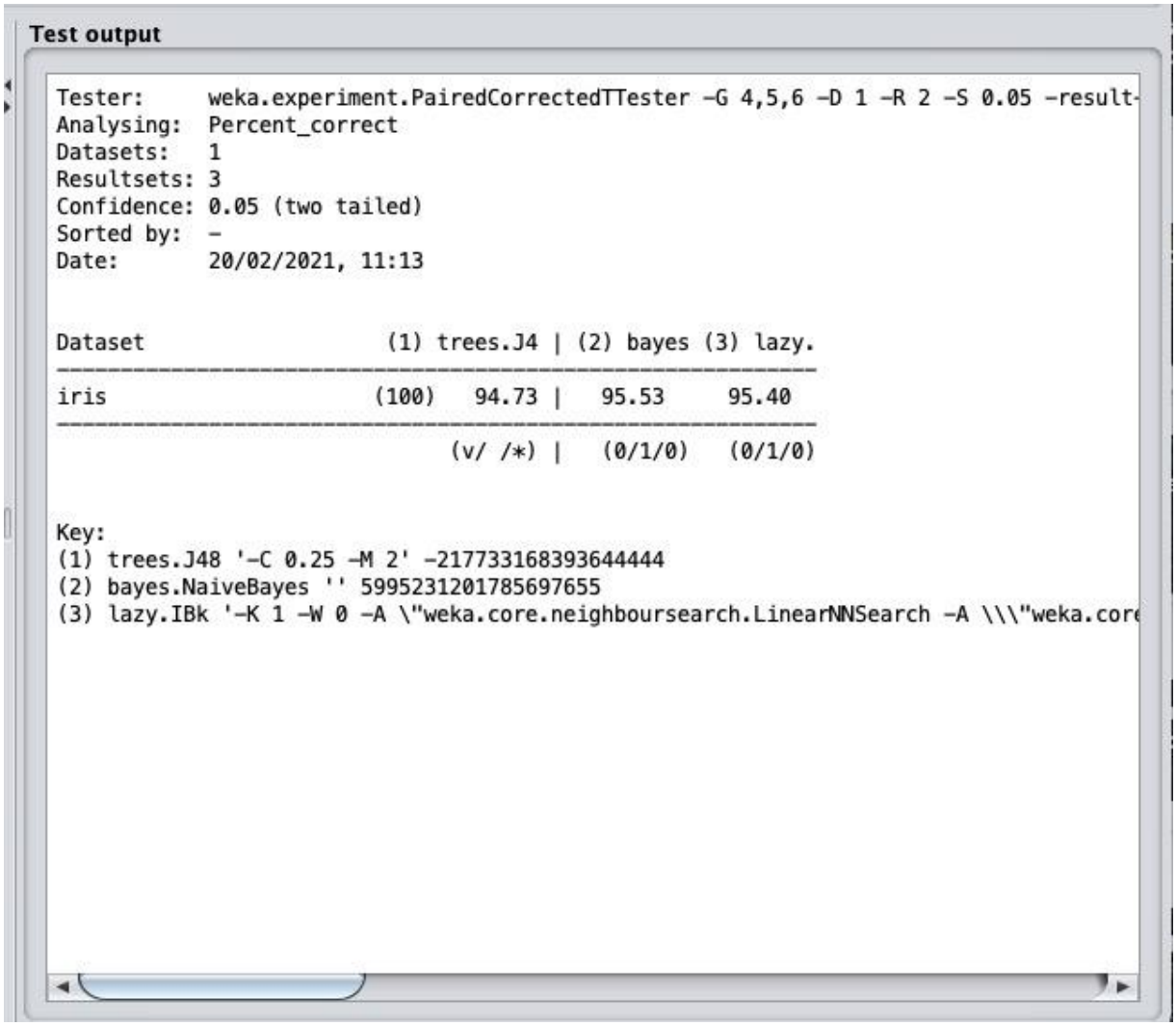




Clusterer does.

5.Compare the Performance of Algorithms in Experiment environment

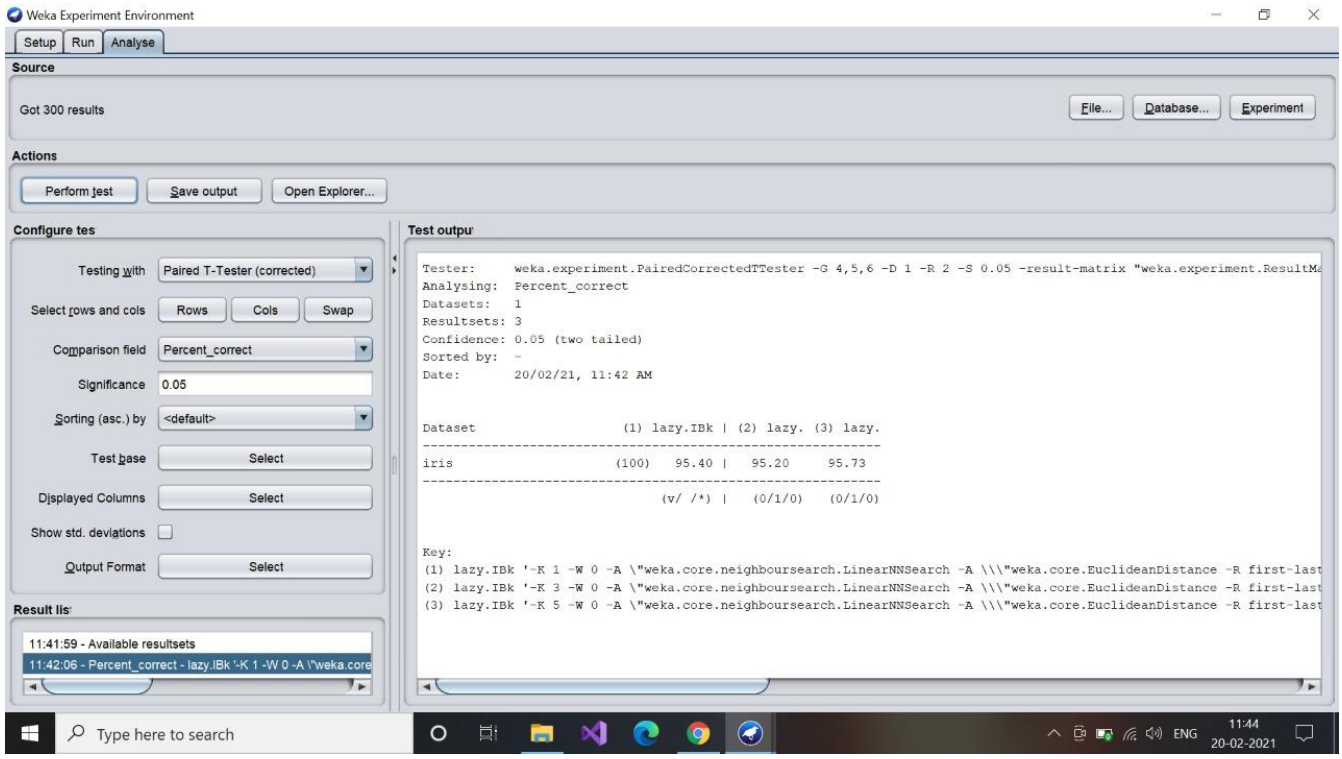
Weka provides a different tool specifically designed for comparing algorithms called the Weka Experiment Environment. The Weka Experiment Environment allows to design and execute controlled experiments with machine learning algorithms and then analyze the results.
Experiment Environment to compare the performance of machine learning algorithms.
1. Open the Weka Chooser GUI.
2. Click the Experimenter button to open the Weka Experiment Environment.
3. Click the New button.
4. Click the Add new... button in the Datasets pane and select iris.arff.
5. Click the Add new... button in the Algorithms panel and add trees.J48, bayes.NaiveBayes, functions.LibSVM and lazy.IBk
6. Click the Run tab and click the Start button.
7. Click the Analyse tab and click the Experiment button and then the Perform test button.
8. Save output.



6.Tune Algorithm Parameters

To get the most out of a machine learning algorithm you must tune the parameters of the method to your problem. You cannot know how to best do this beforehand, therefore you must try out lots of different parameters. The Weka Experiment Environment allows you to design controlled experiments to compare the results of different algorithm parameters and whether the differences are statistically significant.
This experiment compares the performance of k-Nearest Neighbors algorithm by varying its parameter value.

1. Open the Weka Chooser GUI.
2. Click the Experimenter button to open the Weka Experiment Environment
3. Click the New button.
4. Click the Add new... button in the Datasets pane and select iris.arff.
5. Click the Add new... button in the Algorithms pane and add 3 copes of the IBk algorithm.
6. Click each IBk algorithm in the list and click the Edit selected... button and change KNN to 1, 3, 5 for each of the 3 different algorithms.
7. Click the Run tab and click the Start button.
8. Click the Analyse tab and click the Experiment button and then the Perform test button.
9.Save output.



