CS634 DATA MINING

FINAL TERM PROJECT(OPTION 1)

SUPERVISED DATA MINING(CLASSIFICATION)

Vidya Kalappurayil Viswanathan
Email: vk427@njit.edu
UCID: vk427

Contents

1. Introduction

Data mining also known as Knowledge Discovery in Databases (KDD) is the process of discovering hidden patterns and models in large and complex data bases by eliminating randomness. Though used interchangeably, data mining is actually part of the knowledge discovery process. It is the crucial step in which clever techniques are applied to extract patterns potentially useful.[1]

Data mining can be classified into two categories

- Supervised Datamining: In supervised datamining, the data as presented to a machine learning algorithm is fully labeled; all examples are presented with a classification that the machine is meant to reproduce. There will be a subset of data points for which the target value is already known. This data is used to build a model of what a typical data point looks like when it has one of the various target values. The model is then applied to data for which that target value is currently unknown. The algorithm identifies the "*new*" data points that match the model of each target value
- Unsupervised Datamining: Unsupervised data mining does not focus on predetermined attributes, nor does it predict a target value. Unsupervised data mining finds hidden structure and relation among data.[2]

A Classification Algorithm is a supervised algorithm, that attempts to draw conclusion from observed values. Classification either predicts categorical class labels or classifies data (construct a model) based on the training set and the values (class labels) in classifying attributes and uses it in classifying new data[3].Applications of classification algorithms include credit approval, target marketing, disease treatment etc.

2. Objective

The aim of this project is to perform supervised learning on a dataset by implementing two classification algorithms (Option 1) and to compare and evaluate the accuracies of the classification algorithms implemented. The classification algorithms are implemented using the software tool WEKA on the dataset Autistic Spectrum Disorder Screening Data for Adult

The classification algorithms implemented in this project are

- Decision Tree(J48) - Category 3
- Naive Bayes – Category 5

The accuracies of the algorithms implemented are evaluated using 10-fold cross validation methods to identify which algorithm performs better on the given dataset.

3. Software

WEKA (Waikato Environment for Knowledge Analysis) is an open source software developed in Java by the University of Waikato New Zealand. Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code.[4] Weka contains a collection

[1] http://webdocs.cs.ualberta.ca/~zaiane/courses/cmput690/notes/Chapter1/
[2] https://cloudtweaks.com/2014/09/use-supervised-unsupervised-data-mining/
[3] https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/
[4] https://www.cs.waikato.ac.nz/ml/weka/

of visualization tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces for easy access to these functions.[5]

Weka uses the **Attribute Relation File Format (ARFF)** for data analysis, by default. It also supports CSV and ODBC formats, from where the data can be imported. The **Attribute Relation File Format (ARFF)** has two parts:

- The header section defines the relation (data set) name, attribute name and the type.
- The data section lists the data instances.

WEKA can be downloaded from the official website: https://www.cs.waikato.ac.nz/ml/weka/downloading.html

4. Dataset

The dataset used in the project is **Autistic Spectrum Disorder Screening Data for Adult.** Autistic Spectrum Disorder (ASD) is a neurodevelopment condition associated with significant healthcare costs, and early diagnosis can significantly reduce these. The data set contains data of autism screening of adults with 20 attributes (features)that can be utilized to determine the autistic traits and improve the classification of ASD cases. The objective of this project is to implement two Classifier algorithms on this dataset, to predict whether an adult has ASD traits or not. The attribute ASD Class is the label predicted by the algorithms.

The dataset can be downloaded from the following location:

http://archive.ics.uci.edu/ml/datasets/Autism+Screening+Adult

Number of Instances: 704

Number of Attributes: 21

Attribute Information

Attribute	**Description**
Age	Age in years
Gender	Male or Female
Ethnicity	List of common ethnicities
Born with jaundice	Whether the case was born with jaundice (YES/NO)
Family member with PDD	Whether any immediate family member has a PDD (YES/NO)
Who is completing the test (Relation)	Parent, self, caregiver, medical staff, clinician, etc.
Country of residence	List of countries
Used the screening app before	Whether the user has used a screening app (YES/NO)
Screening Method Type	The type of screening methods chosen based on age category (0=toddler, 1=child, 2= adolescent, 3= adult)
Question 1 Answer (A1)	The answer code of the question based on the screening method used

[5] https://en.wikipedia.org/wiki/Weka_(machine_learning)

Question 2 Answer (A2)	The answer code of the question based on the screening method used
Question 3 Answer (A3)	The answer code of the question based on the screening method used
Question 4 Answer (A4)	The answer code of the question based on the screening method used
Question 5 Answer (A5)	The answer code of the question based on the screening method used
Question 6 Answer (A6)	The answer code of the question based on the screening method used
Question 7 Answer (A7)	The answer code of the question based on the screening method used
Question 8 Answer (A8)	The answer code of the question based on the screening method used
Question 9 Answer (A9)	The answer code of the question based on the screening method used
Question 10 Answer (A10)	The answer code of the question based on the screening method used
Screening Score (Result)	The final score obtained based on the scoring algorithm of the screening method used. This was computed in an automated manner
ASD Class	YES/NO

5. Algorithms

5.1. Decision Tree (Category 3)

Decision Tree belongs to the category of Supervised learning algorithms. It is used to create a training model which can be used to predict class or label of target variables by learning decision rules inferred from training data. Decision Tree tries to solve the problem using a tree representation. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. [6]

In a decision tree

- An Internal node represents an attribute
- Leaf node represents the class label
- Branch represents an outcome of the test

Decision tree algorithms usually work top-down, by choosing an attribute at each step that best splits the set of items. Different algorithms use different metrics for measuring "best". These generally measure the homogeneity of the target variable within the subsets.[7] The popular attribute selection measures are

- Information gain
- Gini index

[6] http://www.saedsayad.com/decision_tree.htm
[7] https://en.wikipedia.org/wiki/Decision_tree_learning

Algorithm[8]:

- At each level, select an attribute to branch on (always select the attribute with the highest information gain).
- Suppose there are two classes P (positive) and N (negative) where P contains p training examples and N contains n training examples

$$\boldsymbol{I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}}$$

- Assume that using attribute A, a set S will be partitioned into subsets $\{S_1, S_2, \ldots, S_v\}$. If S_i contains p_i examples of P and n_i examples of N, the entropy, or the expected information needed to classify objects in all subtrees S_i is

$$\boldsymbol{E(A) = \sum_{i=1}^{v}\frac{p_i + n_i}{p+n}\, I(p_i, n_i)}$$

$$\boldsymbol{Gain(A) = I(p,n) - E(A)}$$

5.2. Naive Bayes (Category 5)

The **Naive Bayes** classification algorithm is a probabilistic classifier. Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of feature[9] i.e., Naive Bayes classifier assumes that all the features are unrelated to each other. Presence or absence of a feature does not influence the presence or absence of any other feature.

Bayes Theorem is defined as

$$\boldsymbol{P(A|B) = \frac{P(A|B)\,P(A)}{P(B)}}$$

Where P(A|B) is the posterior probability, P(A) is the prior probability, P(B|A) is the class conditional probability and P(B) is the evidence

Naive Bayes is particularly suited when the dimensionality of the inputs is high. Even if we are working on a data set with millions of records with some attributes, it is suggested to try Naive Bayes approach. Naive Bayes classifiers are among the most successful known algorithms for learning to classify text documents. Spam filtering is the best-known use of Naive Bayesian text classification. It makes use of a naive Bayes classifier to identify spam e-mail.

[8] https://lmscontent.embanet.com/NJIT/CS634/documents/CS634-w03-m01A.pdf
[9] http://scikit-learn.org/stable/modules/naive_bayes.html

Naive Bayes Algorithm[10]:

- Learning/training phase: given a training data set T,
  - For each label Yi, P(Yi) ← calculate P(Y=Yi) using training examples in T;
  - For each attribute value Xj, calculate P(Xj|Yi) using training examples in T.

- Testing Phase: given an unlabeled test record $X^* = <X_1 \ldots X_n>$
  - Assign label Y to X* based on max{P(Yi)P(X1|Yi)P(X2|Yi) P(Xn|Yi)}

6. Implementation details

Software	: WEKA 3.8
Input File Name	: Autistic Spectrum Disorder Screening Data for Adult
Input file format	: ARFF
Algorithms	: Decision Tree (Category 3)
	: Naïve Bayes (Category 5)
Programming Language	: JAVA

7. Implementation and Analysis

Following are the steps carried out in the process of implementing and analyzing the classifier algorithms – Decision Tree (Category.3) and Naïve Bayes (Category.5).

7.1. Data Preparation

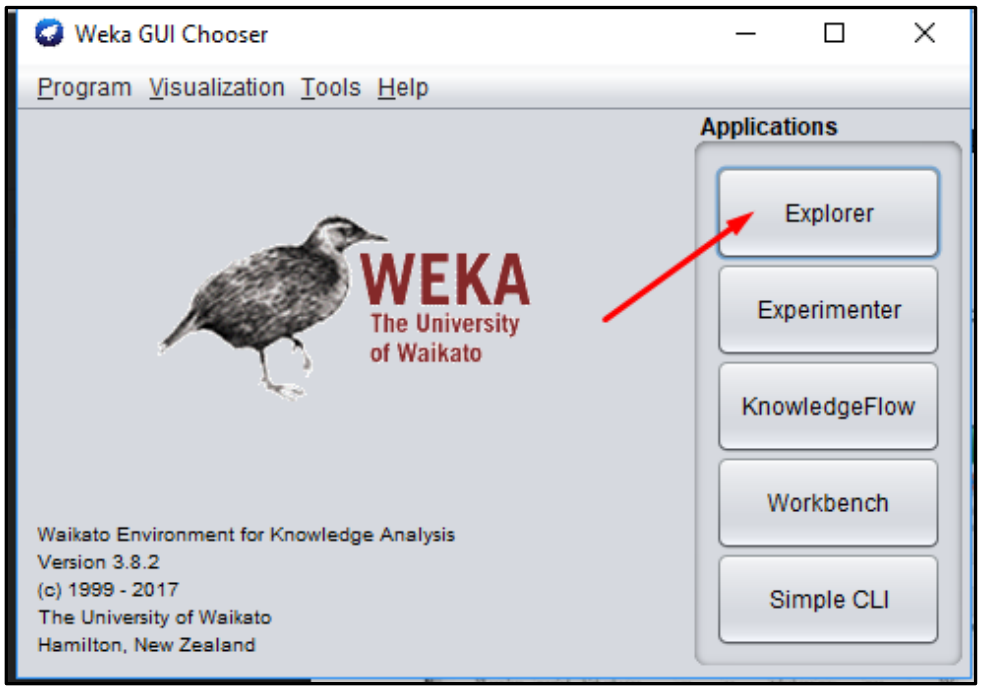
The dataset was already available in arff format with missing values indicated as '?'. The missing values were handled using the Data Preprocessing option in Weka tool

```
@relation adult-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-10

@attribute A1_Score {0,1}
@attribute A2_Score {0,1}
@attribute A3_Score {0,1}
@attribute A4_Score {0,1}
@attribute A5_Score {0,1}
@attribute A6_Score {0,1}
@attribute A7_Score {0,1}
@attribute A8_Score {0,1}
@attribute A9_Score {0,1}
@attribute A10_Score {0,1}
@attribute age numeric
@attribute gender {f,m}
@attribute ethnicity {White-European,Latino,Others,Black,Asian,'Middle Eastern ',Pasifika,'South Asian',Hispanic,Turkish,others}
@attribute jundice {no,yes}
@attribute austim {no,yes}
@attribute contry_of_res {'United States',Brazil,Spain,Egypt,'New Zealand',Bahamas,Burundi,Austria,Argentina,Jordan,Ireland,'United Arab Emirat
@attribute used_app_before {no,yes}
@attribute result numeric
@attribute age_desc {'18 and more'}
@attribute relation {Self,Parent,'Health care professional',Relative,Others}
@attribute Class/ASD {NO,YES}

@data
1,1,1,1,0,0,1,1,0,0,26,f,White-European,no,no,'United States',no,6,'18 and more',Self,NO
1,1,0,1,0,0,0,1,0,1,24,m,Latino,no,yes,Brazil,no,5,'18 and more',Self,NO
1,1,0,1,1,0,1,1,1,1,27,m,Latino,yes,yes,Spain,no,8,'18 and more',Parent,YES
1,1,0,1,0,0,1,1,0,1,35,f,White-European,no,yes,'United States',no,6,'18 and more',Self,NO
1,0,0,0,0,0,0,1,0,0,40,f,?,no,no,Egypt,no,2,'18 and more',?,NO
1,1,1,1,1,0,1,1,1,1,36,m,Others,yes,no,'United States',no,9,'18 and more',Self,YES
0,1,0,0,0,0,0,1,0,0,17,f,Black,no,no,'United States',no,2,'18 and more',Self,NO
1,1,1,1,0,0,0,0,1,0,64,m,White-European,no,no,'New Zealand',no,5,'18 and more',Parent,NO
```