1. Five Number Summary

jupyter DMlab Last Checkpoint: 4 minutes ago (autosaved) Logout

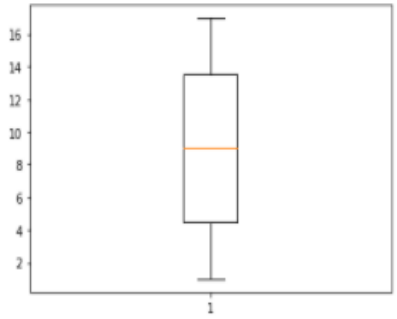
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Five Number Summary

In [1]:
```
import matplotlib.pyplot as plt
from numpy import percentile
import numpy as np
```

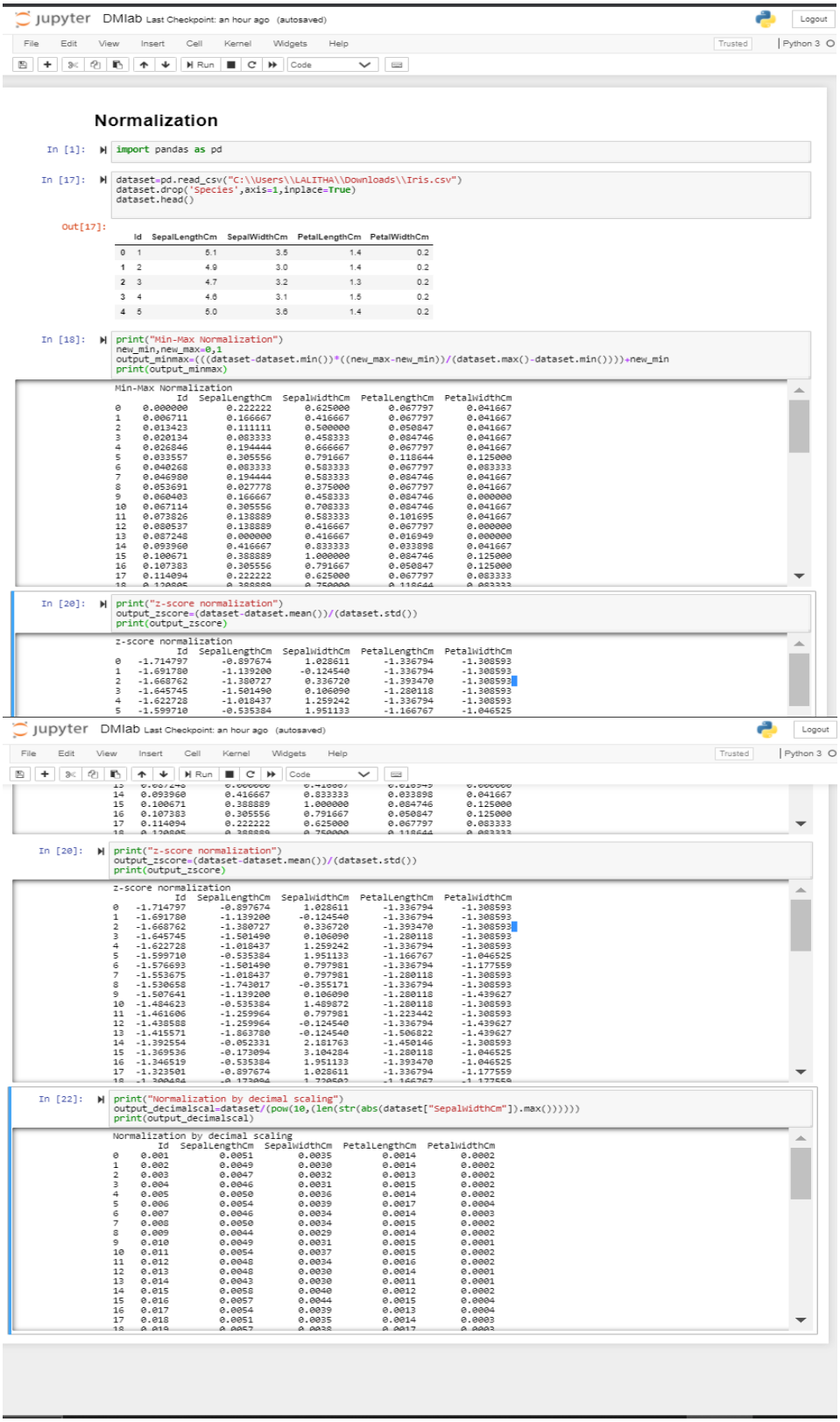
In [2]:
```
dataip = [1,2,15,3,6,17,8,16,8,3,10,12,16,12,9]
data=np.array(dataip)
plt.boxplot(data)
plt.show()
```



In [3]:
```
# calculate quartiles
quartiles = percentile(data, [25, 50, 75])
# calculate min/max
data_min, data_max = data.min(), data.max()
# print 5-number summary
print('Min: %.3f'% data_min)
print('Q1: %.3f'% quartiles[0])
print('Median: %.3f'% quartiles[1])
print('Q3: %.3f'% quartiles[2])
print('Max: %.3f'% data_max)
```

```
Min: 1.000
Q1: 4.500
Median: 9.000
Q3: 13.500
Max: 17.000
```

2.Normalisation



3.Binning

In [1]:
```
import numpy as np
```

In [2]:
```
price = np.array([5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204, 215])
dataset=np.sort(price)
nob=3

print(dataset.size)
```

```
12
```

In [3]:
```
binsize=(int)(dataset.size/nob)
print(binsize)
print('..........Equal height binning...')
print(dataset[0:binsize],dataset[binsize:2*binsize],dataset[2*binsize:3*2*binsize])
```

```
4
..........Equal height binning...
[ 5 10 11 13] [15 35 50 55] [ 72  92 204 215]
```

In [5]:
```
print('...........Equal width binning...')

width=int((dataset.max()-dataset.min())/nob)



bin1=[]
bin2=[]
bin3=[]
for  i in range(len(dataset)):
    if dataset[i]>=dataset.min() and dataset[i]<=(width+dataset.min()):
        bin1.append(dataset[i])
    elif dataset[i]>(width+dataset.min()) and dataset[i]<=2*(width+dataset.min()):
            bin2.append(dataset[i])

    else:
            bin3.append(dataset[i])


print(bin1)
print(bin2)
print(bin3)
```

```
...........Equal width binning...
[5, 10, 11, 13, 15, 35, 50, 55, 72]
[92]
[204, 215]
```

In [ ]:
```

```

4.Distance Metrics

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Distance Metrics

In [1]:

```
import math
from decimal import Decimal
"""Five similarity measures function """
def euclidean_distance(x,y):

    """return euclidean distance between two lists """
    return math.sqrt(sum(pow(a-b,2) for a, b in zip(x, y)))

def manhattan_distance(x,y):

     """return manhattan distance between two lists """
     return sum(abs(a-b) for a,b in zip(x,y))

def minkowski_distance(x,y,p_value):

    """return minkowski distance between two lists """
    return nth_root(sum(pow(abs(a-b),p_value) for a,b in zip(x, y)),
           p_value)

def nth_root(value, n_root):

     """returns the n_root of an value """
     root_value = 1/float(n_root)
     return round (Decimal(value) ** Decimal(root_value),3)

def cosine_similarity(x,y):

     """return cosine similarity between two lists """
     numerator = sum(a*b for a,b in zip(x,y))
     denominator =square_rooted(x)*square_rooted(y)
     return round(numerator/float(denominator),3)

def square_rooted(x):
     """return 3 rounded square rooted value """
     return round(math.sqrt(sum([a*a for a in x])),3)

def jaccard_similarity(x,y):

     """returns the jaccard similarity between two lists """
     intersection_cardinality = len(set.intersection(*[set(x), set(y)]))
     union_cardinality = len(set.union(*[set(x), set(y)]))
     return intersection_cardinality/float(union_cardinality)
```

In [2]:

```
x=[22, 1, 42, 10]
y=[20, 0, 36, 8]

print(euclidean_distance(x,y))
print(manhattan_distance(x,y))
print(minkowski_distance(x,y,3))
print(cosine_similarity(x,y))
print(jaccard_similarity(x,y))
```

```
6.708203932499369
11
6.153
0.999
0.0
```

6.Apriori

Apriori

In [3]:
```
import itertools as it
```

In [4]:
```
data=[[1,2,3],[2,3,5],[1,2,4],[3,4]]
l=[]
u=[]
for x in data:
    print(x)
c1=[1,2,3,4,5]
for i in range(1,len(c1)):
    y=it.combinations(c1,i)
    for z in y:
        count=0
        for x in data:
         if (set(z).issubset(x)):
          count=count+1
         if(count>=2):
          l.append(z)
          break
print(l)
```

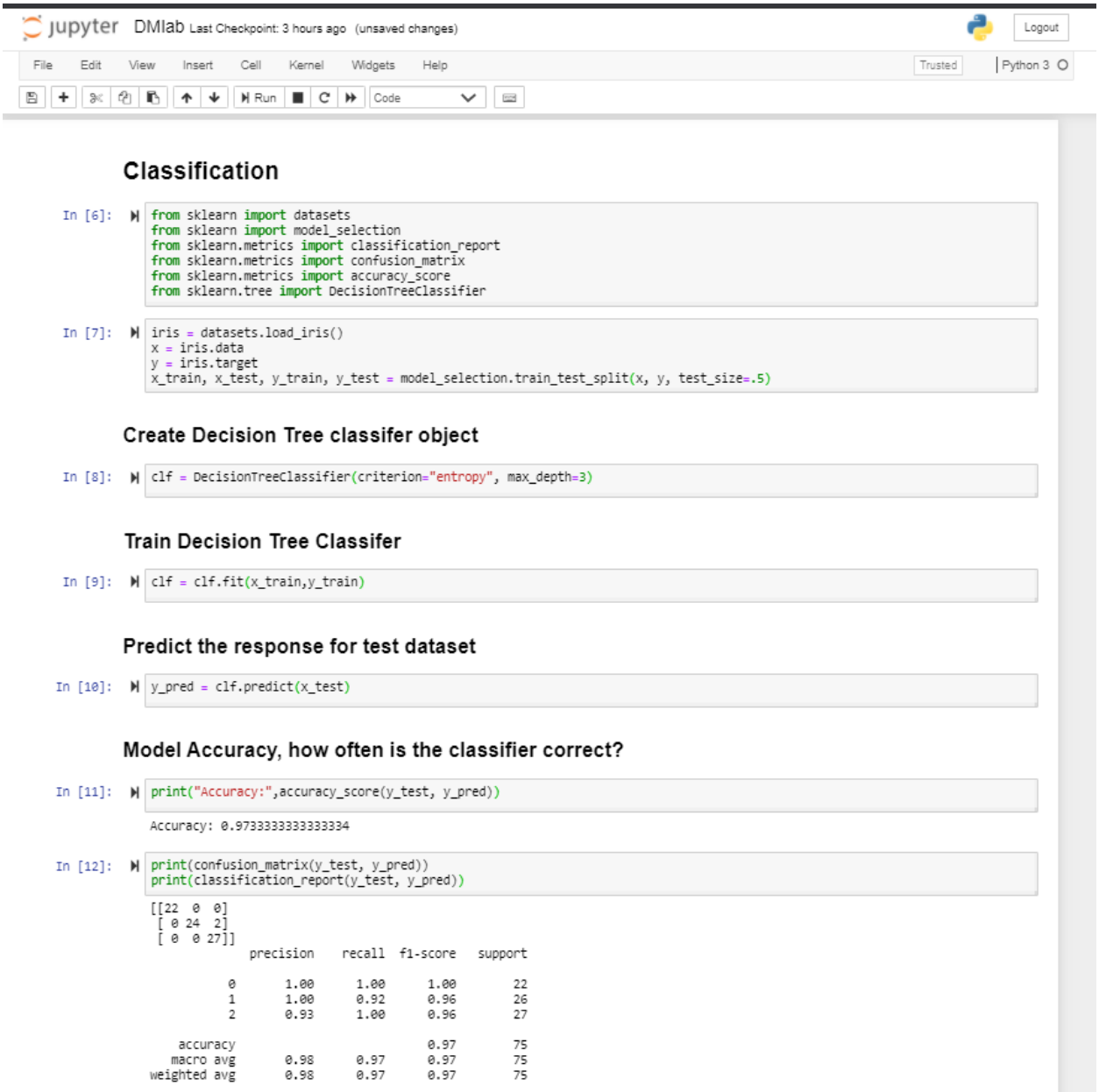
```
[1, 2, 3]
[2, 3, 5]
[1, 2, 4]
[3, 4]
[(1,), (2,), (3,), (4,), (1, 2), (2, 3)]
```

In [5]:
```
l1=[3]
r1=[1,2]
count1=0.0
count2=0.0
for x in data:
    if(set(l1).issubset(x) and set(r1).issubset(x)):
        count1=count1+1
print(count1)
for y in data:
    if(set(l1).issubset(y)):
        count2=count2+1
print(count2)
conf=(count1/count2)*100
print(conf)
```

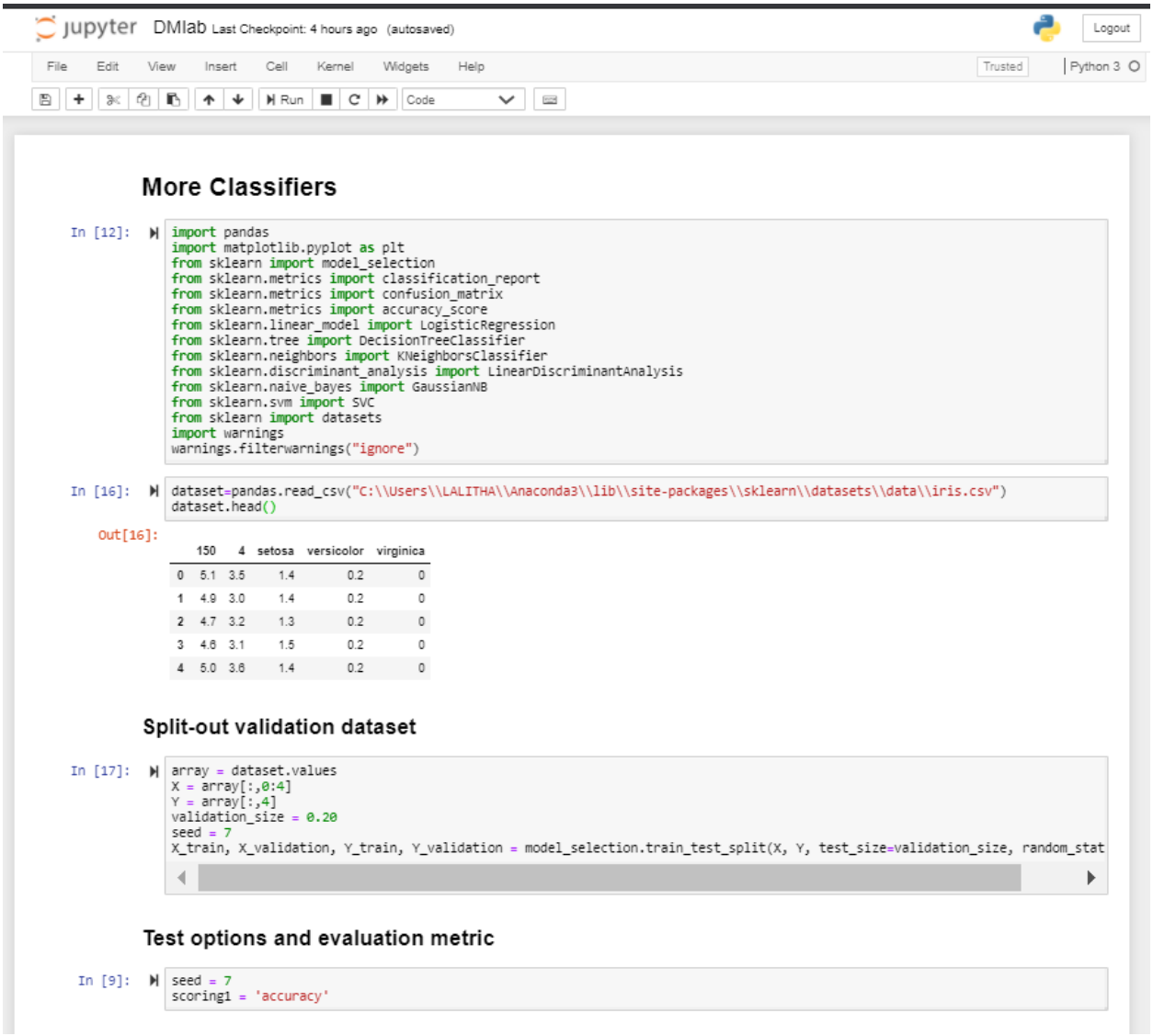
```
1.0
3.0
33.33333333333333
```

In [ ]:
```

```

7.Classification



8.More Classifiers



jupyter DMlab Last Checkpoint: 4 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

Spot Check Algorithms

In [7]:
```
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
```

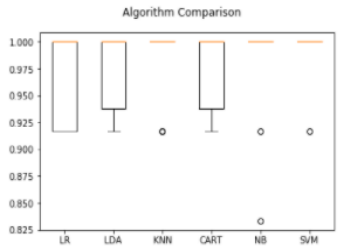
Evaluate each model in turn

In [18]:
```
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, scoring=scoring1, cv=kfold)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)"% (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.966667 (0.040825)
LDA: 0.975000 (0.038188)
KNN: 0.983333 (0.033333)
CART: 0.975000 (0.038188)
NB: 0.975000 (0.053359)
SVM: 0.991667 (0.025000)
```