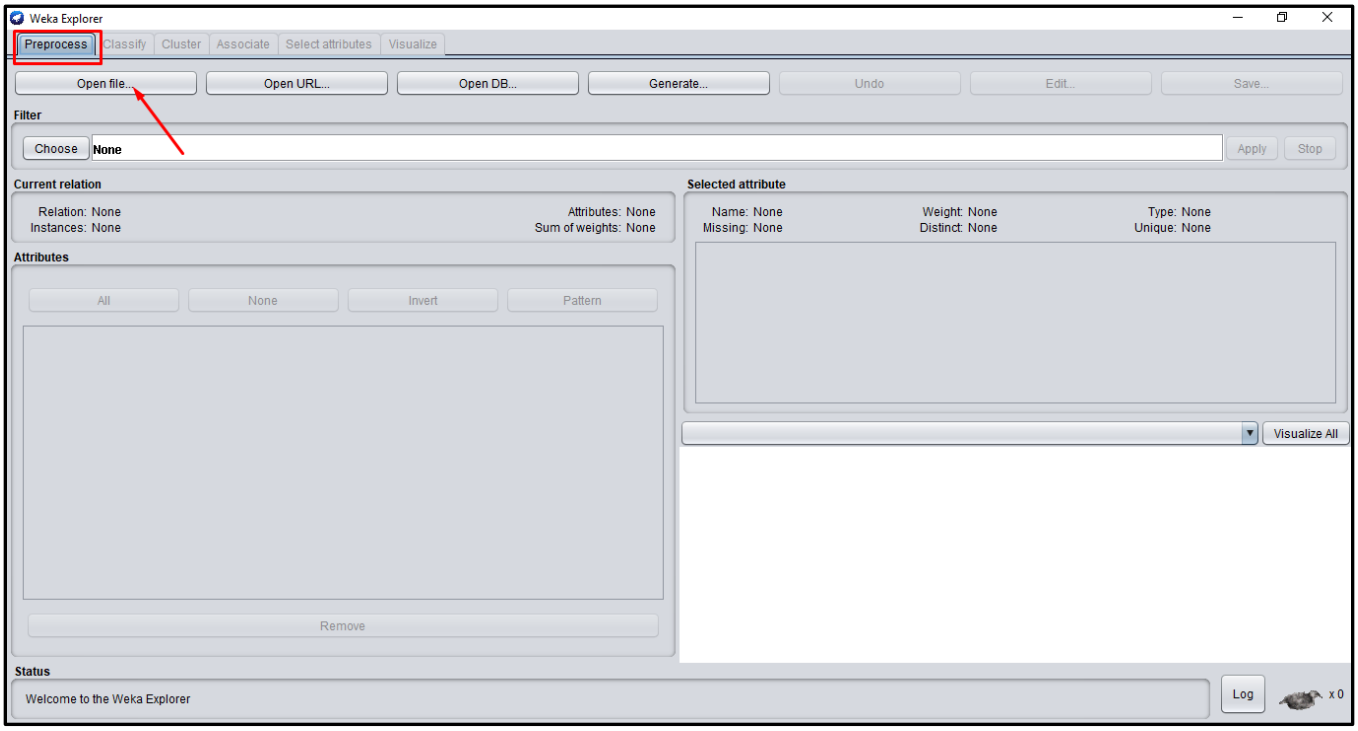
[10] https://lmscontent.embanet.com/NJIT/CS634/documents/CS634-w05-m01A.pdf

7.2. Data Preprocessing using WEKA

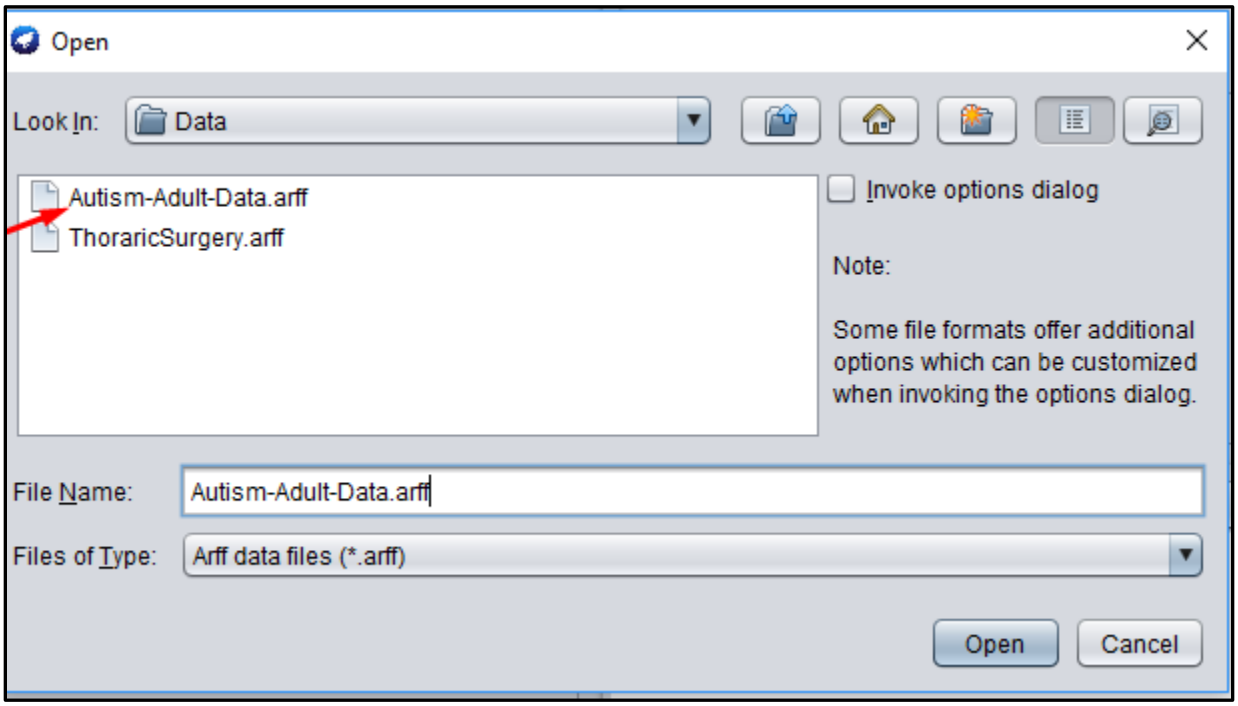
- Download and install WEKA from the URL: https://www.cs.waikato.ac.nz/ml/weka/downloading.html
- Launch the WEKA application. Click on the Explorer tab under WEKA GUI Chooser



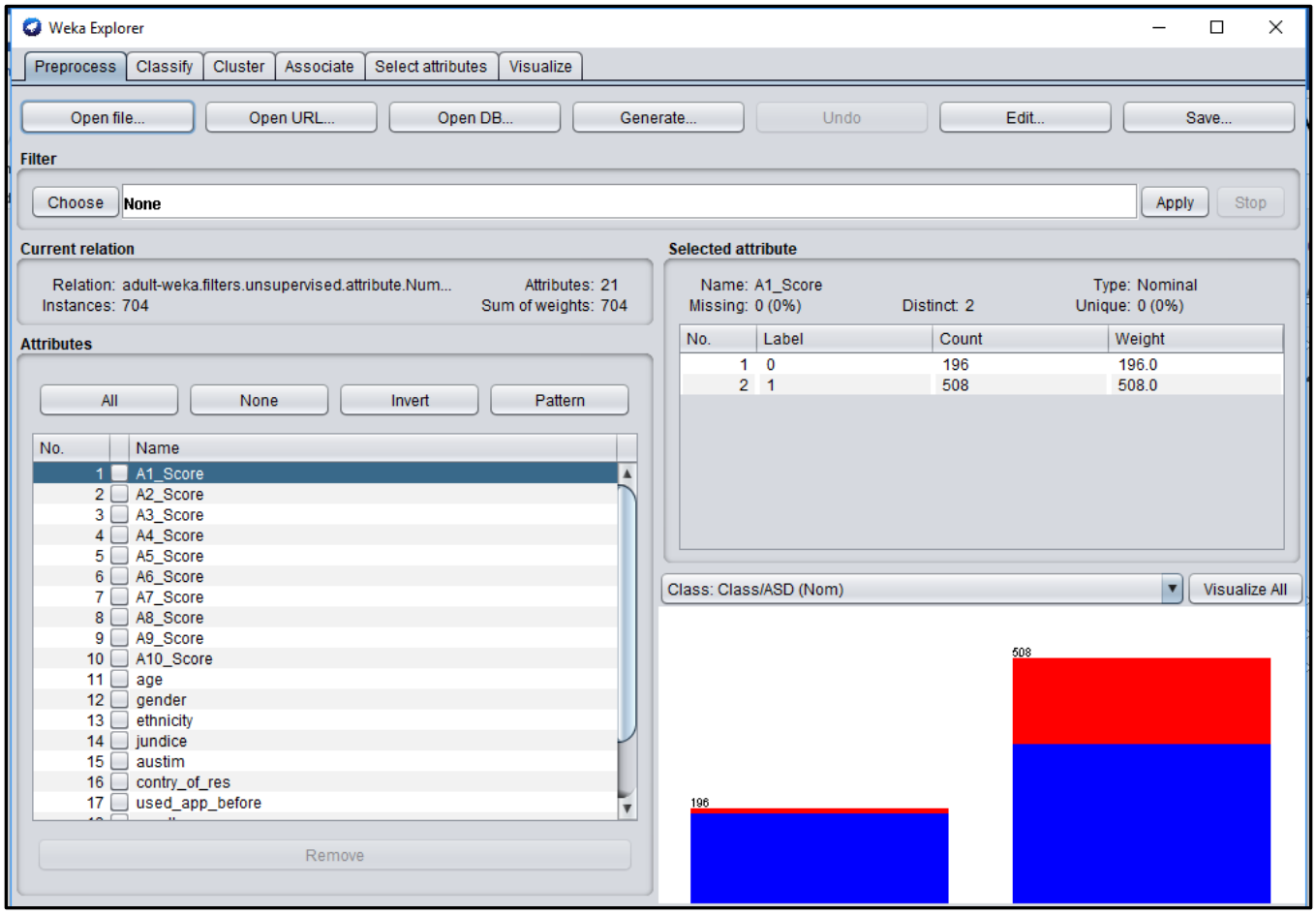
- Under the **Preprocess** tab, click on **Open file** to open the dataset



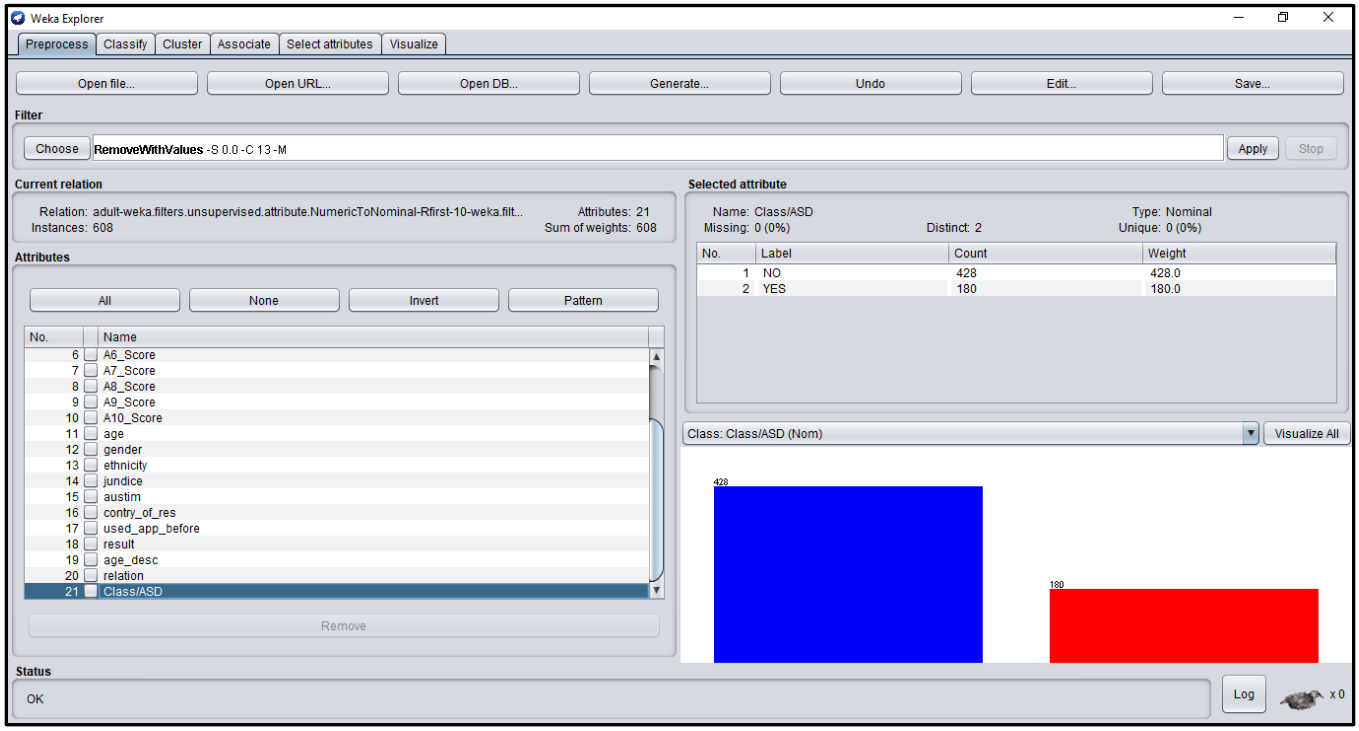
- Select the specific dataset (here: Autism-Adult-Data.arff) and click **Open**