Compare Algorithms

In [19]:
```
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



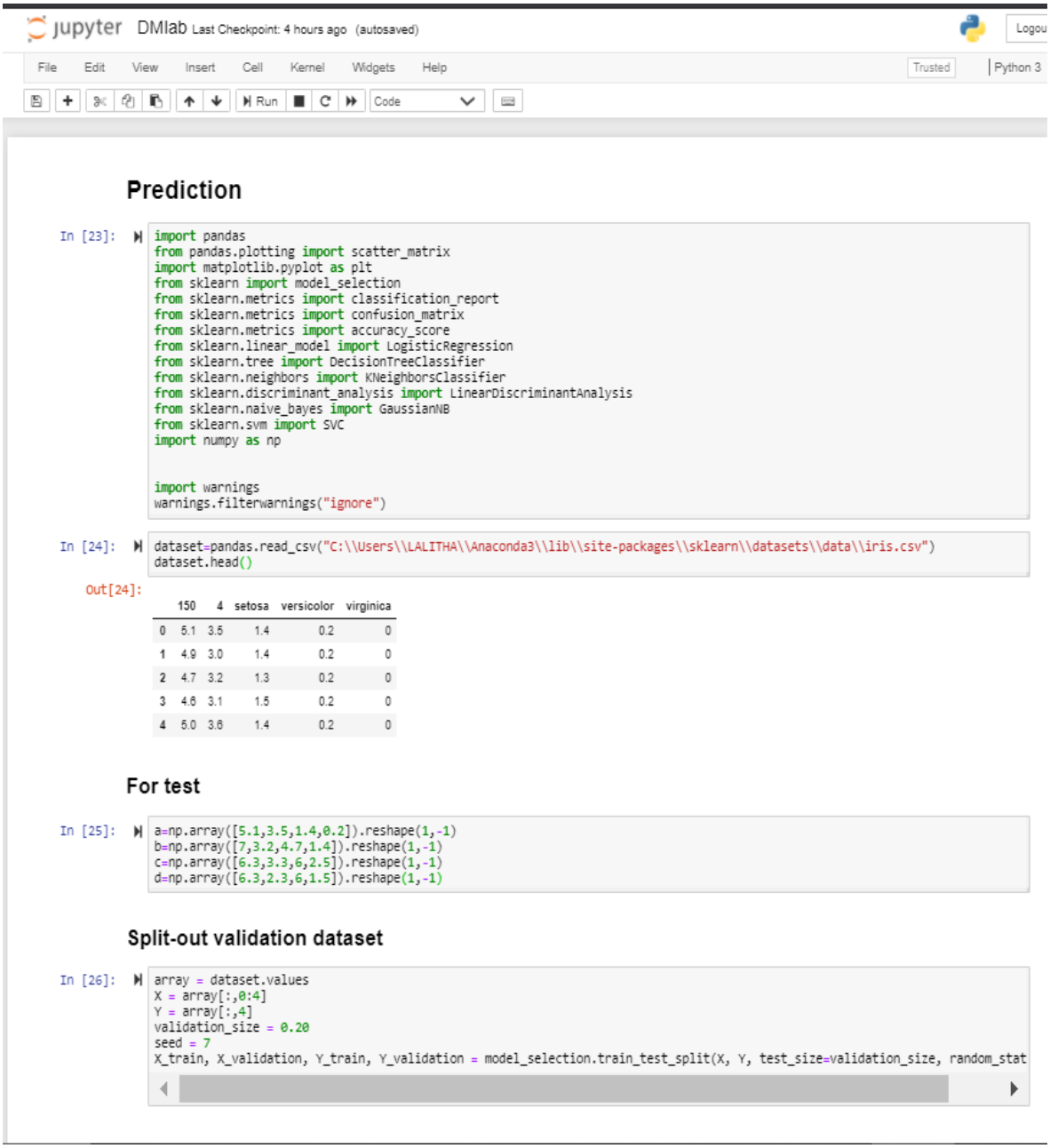
Make predictions on validation dataset

In [20]:
```
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
predictions = knn.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.9
[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00         7
         1.0       0.85      0.92      0.88        12
         2.0       0.90      0.82      0.86        11

    accuracy                           0.90        30
   macro avg       0.92      0.91      0.91        30
weighted avg       0.90      0.90      0.90        30
```

In [ ]:

9.Prediction



jupyter DMlab Last Checkpoint: 4 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

Split-out validation dataset

In [26]:
```
array = dataset.values
X = array[:,0:4]
Y = array[:,4]
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y, test_size=validation_size, random_stat
```

Test options and evaluation metric

In [27]:
```
seed = 7
scoring = 'accuracy'
```

Spot Check Algorithms

In [28]:
```
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
```

Evaluate each model in turn

In [29]:
```
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, scoring=scoring1, cv=kfold)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)"% (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.966667 (0.040825)
LDA: 0.975000 (0.038188)
KNN: 0.983333 (0.033333)
CART: 0.975000 (0.038188)
NB: 0.975000 (0.053359)
SVM: 0.991667 (0.025000)
```

Make predictions on validation dataset

In [30]:
```
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)

print("prediction of new instances")
predictionsa = knn.predict(a)
```

Make predictions on validation dataset

In [30]:
```
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)

print("prediction of new instances")
predictionsa = knn.predict(a)
print(predictionsa)

predictionsb = knn.predict(b)
print(predictionsb)


predictionsc = knn.predict(c)
print(predictionsc)


predictionsd = knn.predict(d)
print(predictionsd)
```

```
prediction of new instances
[0.]
[1.]
[2.]
[2.]
```

In [31]:
```
predictions = knn.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

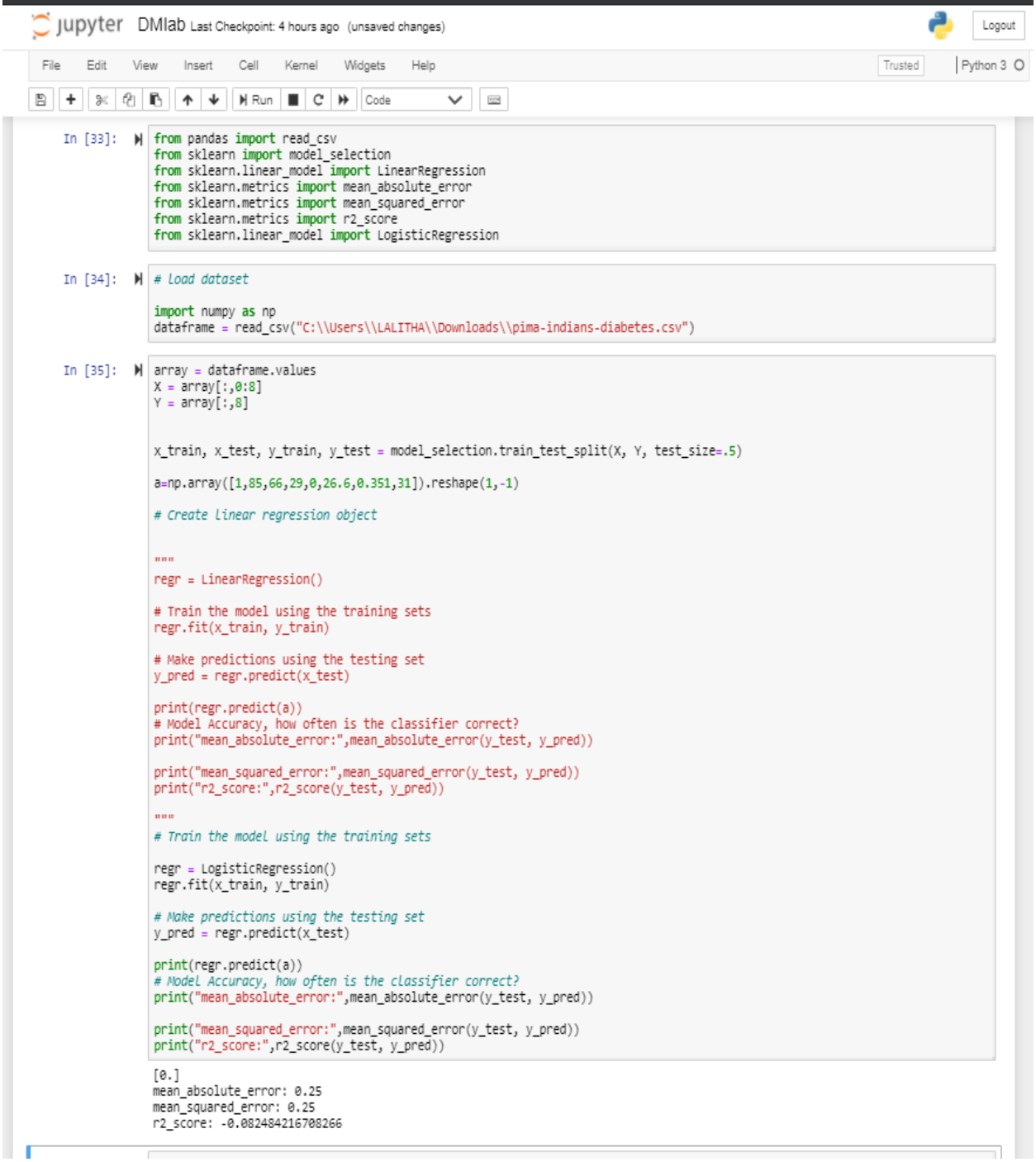
```
0.9
[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00         7
         1.0       0.85      0.92      0.88        12
         2.0       0.90      0.82      0.86        11

    accuracy                           0.90        30
   macro avg       0.92      0.91      0.91        30
weighted avg       0.90      0.90      0.90        30
```

In [32]:
```
svc = SVC()
svc.fit(X_train, Y_train)
predictions = svc.predict(a)
print(predictions)
```