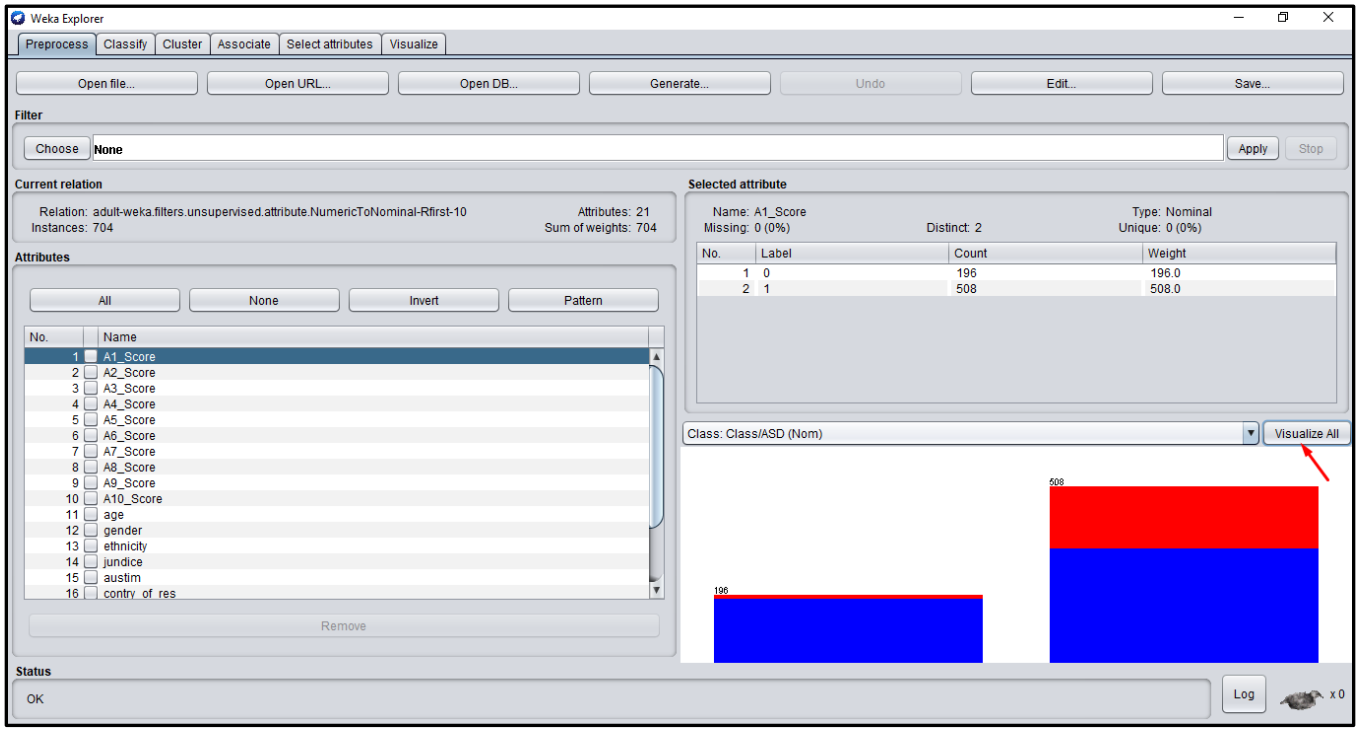
- On clicking Open, the file is opened in WEKA Explorer. The list of attributes in the dataset and a graphical representation of the selected attribute is displayed



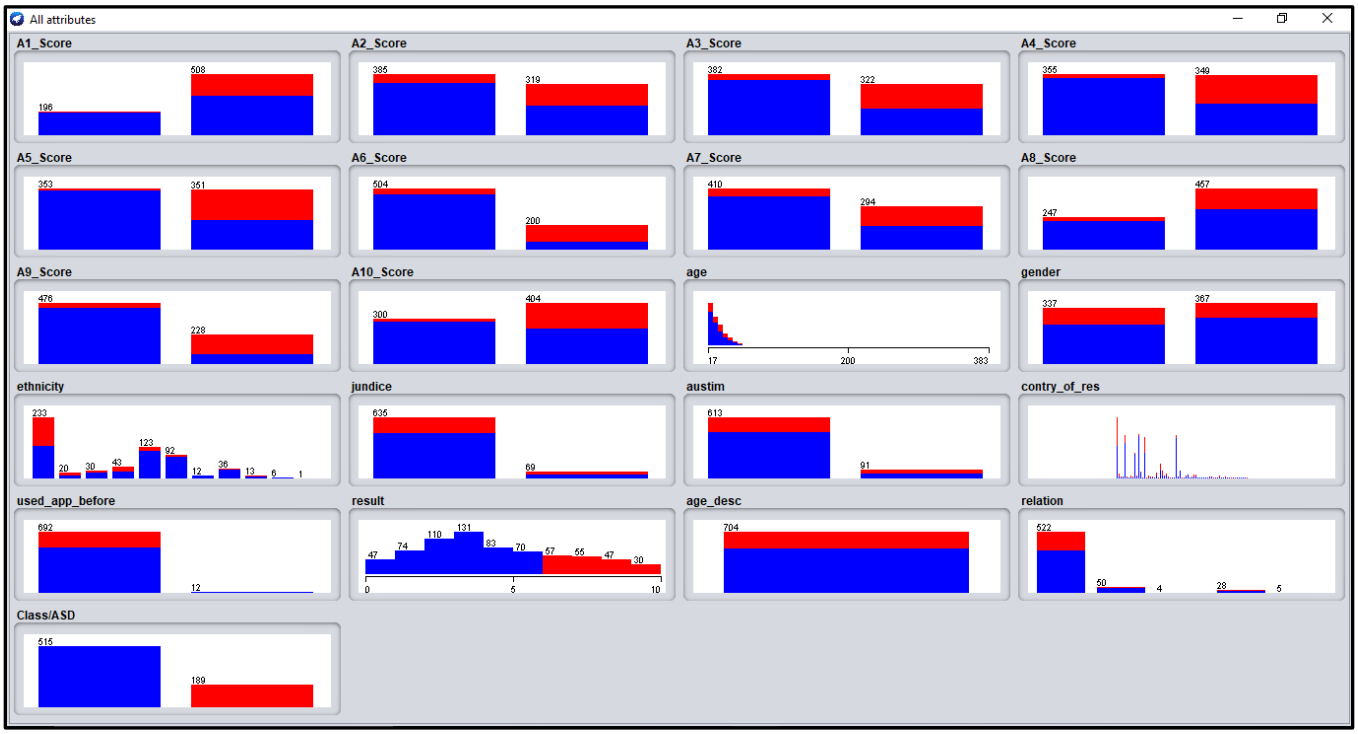
- The below screenshot shows the visual representation for the Class/Label attribute, with the number of YES and NO labels



- Click on **Visualize all** to view the graphical representation of all attributes in the dataset



- Below is the screenshot of visual representation for the all the attributes in the dataset



7.2.1. Data Preprocessing Steps

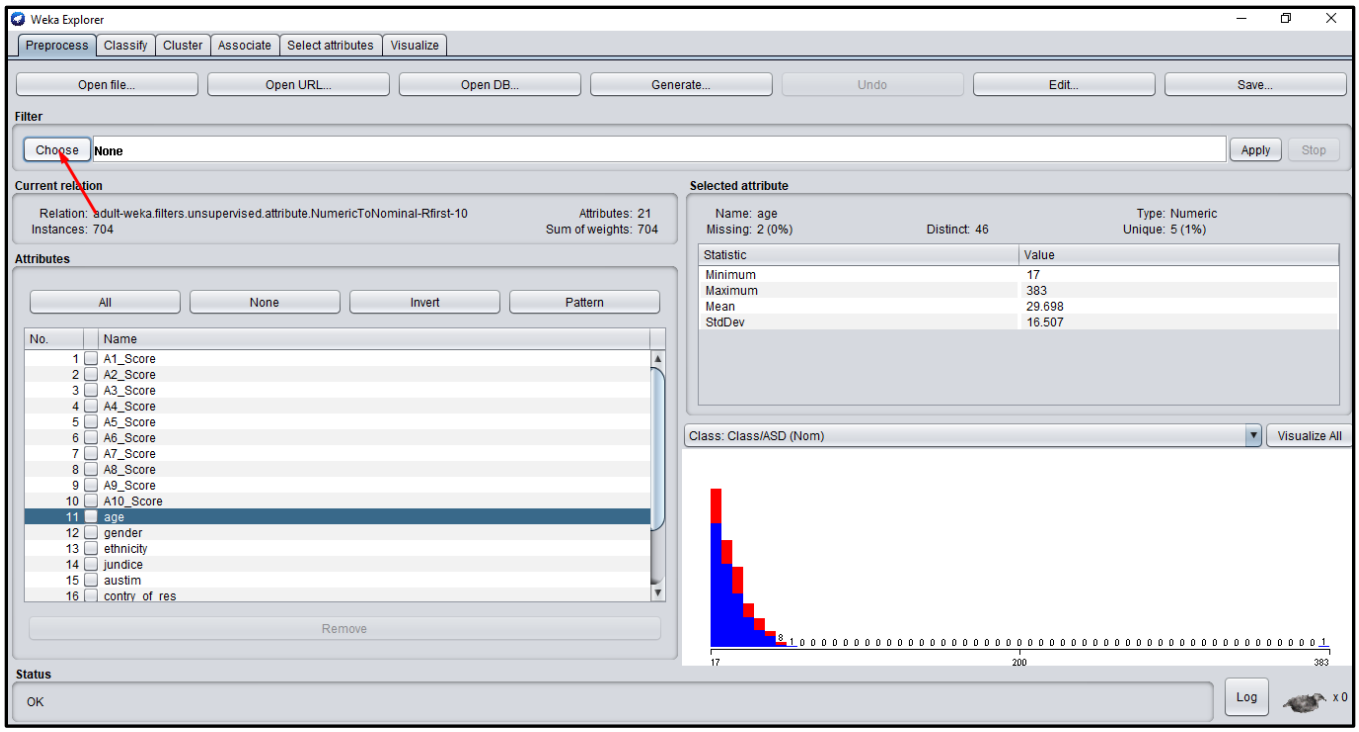
In the data preprocessing stage, the Autism dataset is found to have missing values (for the attributes ethnicity, age) and erroneous values (age =383). This can be handled in WEKA using the Filter Option.

7.2.1.1. Remove Missing Values

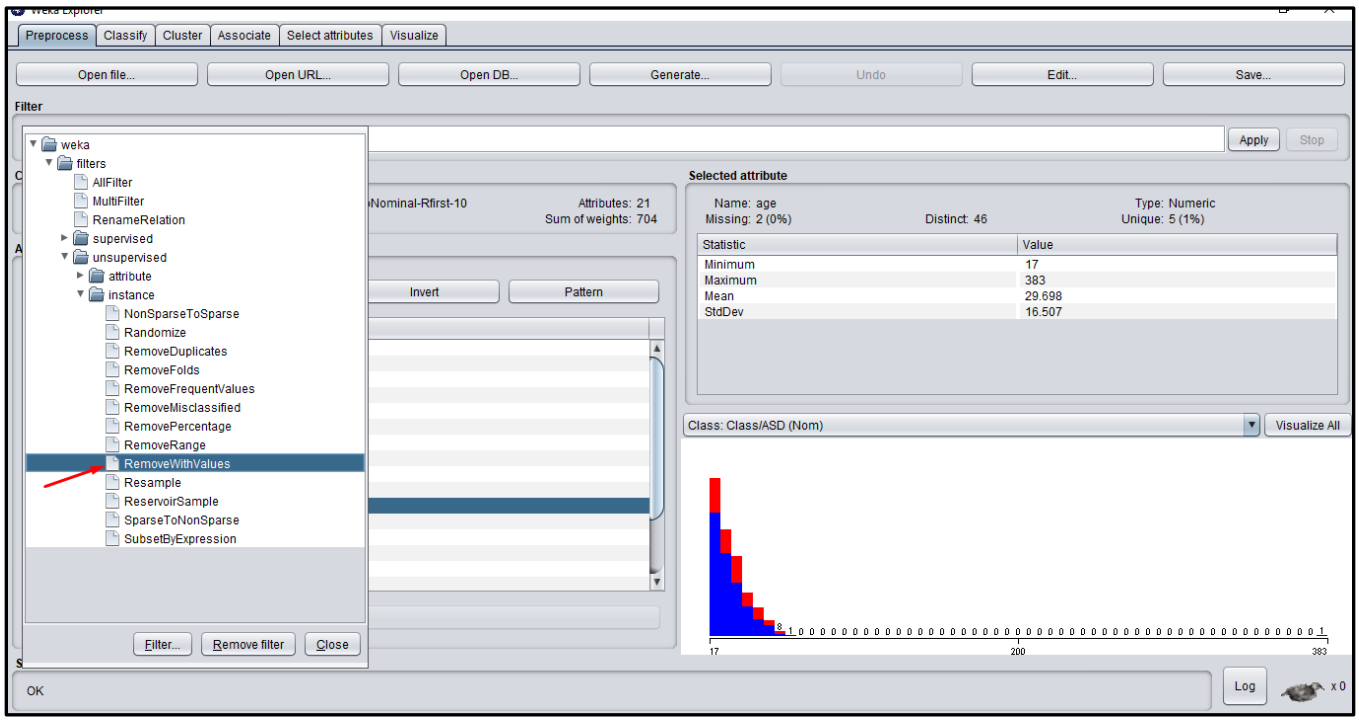
The Autism dataset has missing values for 95 instances. Here, we handle it by removing those instances from the training dataset. An alternate approach is to replace the missing values with the median values, which can be handled in the tool itself, using filter.

In order to handle the missing values in the dataset

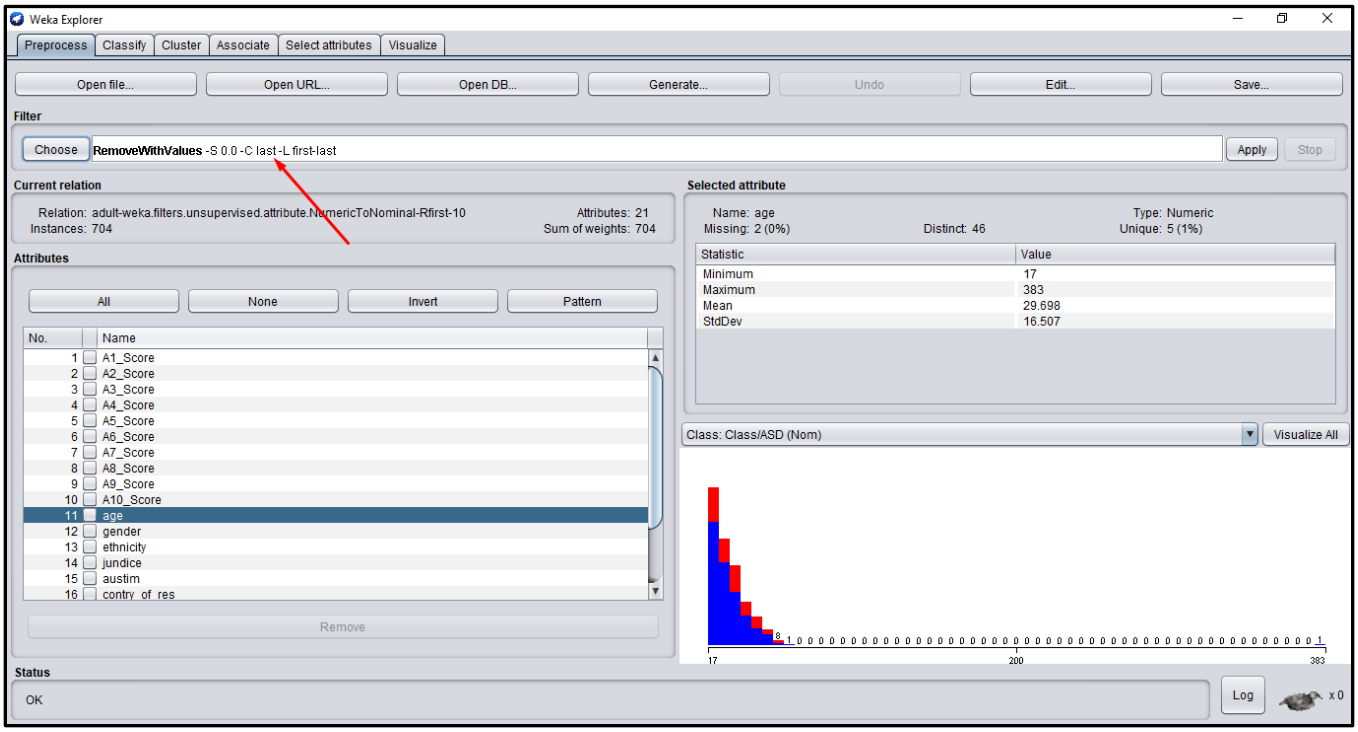
- Click on **Choose** button under the 'Filter' section, of the Preprocess Tab



- Navigate to **Filters** > **Unsupervised** > **Instance**, and select **RemoveMissingValues**

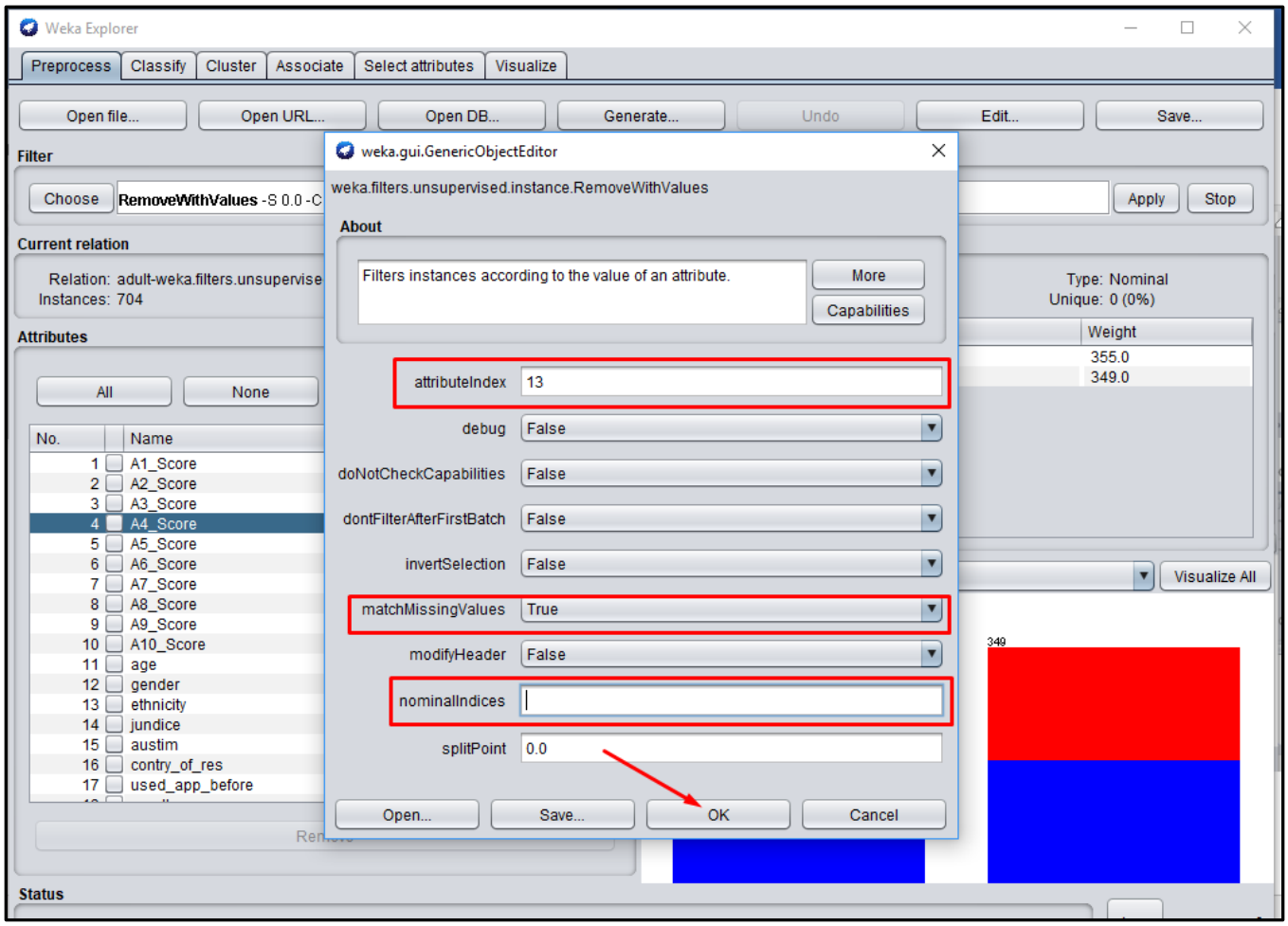


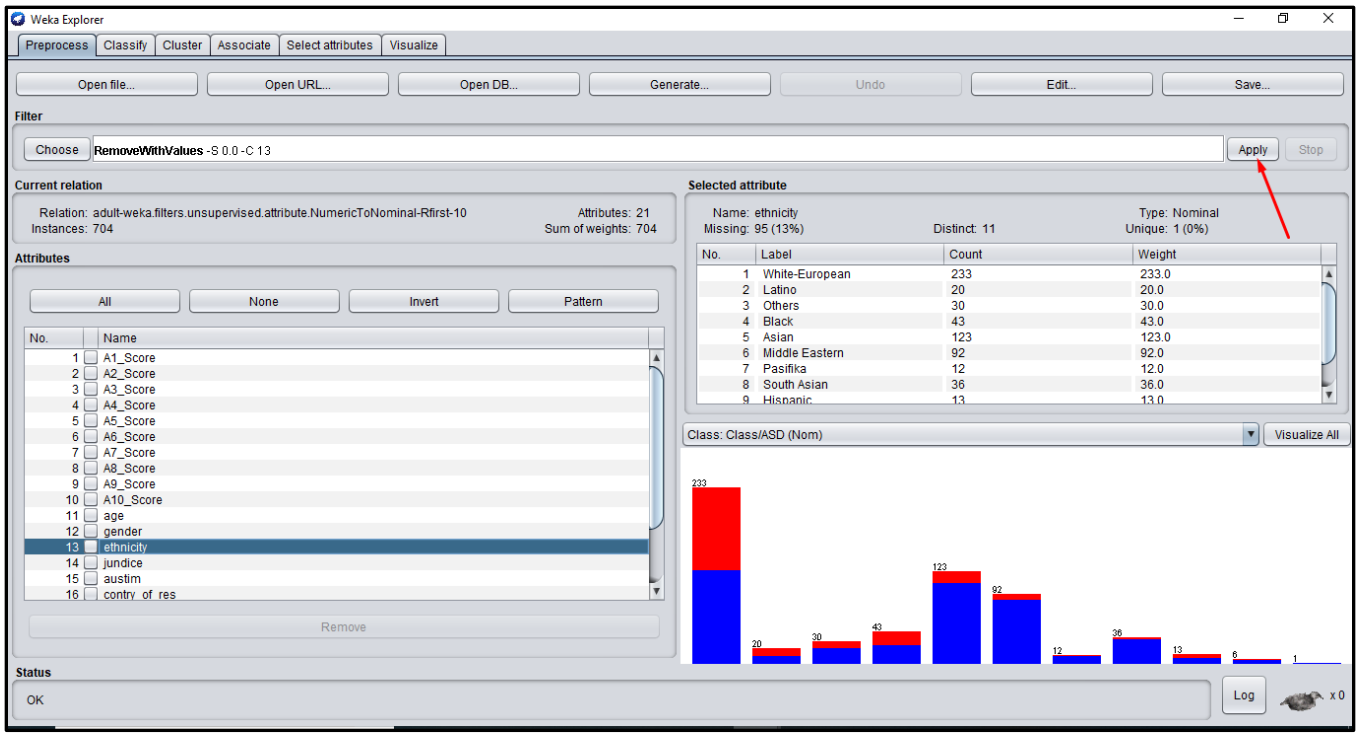
- Left Click on the indicated area in the screenshot to open the **Edit properties window** for the object. A pop-up window will be opened.



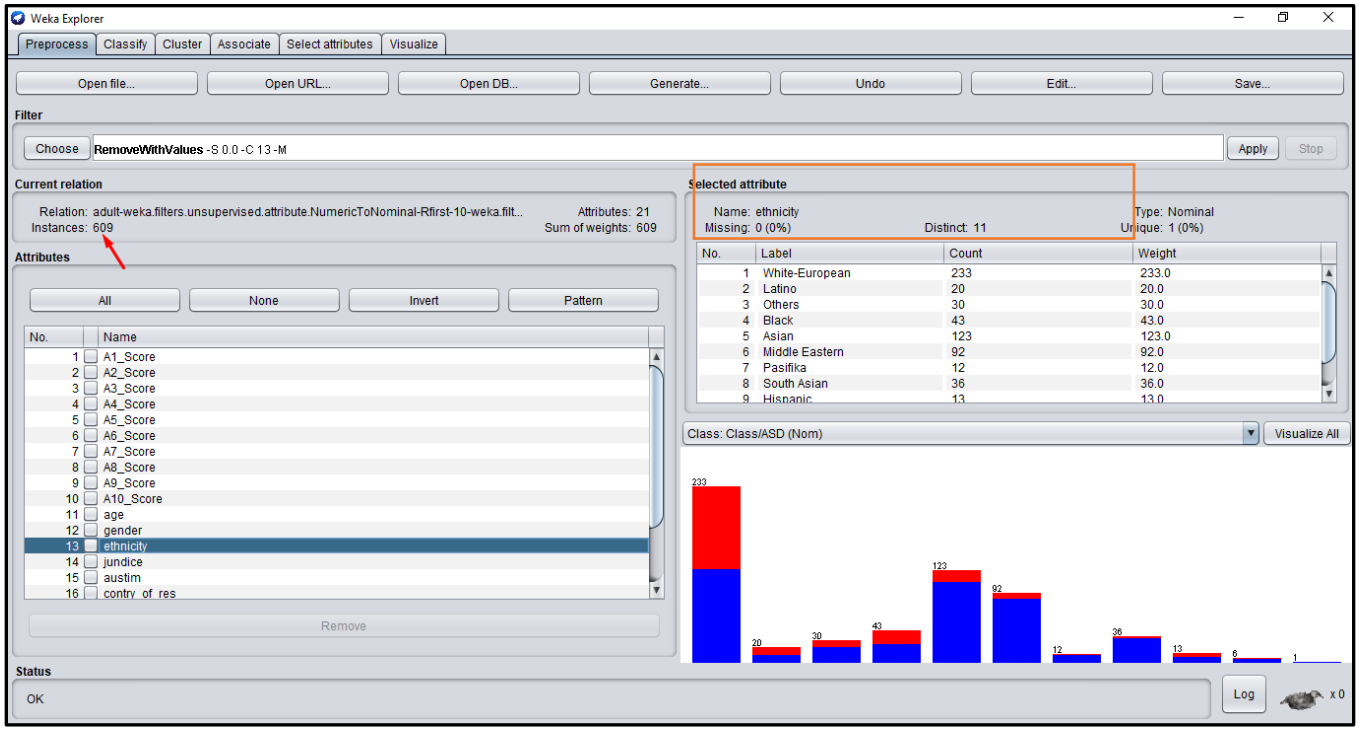
- On the object Editor, set the following and click '**OK'**. Click on **Apply** button in the Preprocess tab to apply the filter
    - **attribute index to 13 (index of the attribute with missing value, here, Ethnicity with index =13)**
    - **matchMissingvalues to TRUE**
    - **Nominal Indices to blank**