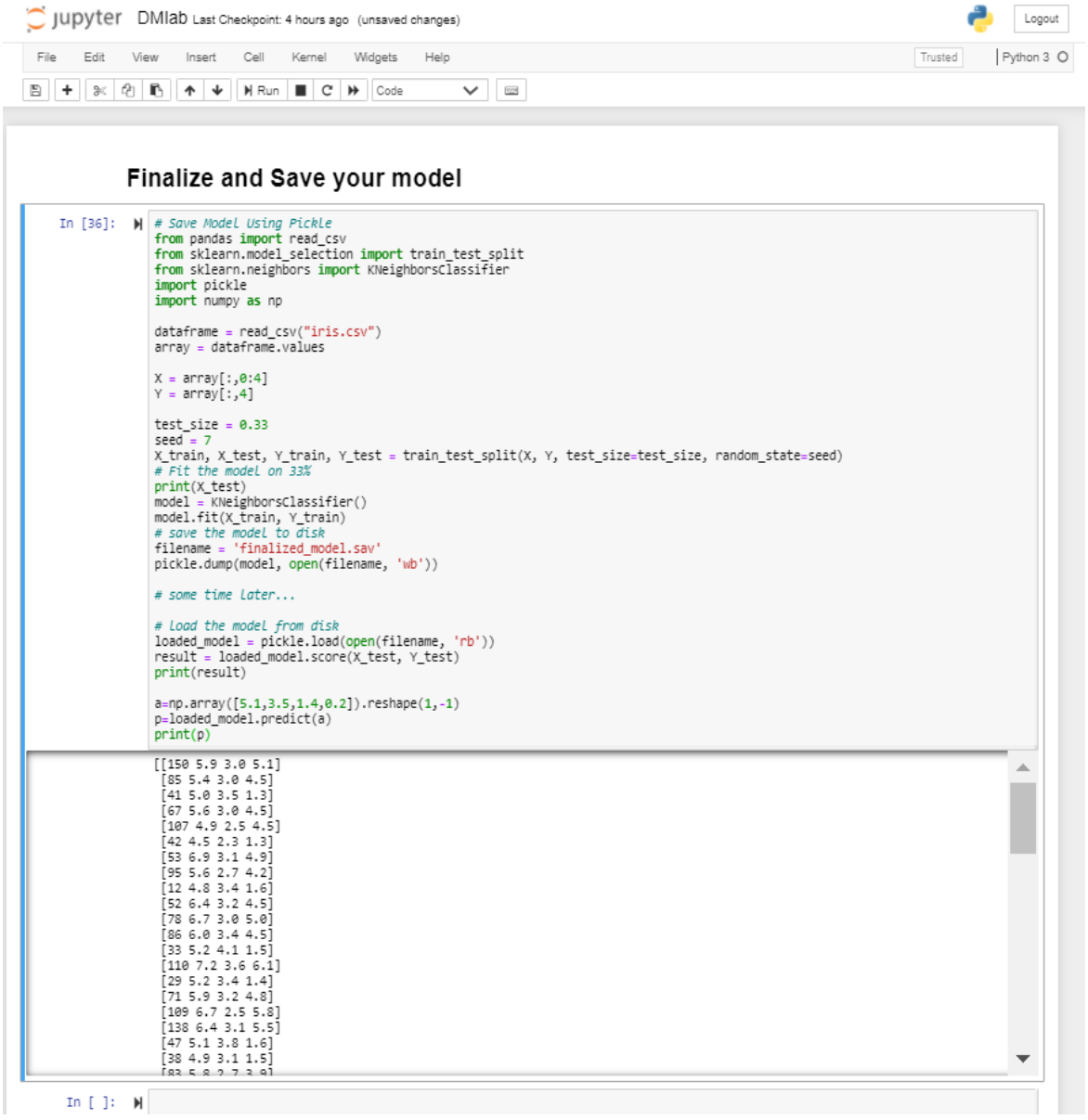
```
[0.]
```

In [ ]:
```
```

10. Regression



11. Finalize and Save your model



12.Clustering

jupyter DMlab Last Checkpoint: 4 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [37]:

```
from sklearn import cluster
import pandas as pd

dataset=pd.read_csv("C:\\Users\\LALITHA\\Downloads\\pima-indians-diabetes.csv")
km = cluster.KMeans(n_clusters=3).fit(dataset)
print(km.cluster_centers_)
print(km)


from sklearn.cluster import AgglomerativeClustering
ac = AgglomerativeClustering().fit(dataset)
print(ac.labels_)
print(type(ac.labels_))

ac.labels_.tofile("D:/output.csv")


from sklearn.cluster import DBSCAN
dbs = DBSCAN(eps=3, min_samples=3).fit(dataset)
print(dbs)
print(dbs.components_)
```

```
[[3.97773279e+00 1.13939271e+02 6.77631579e+01 1.49574899e+01
  1.44291498e+01 3.07997976e+01 4.31536437e-01 3.37267206e+01
  2.97570850e-01]
 [4.02631579e+00 1.58447368e+02 7.20000000e+01 3.22631579e+01
  4.41289474e+02 3.51078947e+01 5.69210526e-01 3.47631579e+01
  5.78947368e-01]
 [3.52765957e+00 1.29327660e+02 7.14468085e+01 3.03063830e+01
  1.59102128e+02 3.39893617e+01 5.40276596e-01 3.19021277e+01
  4.17021277e-01]]
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
[1 1 1 0 1 1 1 0 1 1 1 1 0 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 1
 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 0 1 0 1 1 0 0
 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1
 1 0 1 0 0 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1
 0 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 1 1
 0 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 0
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 1 0 0
 0 0 1 1 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 0 1 1 1 1 1 0 0
 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1
 1 0 1 0 0 1 0 0 1 1 1 1 0 1 1 1 0 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 1
 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1
 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 1 1 1 1 1 1 1 1 0
 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1
 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 1 0 0 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1
 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 0 1 0 0 1 0 1 0 1 0 1 1 1
 1 1 1 0 1 1 0 1 1 0 1 0 0 1 0 1 1 0 0 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0
 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1]
<class 'numpy.ndarray'>
DBSCAN(algorithm='auto', eps=3, leaf_size=30, metric='euclidean',
       metric_params=None, min_samples=3, n_jobs=None, p=None)
[]
```

In [ ]:

A case study

On

Prediction of Structure in Ionosphere

By

Sanikommu Narendra (2451-17-733-062)

Bobbala Sanjay Yadav (2451-17-733-074)

Papaganti Prasanna Kuma (2451-17-733-096)

Department of Computer Science and Engineering

M.V.S.R. ENGINEERING COLLEGE

(Affiliated to Osmania University & Recognized by AICTE)

Nadergul, Saroor Nagar Mandal, Hyderabad – 501 510

2020-21.

Objective

In this case study, we are predicting whether or not there is structure in the ionosphere or not

Data Set Information

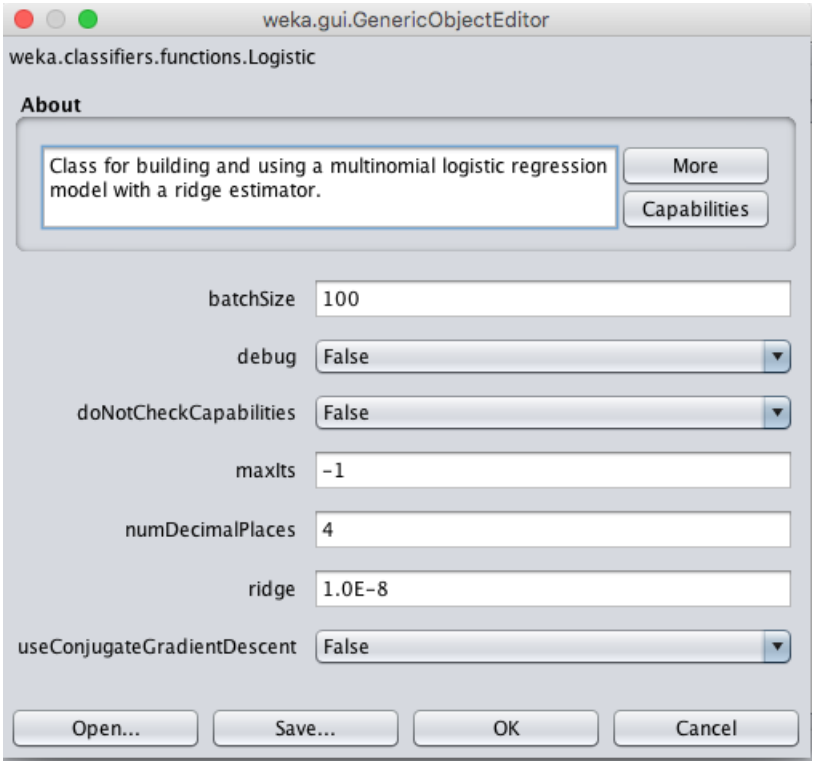
Data Set Characteristics:	Multivariate	**Number of Instances:**	351
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	34
Associated Tasks:	Classification	**Missing Values?**	No

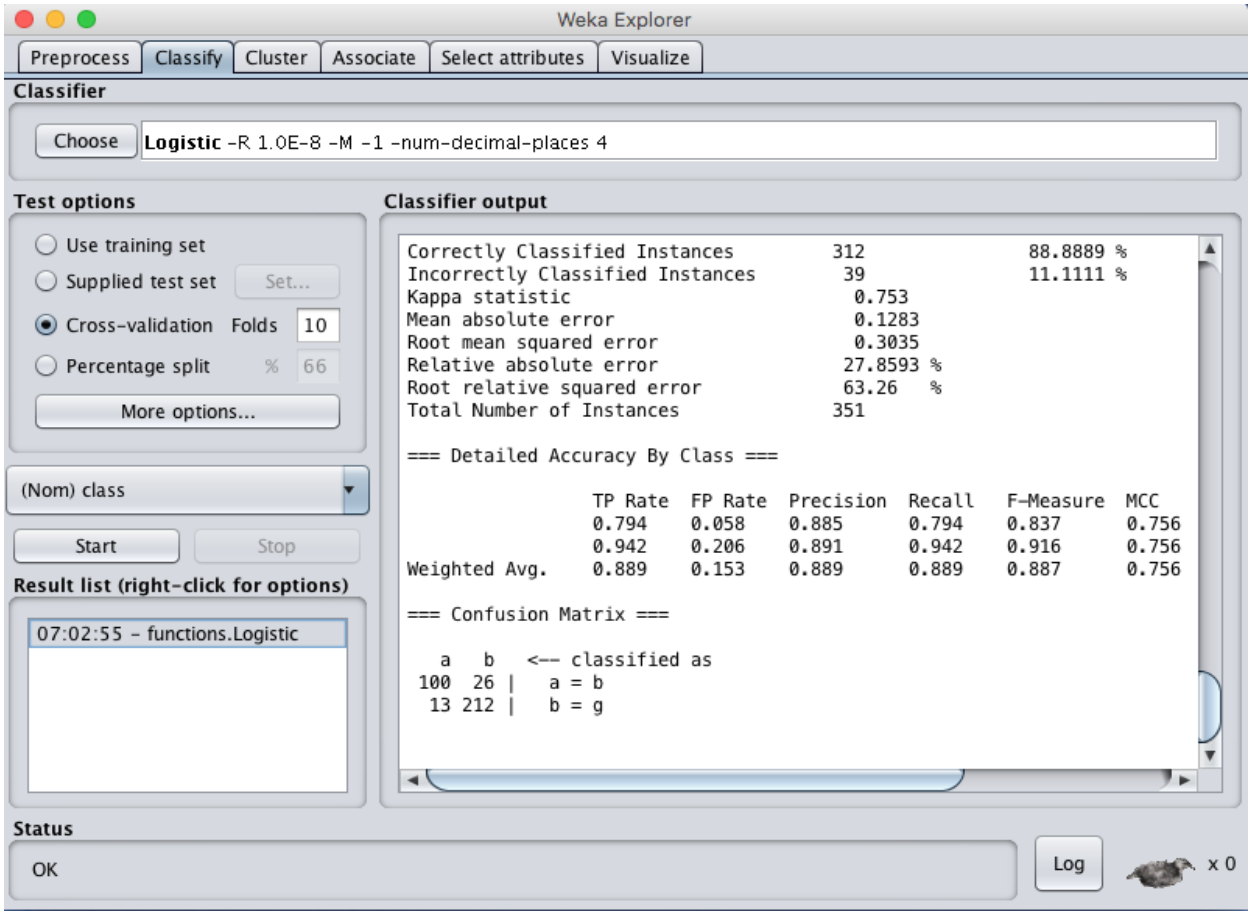
Attribute Information:

-- All 34 are continuous.

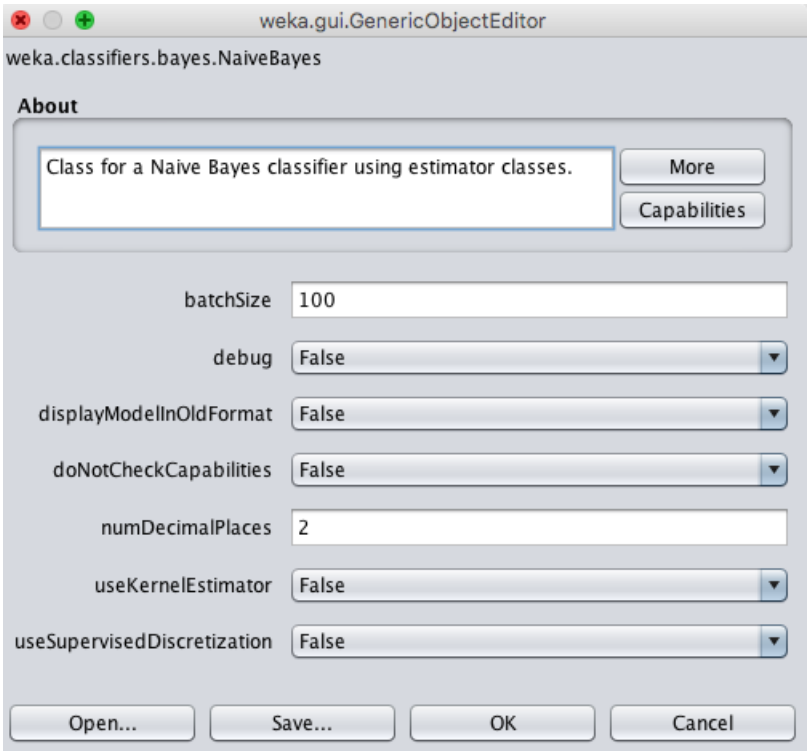
-- The 35th attribute is either "good" or "bad".