Note: -The filter will not be applied until the 'Apply' button is clicked


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
RemoveWithValues -S 0.0 -C
Apply
Stop
Current relation
Relation: adult-weka.filters.unsupervise
Instances: 704
Attributes
All
None
No.
Name
1 A1_Score
2 A2_Score
3 A3_Score
4 A4_Score
5 A5_Score
6 A6_Score
7 A7_Score
8 A8_Score
9 A9_Score
10 A10_Score
11 age
12 gender
13 ethnicity
14 jundice
15 austim
16 contry_of_res
17 used_app_before
Type: Nominal
Unique: 0 (0%)
Weight
355.0
349.0
Visualize All
349
Status
weka.gui.GenericObjectEditor
weka.filters.unsupervised.instance.RemoveWithValues
About
Filters instances according to the value of an attribute.
More
Capabilities
attributeIndex 13
debug False
doNotCheckCapabilities False
dontFilterAfterFirstBatch False
invertSelection False
matchMissingValues True
modifyHeader False
nominalIndices
splitPoint 0.0
Open...
Save...
OK
Cancel



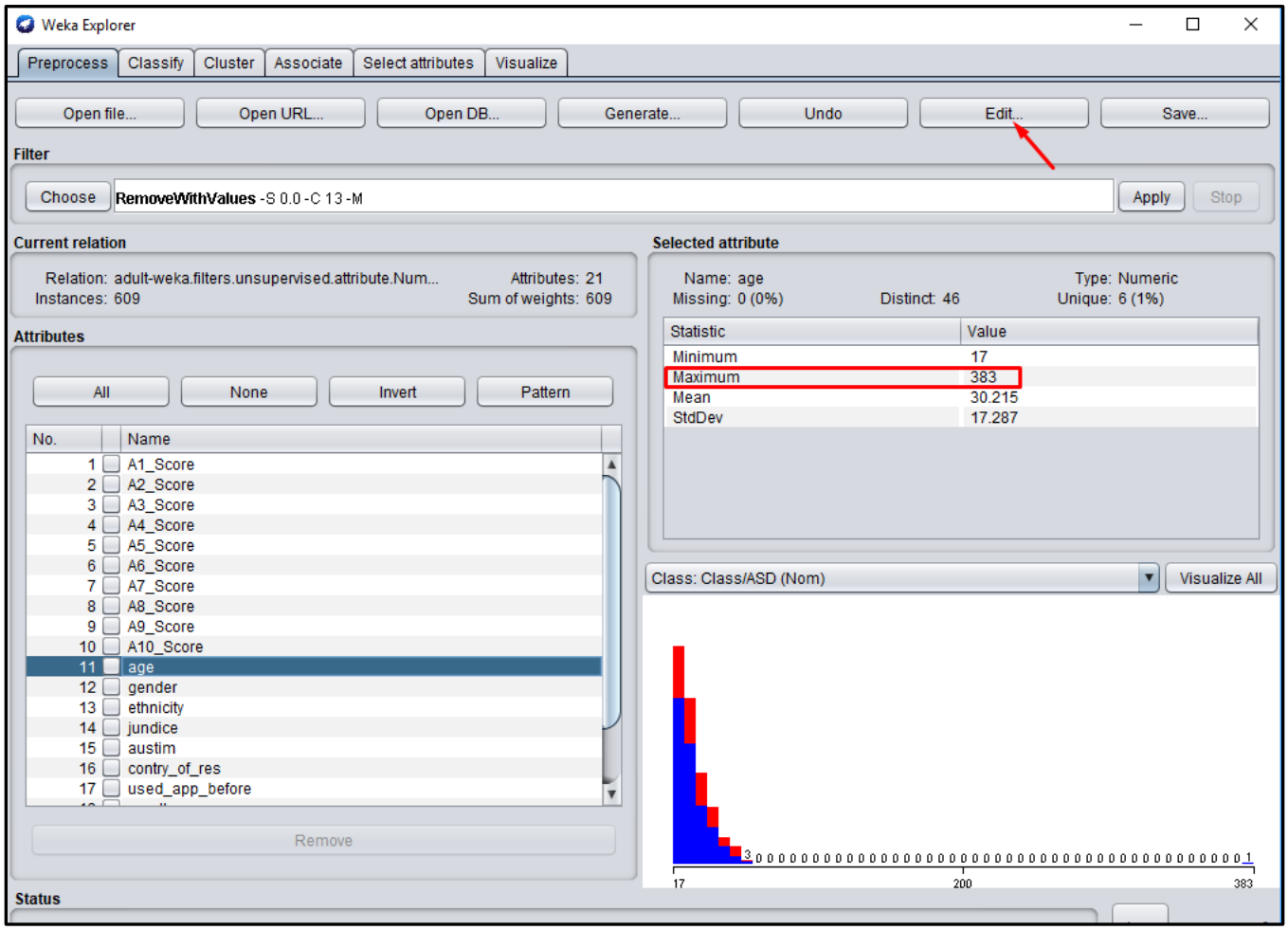
- The below screenshot indicates that the instances with missing values (count =95) has been deleted and the number of total instances is now 609.The Missing values for the attribute is now shown as 0%



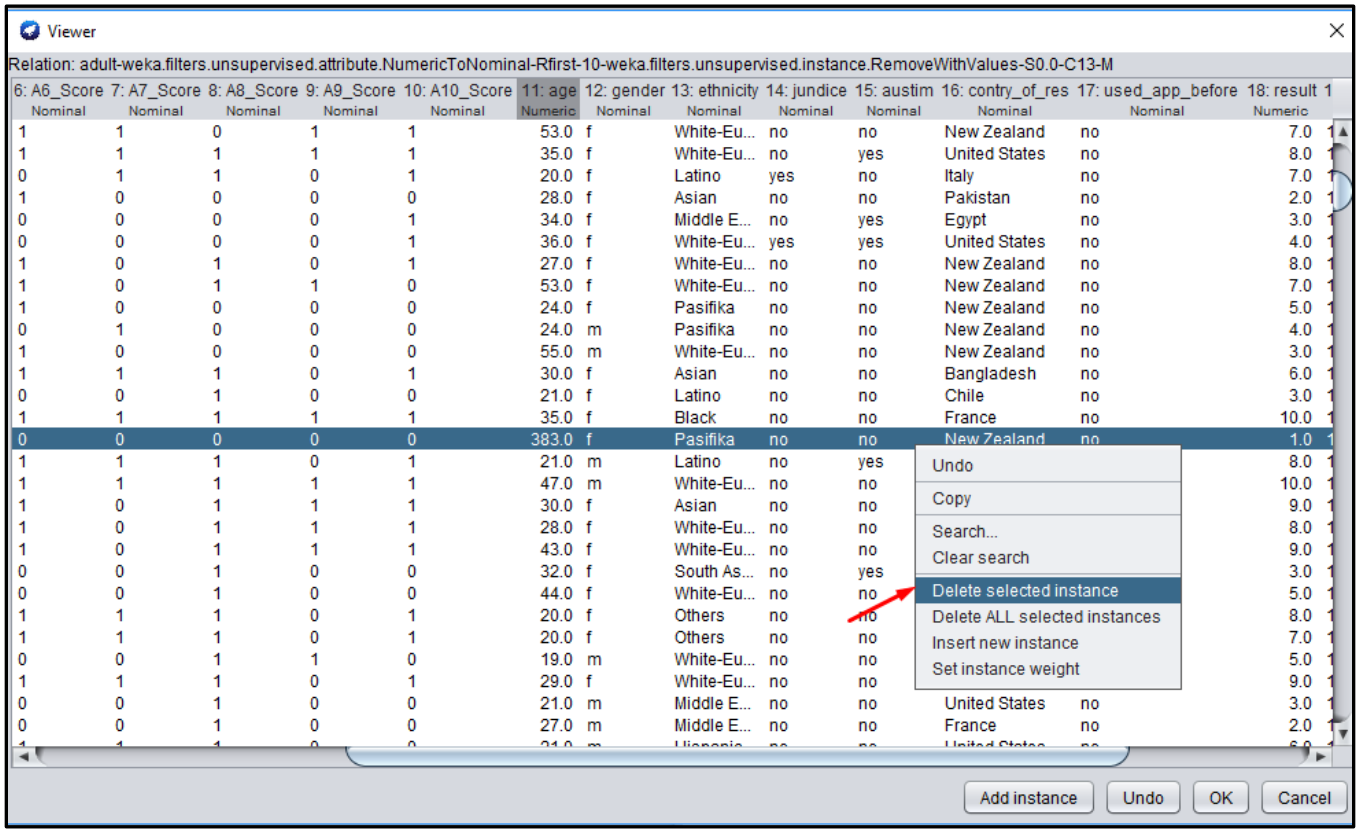
7.2.1.2. Remove erroneous values

The Dataset is found to have an erroneous value for the age attribute (age =383). This instance is deleted to avoid any errors while implementing the algorithms. In order to delete the instance,

- Click on Edit button under the Preprocess Tab. The dataset will be opened in the edit mode in a new window.



- Navigate to the instance with the incorrect value, right click and choose ‘Delete Selected Instance’. Click OK. The selected instance will be deleted. (Check on the Data preprocess tab to view the number of instances in the dataset)

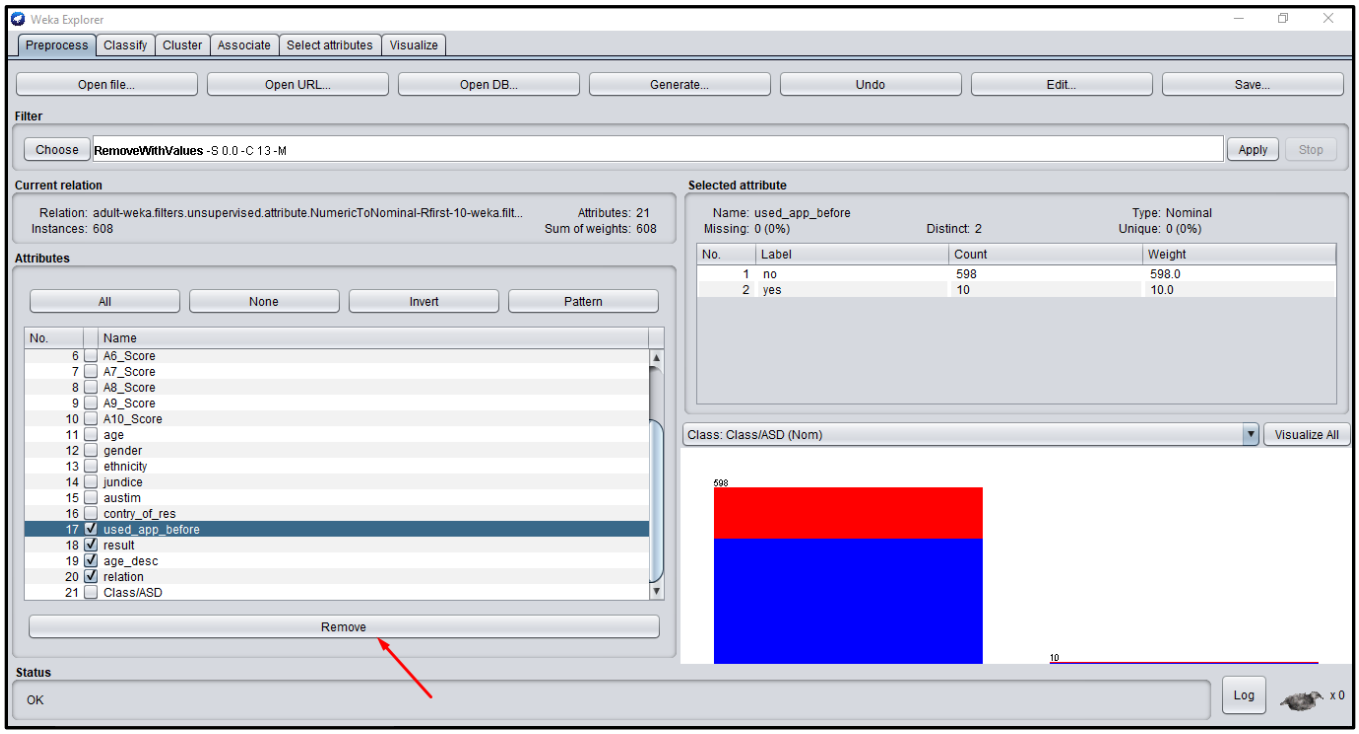


7.2.1.3. Remove attributes (Feature selection)

The less important features can be removed from the dataset. Here the attributes sucha s ‘used app before’,age desc etc are least importantant features. In this experiment we focus only on 17 attributes.

Inorder to delete the insignificant attributes,

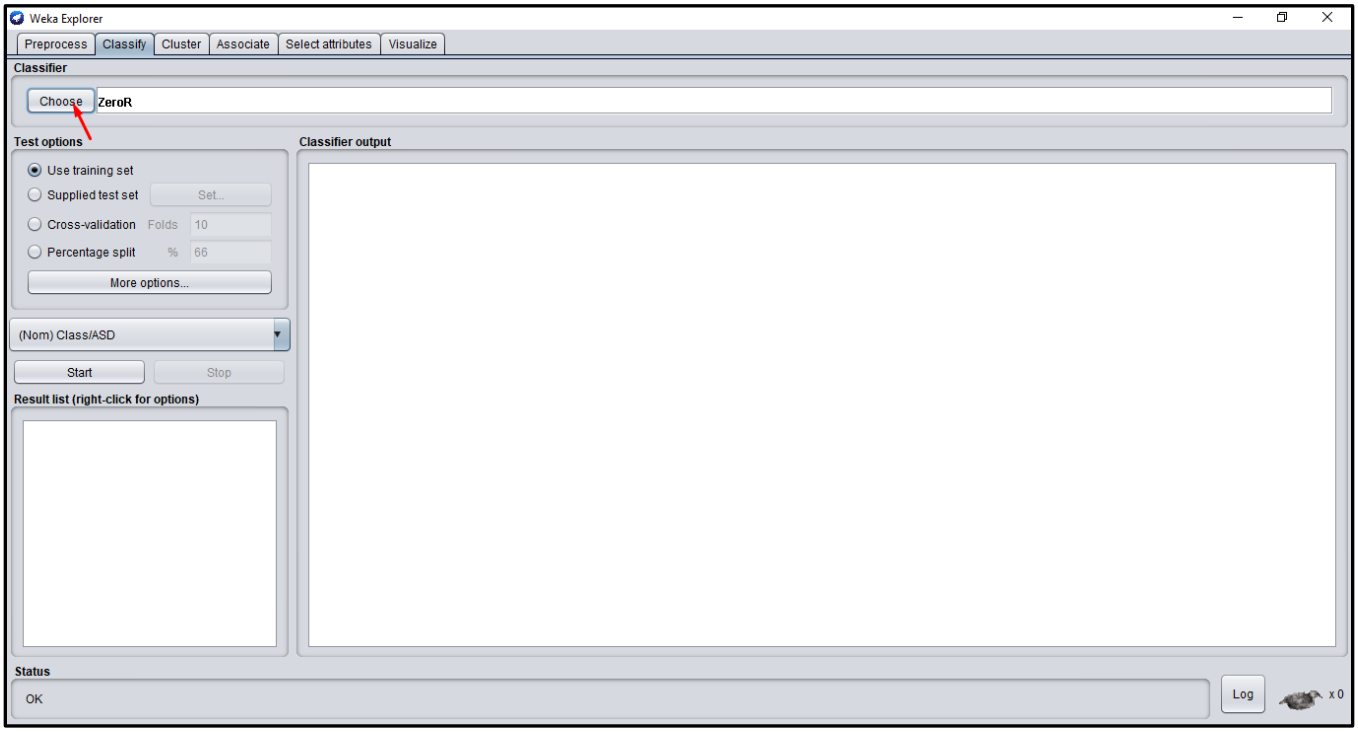
- Check the attributes to be deleted, from the list of Attributes in the Data prepocess Tab, and then click **Remove**

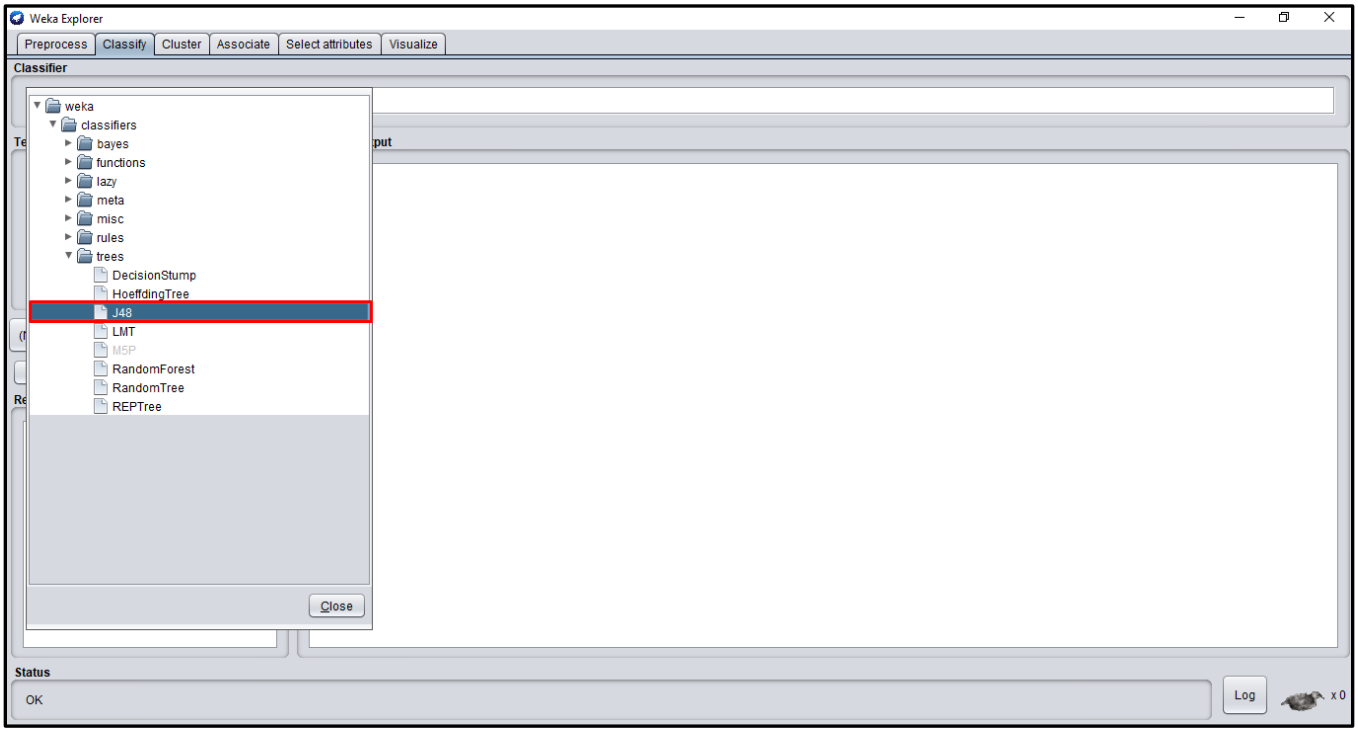


- Now the dataset is ready to perform classification algorithms

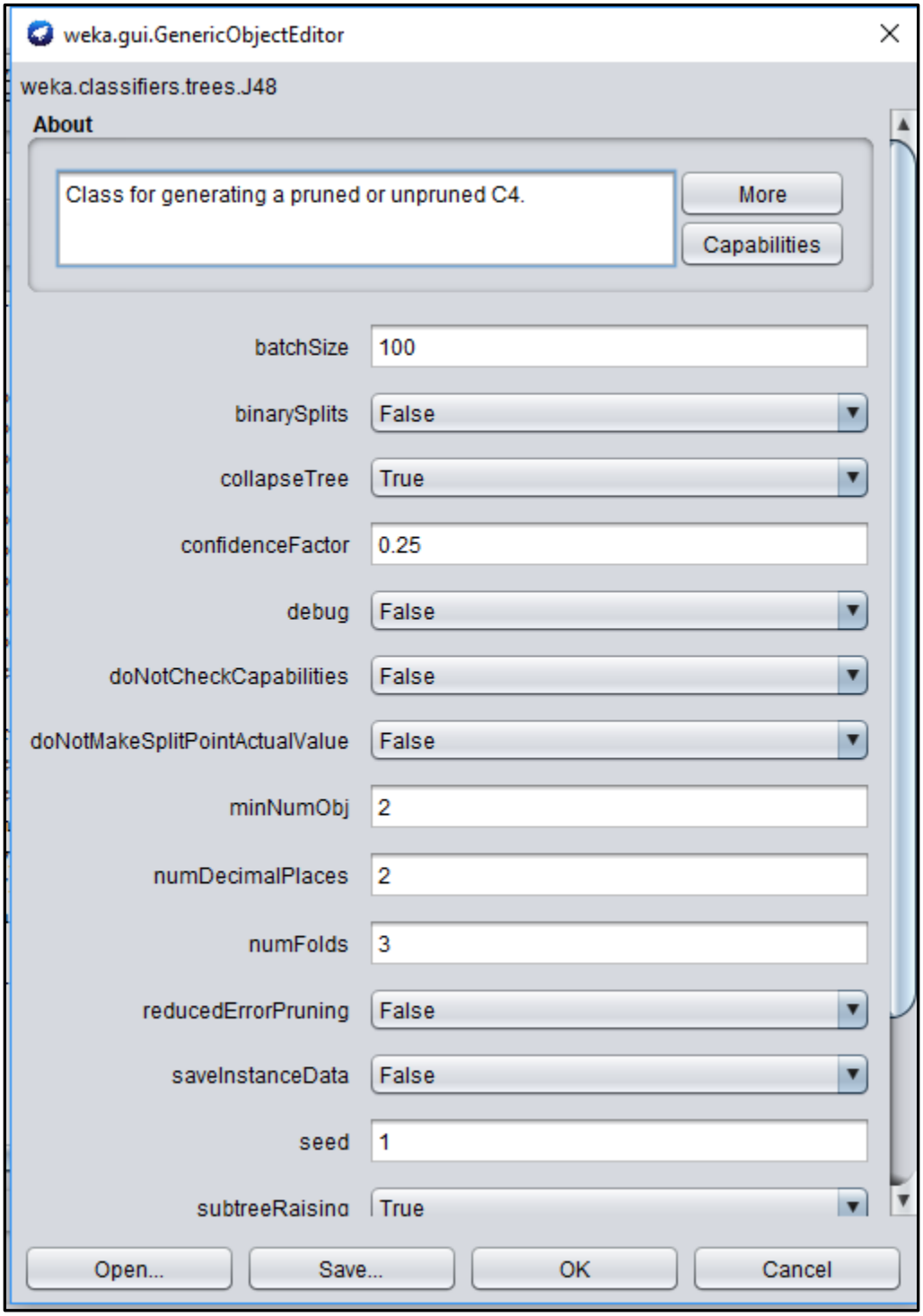
7.3. Implementation of Decision Tree Algorithm using WEKA

- Click **Classify** Tab on the top of the Weka Explorer window. Click **Choose** button under Classifier. The drop-down lists all classifier algorithms. Choose **J48** from the Trees folder as shown below. J48 is an open source Java implementation of the C4.5 algorithm in the Weka data mining tool.

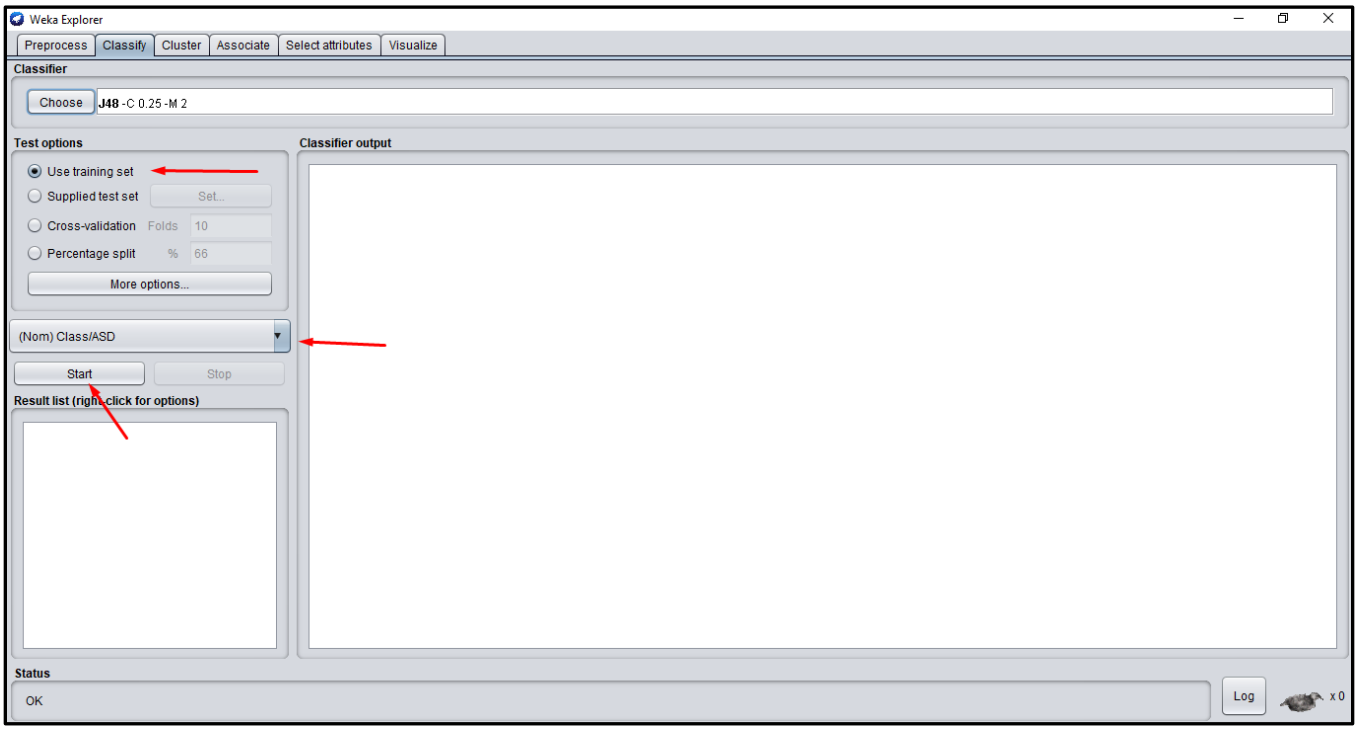




- Left click the field of Classifier. The property window of Decision Tree opens. Click OK to save all the default settings. Here the confidence is 0.25



- Choose the Test options as “**Use Training Dataset**” and select “**Nom Class/ASD**” as highlighted in the screenshot below. Once the above settings are done, click on **Start** button to implement the J48 Classifier algorithm on the dataset .The output is displayed in the Classifier Output window.