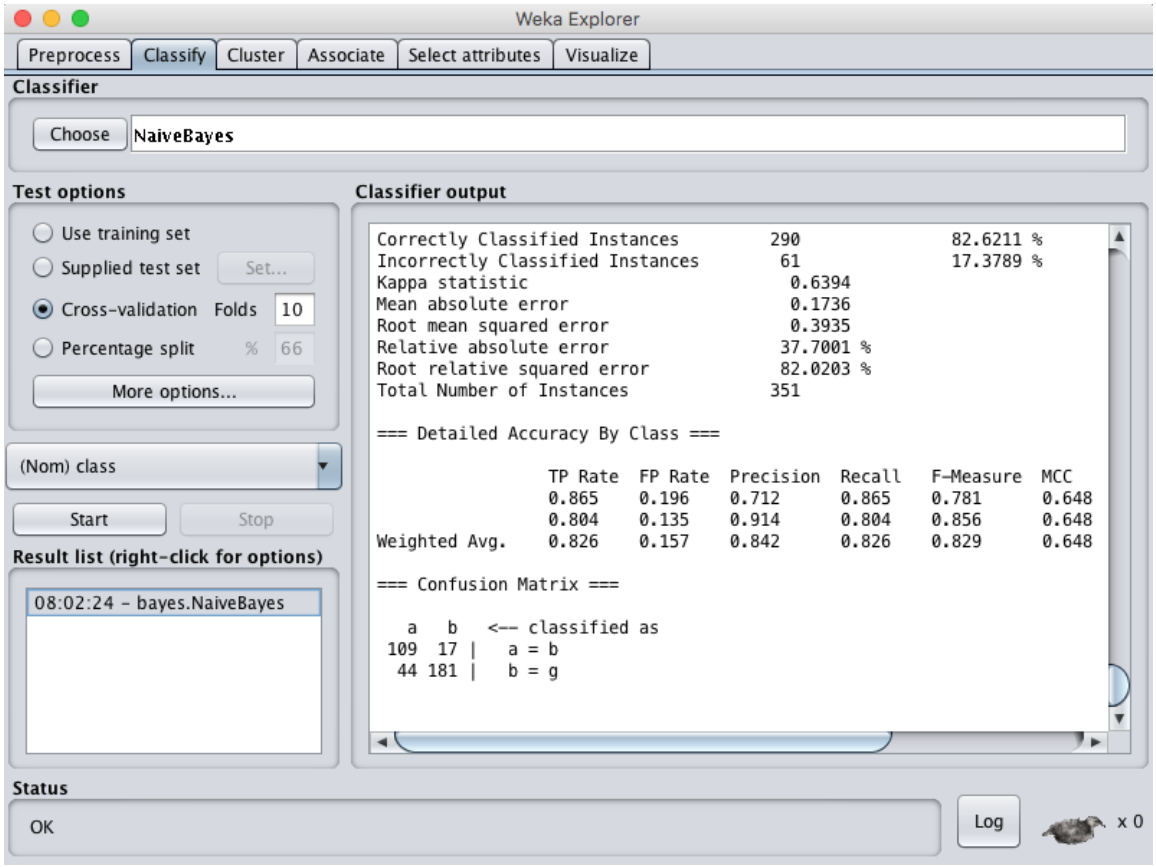
Logistic Regression



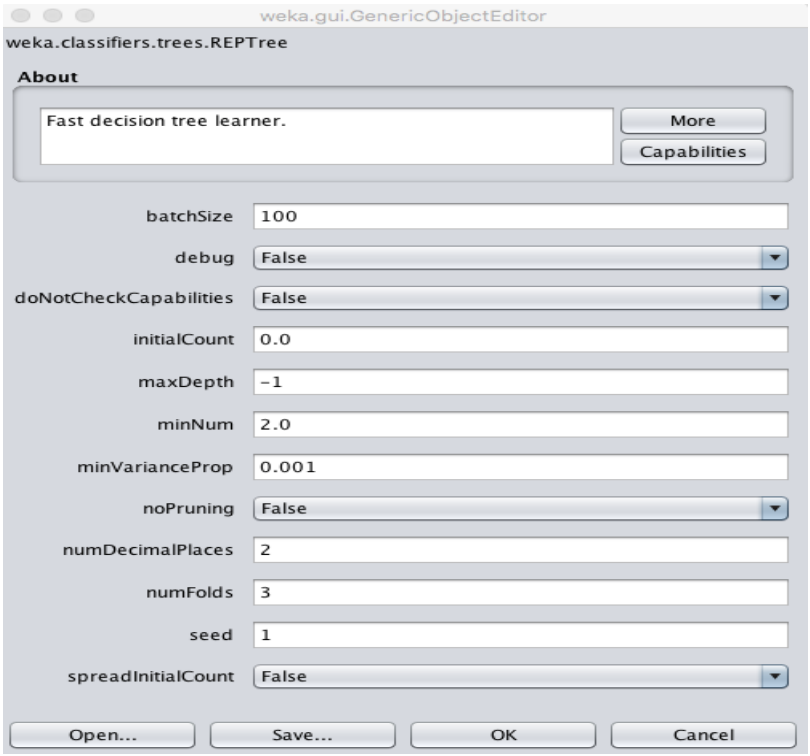


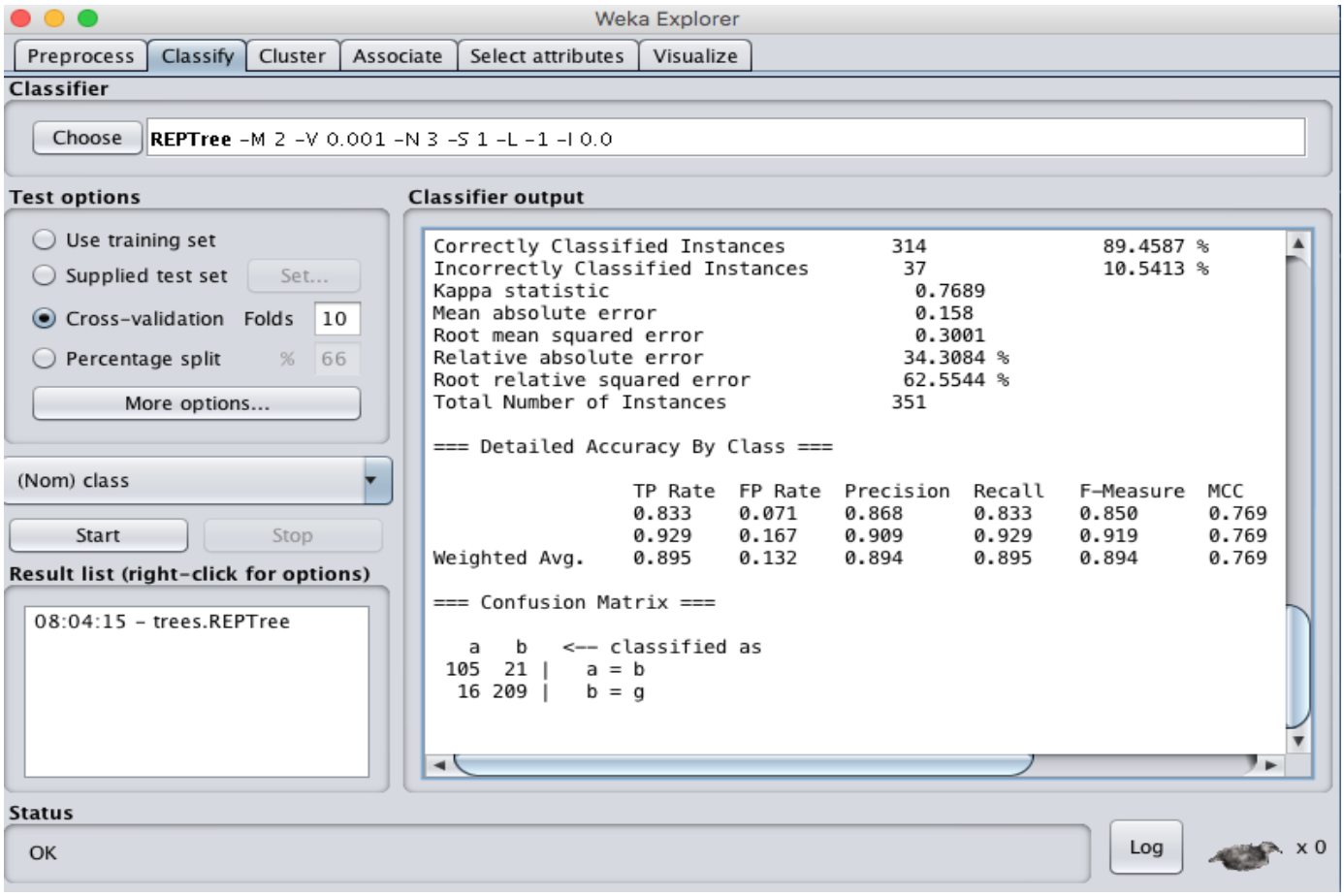
Naive Bayes


weka.gui.GenericObjectEditor
weka.classifiers.bayes.NaiveBayes
About
Class for a Naive Bayes classifier using estimator classes.
More
Capabilities
batchSize
100
debug
False
displayModelInOldFormat
False
doNotCheckCapabilities
False
numDecimalPlaces
2
useKernelEstimator
False
useSupervisedDiscretization
False
Open...
Save...
OK
Cancel

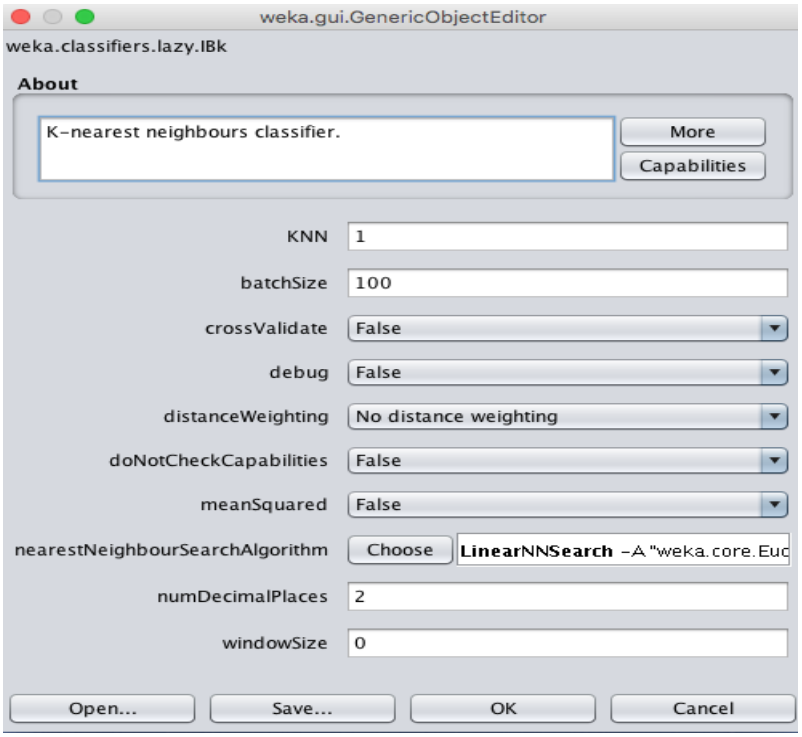

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
NaiveBayes
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
08:02:24 - bayes.NaiveBayes
Classifier output
Correctly Classified Instances 290 82.6211 %
Incorrectly Classified Instances 61 17.3789 %
Kappa statistic 0.6394
Mean absolute error 0.1736
Root mean squared error 0.3935
Relative absolute error 37.7001 %
Root relative squared error 82.0203 %
Total Number of Instances 351
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC
0.865 0.196 0.712 0.865 0.781 0.648
0.804 0.135 0.914 0.804 0.856 0.648
Weighted Avg. 0.826 0.157 0.842 0.826 0.829 0.648
=== Confusion Matrix ===
a b <-- classified as
109 17 | a = b
44 181 | b = g
Status
OK
Log
x 0