7.3.1. Output/Analysis by Decision Tree Algorithm

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     adult-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-10-weka.filters
Instances:    608
Attributes:   17
              A1_Score
              A2_Score
              A3_Score
              A4_Score
              A5_Score
              A6_Score
              A7_Score
              A8_Score
              A9_Score
              A10_Score
              age
              gender
              ethnicity
              jundice
              austim
              contry_of_res
              Class/ASD
Test mode:    evaluate on training data

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

J48 pruned tree
------------------

A9_Score = 0
|   A5_Score = 0: NO (246.0/1.0)
|   A5_Score = 1
|   |   A2_Score = 0: NO (92.0/7.0)
|   |   A2_Score = 1
|   |   |   A10_Score = 0: NO (24.0/3.0)
|   |   |   A10_Score = 1
|   |   |   |   A8_Score = 0: NO (8.0)
|   |   |   |   A8_Score = 1: YES (30.0/7.0)
A9_Score = 1
```

```
A9_Score = 1
|   A5_Score = 0
|   |   A3_Score = 0: NO (22.0)
|   |   A3_Score = 1
|   |   |   A7_Score = 0
|   |   |   |   austim = no: NO (13.0/1.0)
|   |   |   |   austim = yes: YES (2.0)
|   |   |   A7_Score = 1: YES (5.0)
|   A5_Score = 1
|   |   A1_Score = 0
|   |   |   A7_Score = 0: NO (10.0/1.0)
|   |   |   A7_Score = 1
|   |   |   |   A6_Score = 0: NO (3.0/1.0)
|   |   |   |   A6_Score = 1: YES (5.0)
|   |   A1_Score = 1
|   |   |   A3_Score = 0
|   |   |   |   A6_Score = 0
|   |   |   |   |   A4_Score = 0: NO (9.0)
|   |   |   |   |   A4_Score = 1
|   |   |   |   |   |   A8_Score = 0: NO (2.0)
|   |   |   |   |   |   A8_Score = 1: YES (6.0/1.0)
|   |   |   |   A6_Score = 1
|   |   |   |   |   A10_Score = 0
|   |   |   |   |   |   austim = no: NO (2.0)
|   |   |   |   |   |   austim = yes: YES (2.0)
|   |   |   |   |   A10_Score = 1: YES (16.0)
|   |   |   A3_Score = 1: YES (111.0/3.0)

Number of Leaves  :     19

Size of the tree :      37


Time taken to build model: 0.04 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.02 seconds
```

```
=== Summary ===

Correctly Classified Instances         583               95.8882 %
Incorrectly Classified Instances        25                4.1118 %
Kappa statistic                          0.9009
Mean absolute error                      0.0714
Root mean squared error                  0.1889
Relative absolute error                 17.1121 %
Root relative squared error             41.3802 %
Total Number of Instances              608

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.974    0.078    0.968      0.974    0.971      0.901    0.981     0.988     NO
                 0.922    0.026    0.938      0.922    0.930      0.901    0.981     0.952     YES
Weighted Avg.    0.959    0.062    0.959      0.959    0.959      0.901    0.981     0.977

=== Confusion Matrix ===

   a   b   <-- classified as
 417  11 |   a = NO
  14 166 |   b = YES
```

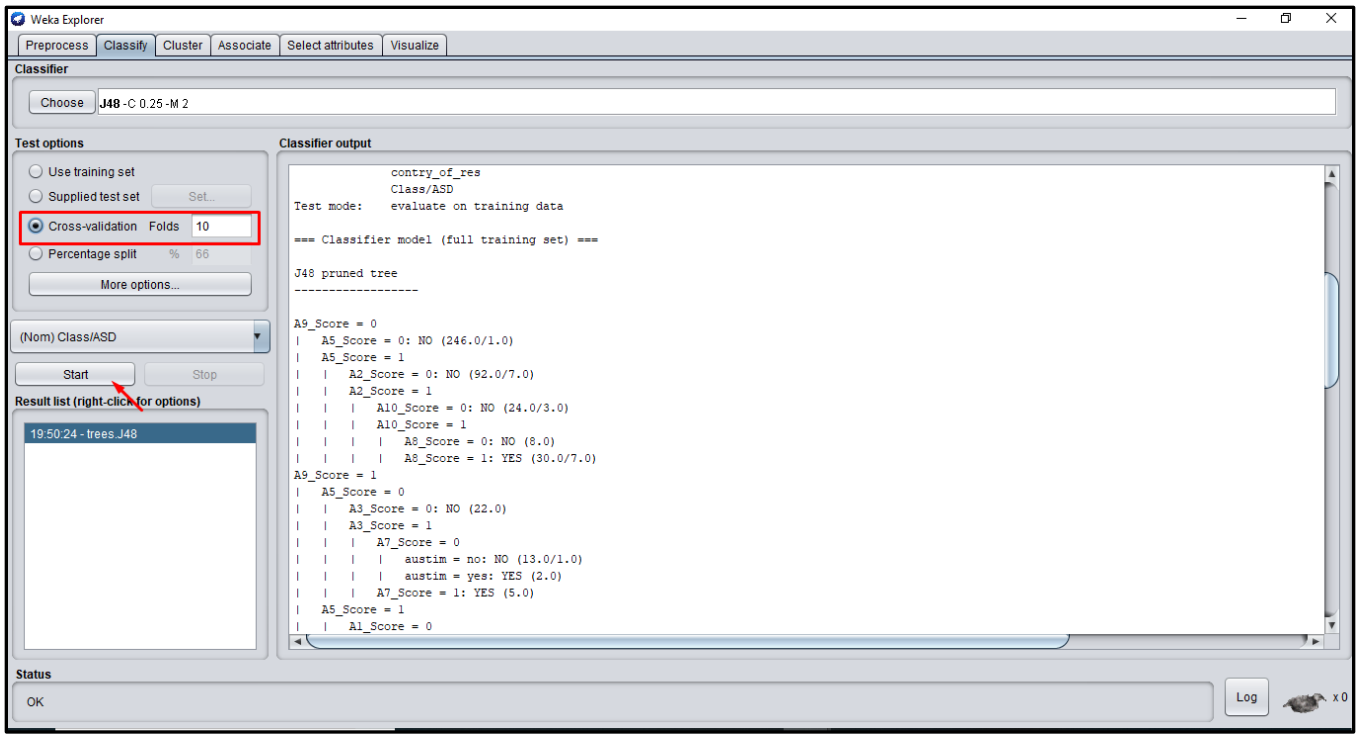
Accuracy = (TP + TN)/Total
= 583/608
=95.8882

The Accuracy of the algorithm using the Training set is 95.882% and Precision = 417/ (417+14) = 96.75%

7.3.2. 10-fold Cross validation of Decision Tree Algorithm

Cross-validation, a standard evaluation technique, is a systematic way of running repeated percentage splits. Divide a dataset into 10 pieces ("folds"), then hold out each piece in turn for testing and train on the remaining 9 together. This gives 10 evaluation results, which are averaged. Having done 10-fold cross-validation and computed the evaluation results, Weka invokes the learning algorithm a final (11th) time on the entire dataset to obtain the model that it prints out.10-fold Cross Validation is suitable for classifiers that predict labels (positive or negative).

- To perform Cross- Validation of decision- tree algorithm - Under the Classify Tab, select the radio button – **Cross-validation** and input "**10**" as value for "**Folds**" field under Test options.
- Select "**(Nom) y**" as highlighted below. click on "**Start**" button. On clicking Start button, algorithm will run on the data set and will provide output under Classifier output window as shown below


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
J48 -C 0.25 -M 2
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Nom) Class/ASD
Start
Stop
Result list (right-click for options)
19:50:24 - trees.J48
Classifier output
contry_of_res
Class/ASD
Test mode: evaluate on training data
=== Classifier model (full training set) ===
J48 pruned tree
A9_Score = 0
| A5_Score = 0: NO (246.0/1.0)
| A5_Score = 1
| | A2_Score = 0: NO (92.0/7.0)
| | A2_Score = 1
| | | A10_Score = 0: NO (24.0/3.0)
| | | A10_Score = 1
| | | | A8_Score = 0: NO (8.0)
| | | | A8_Score = 1: YES (30.0/7.0)
A9_Score = 1
| A5_Score = 0
| | A3_Score = 0: NO (22.0)
| | A3_Score = 1
| | | A7_Score = 0
| | | | austim = no: NO (13.0/1.0)
| | | | austim = yes: YES (2.0)
| | | A7_Score = 1: YES (5.0)
| A5_Score = 1
| | A1_Score = 0
Status
OK
Log
x 0

7.3.3. Cross validation results

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     adult-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-10-weka.filt
Instances:    608
Attributes:   17
              A1_Score
              A2_Score
              A3_Score
              A4_Score
              A5_Score
              A6_Score
              A7_Score
              A8_Score
              A9_Score
              A10_Score
              age
              gender
              ethnicity
              jundice
              austim
              contry_of_res
              Class/ASD
Test mode:    10-fold cross-validation

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

J48 pruned tree
------------------

A9_Score = 0
|   A5_Score = 0: NO (246.0/1.0)
|   A5_Score = 1
|   |   A2_Score = 0: NO (92.0/7.0)
|   |   A2_Score = 1
|   |   |   A10_Score = 0: NO (24.0/3.0)
|   |   |   A10_Score = 1
|   |   |   |   A8_Score = 0: NO (8.0)
|   |   |   |   A8_Score = 1: YES (30.0/7.0)
A9_Score = 1
```

```
A9_Score = 1
|   A5_Score = 0
|   |   A3_Score = 0: NO (22.0)
|   |   A3_Score = 1
|   |   |   A7_Score = 0
|   |   |   |   austim = no: NO (13.0/1.0)
|   |   |   |   austim = yes: YES (2.0)
|   |   |   A7_Score = 1: YES (5.0)
|   A5_Score = 1
|   |   A1_Score = 0
|   |   |   A7_Score = 0: NO (10.0/1.0)
|   |   |   A7_Score = 1
|   |   |   |   A6_Score = 0: NO (3.0/1.0)
|   |   |   |   A6_Score = 1: YES (5.0)
|   |   A1_Score = 1
|   |   |   A3_Score = 0
|   |   |   |   A6_Score = 0
|   |   |   |   |   A4_Score = 0: NO (9.0)
|   |   |   |   |   A4_Score = 1
|   |   |   |   |   |   A8_Score = 0: NO (2.0)
|   |   |   |   |   |   A8_Score = 1: YES (6.0/1.0)
|   |   |   |   A6_Score = 1
|   |   |   |   |   A10_Score = 0
|   |   |   |   |   |   austim = no: NO (2.0)
|   |   |   |   |   |   austim = yes: YES (2.0)
|   |   |   |   |   A10_Score = 1: YES (16.0)
|   |   |   A3_Score = 1: YES (111.0/3.0)

Number of Leaves  :     19

Size of the tree :      37


Time taken to build model: 0.01 seconds
```