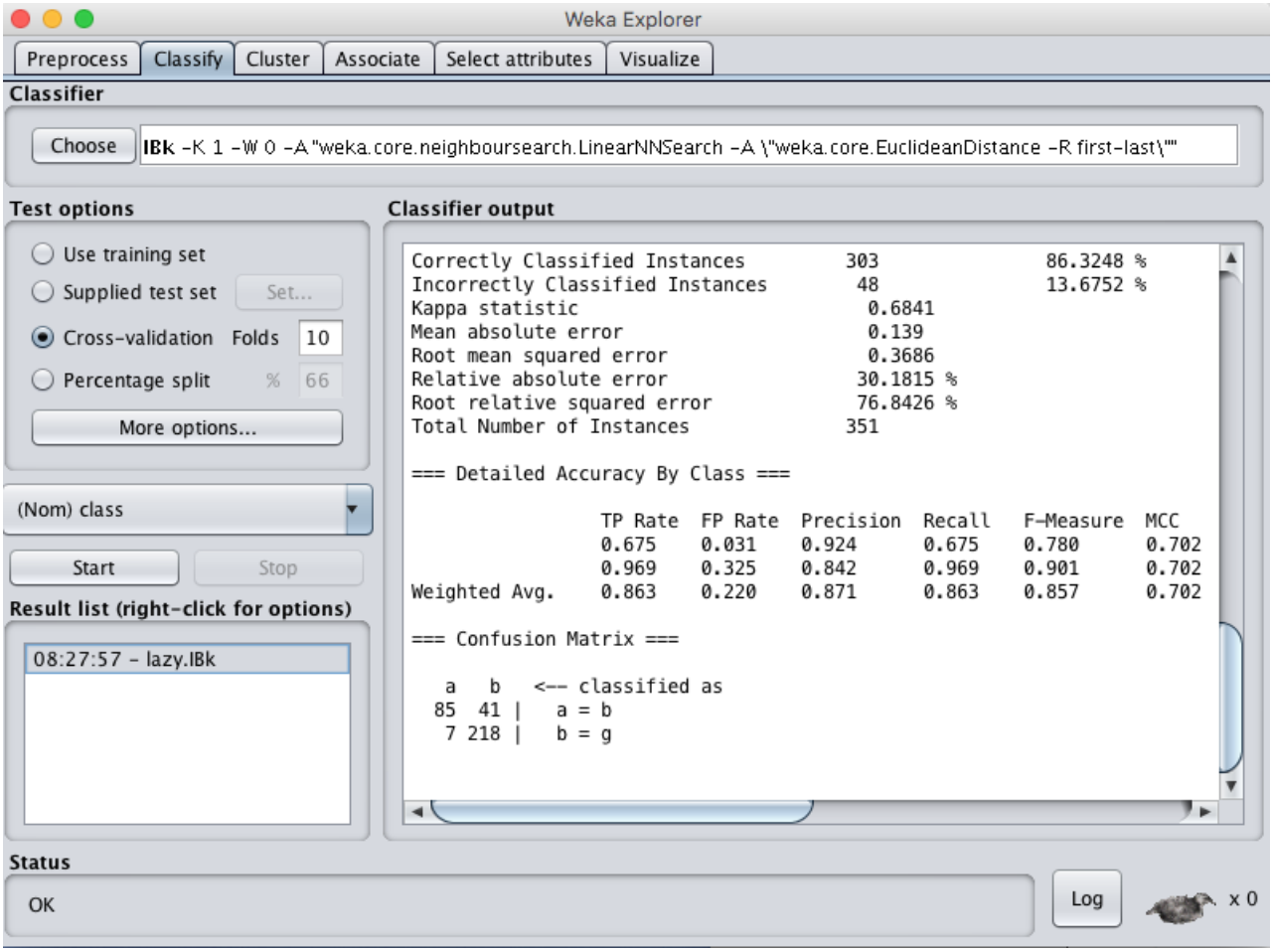
Decision Tree



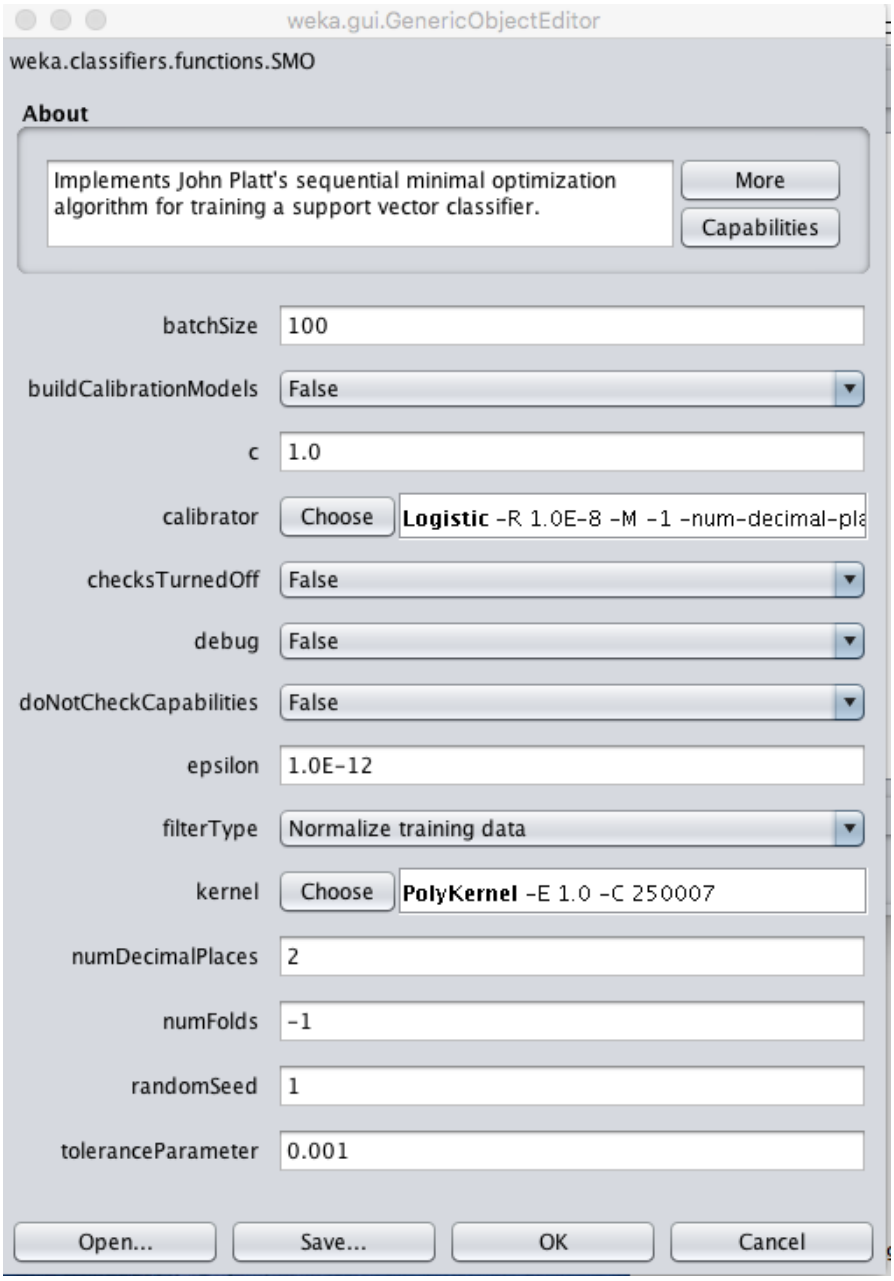


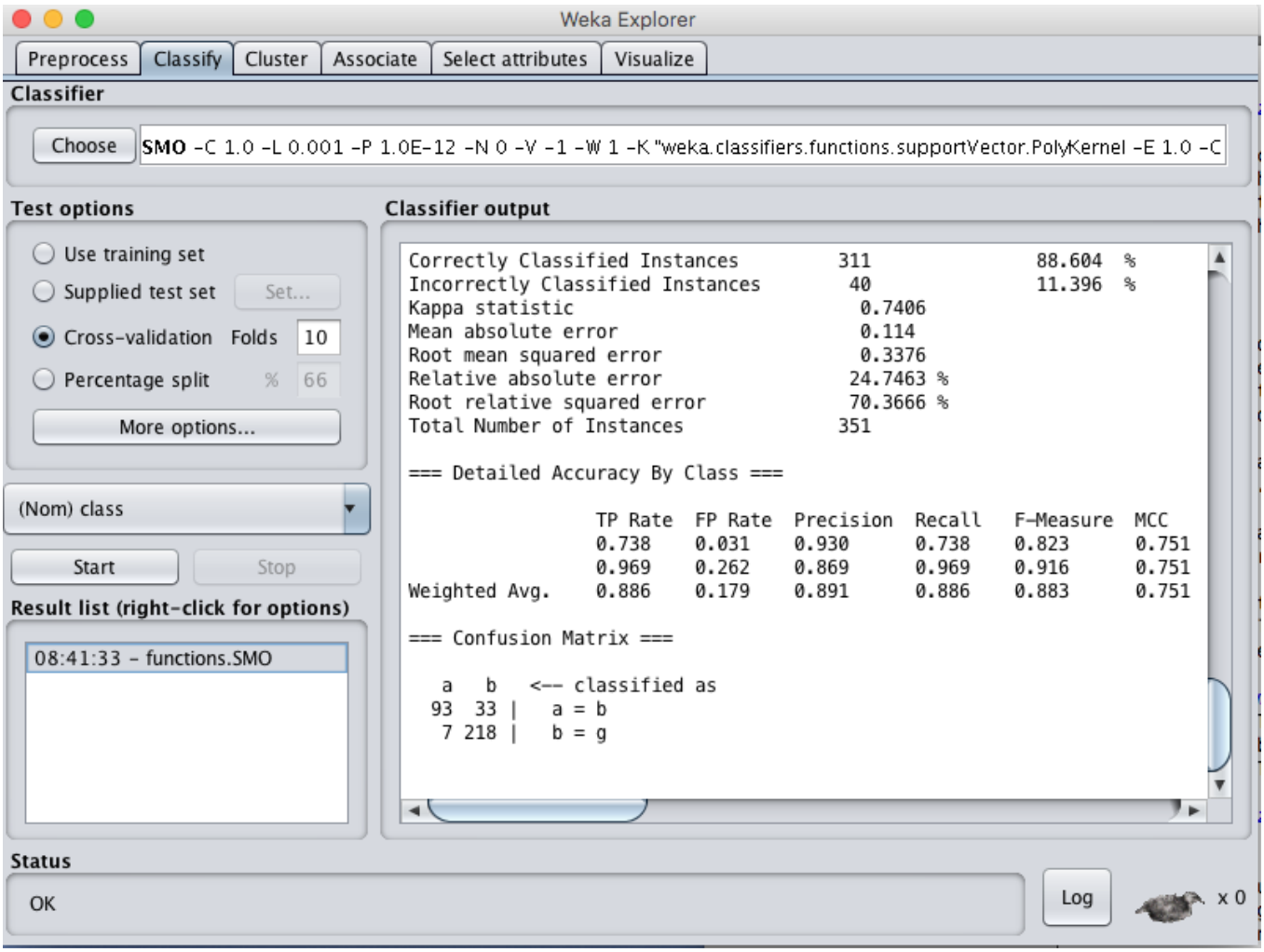
k-Nearest Neighbors





Support Vector Machines





Observations

Comparing all the algorithms SVM showed the least Relative absolute errors.