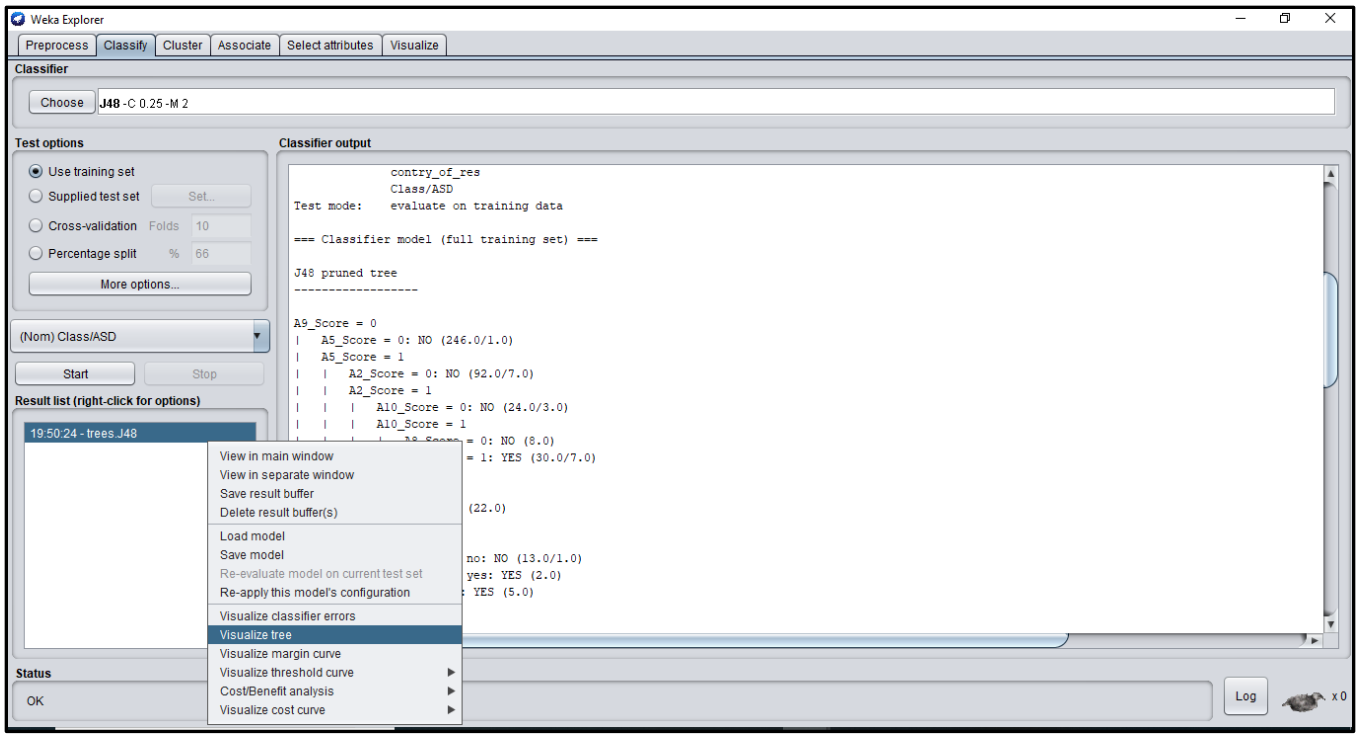
```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         547               89.9671 %
Incorrectly Classified Instances        61               10.0329 %
Kappa statistic                          0.7589
Mean absolute error                      0.1162
Root mean squared error                  0.2933
Relative absolute error                 27.8604 %
Root relative squared error             64.2378 %
Total Number of Instances              608

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0.930     0.172     0.928       0.930   0.929      0.759    0.924     0.952     NO
               0.828     0.070     0.832       0.828   0.830      0.759    0.924     0.798     YES
Weighted Avg.  0.900     0.142     0.900       0.900   0.900      0.759    0.924     0.906

=== Confusion Matrix ===

   a   b   <-- classified as
 398  30 |   a = NO
  31 149 |   b = YES
```

Accuracy = (TP + TN)/Total
= 547/608
=89.9671

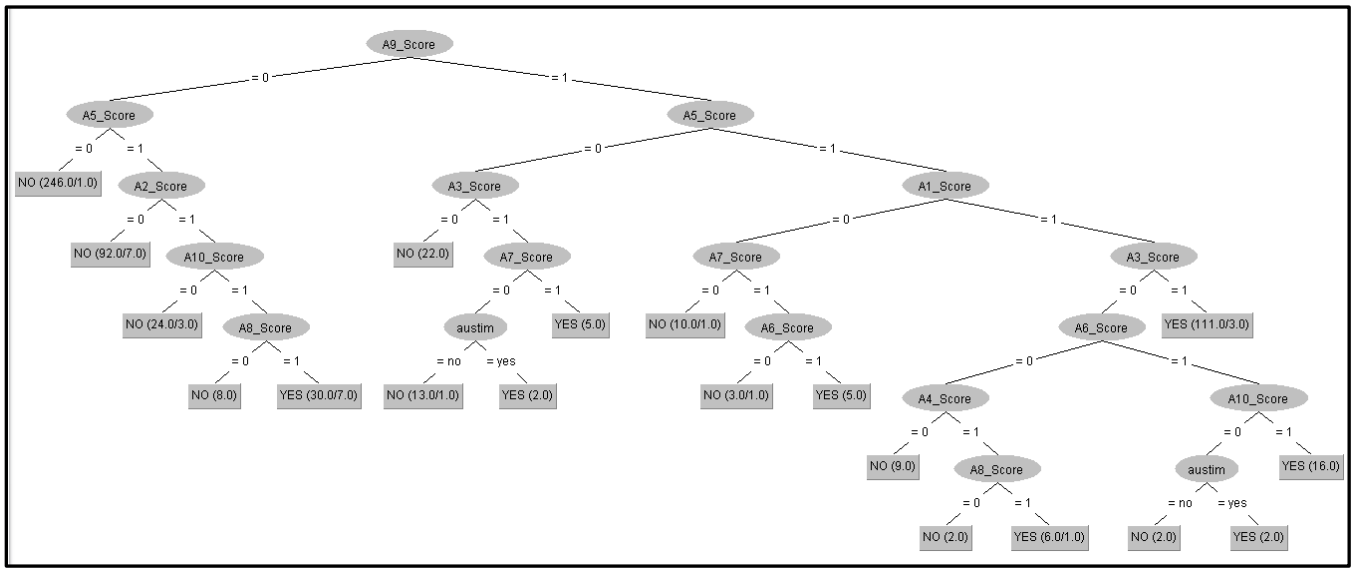
- **The accuracy of the decision tree algorithm using 10-fold cross validation method is 89.9671% and the precision of the predicted outcome is 92.8%**

7.3.4. Decision Tree Visualization

- To visualize the decision tree we built, right -click on the Result list item for J48.Select the option "**Visualize tree**". Right click on the Tree view and choose "Fit to Screen" to adjust the output tree.



- The decision tree generated for Autism Dataset is as shown below



7.3.5. Threshold Curve - Decision Tree Algorithm

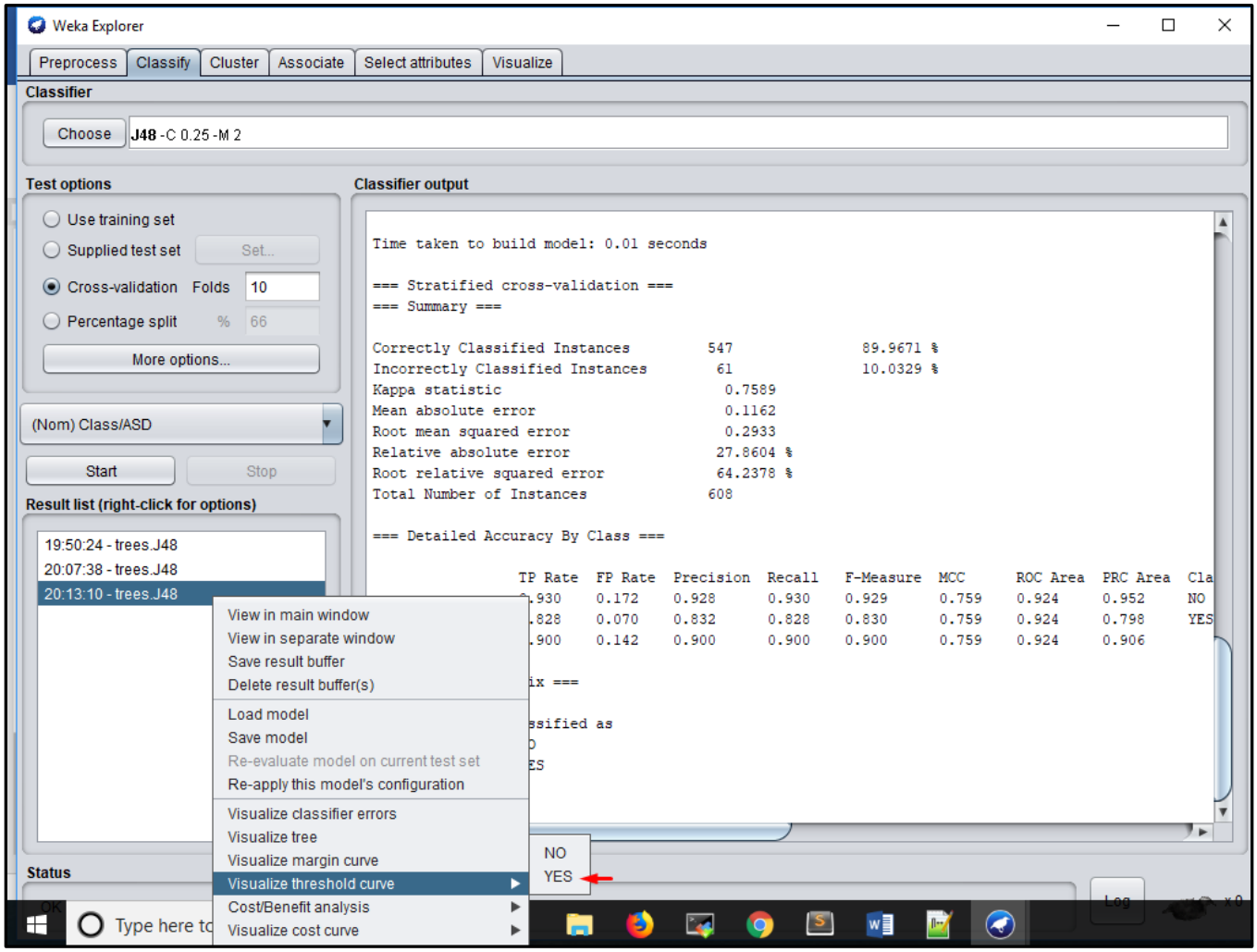
ROC Curve

Receiver Operating Characteristic (ROC) is a performance graphing method. It is a plot of True positive (TPR) and False positive rates (FPR). It is used for evaluating data mining schemes and comparing the relative performance among different classifiers. TPR is plotted on the Y axis and FPR on the X axis. It depicts the relative trade offs between benefits (TP) and costs (FP)[11].

AUC (Area under ROC Curve)

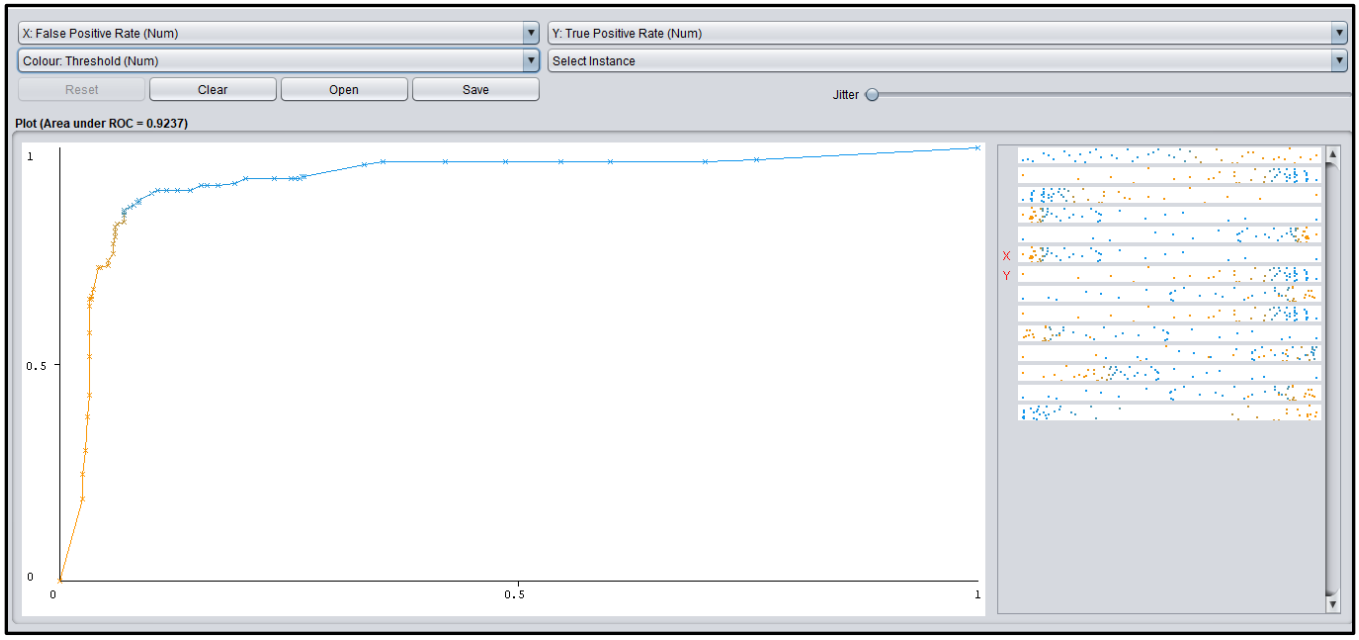
AUC is often used to compare classifiers. The bigger AUC, the better. AUC can be calculated by summing up the areas of the small rectangles underneath ROC. Classifier A is better than classifier B if A's AUC is larger than B's AUC. A perfect classifier has an AUC of 1.

- To visualize the Threshold curve, right-click on the J48 option in the Results list pane and select "**view Threshold Curve**" and click on YES.



[11] https://web.uvic.ca/~maryam/DMSpring94/Slides/9_roc.pdf

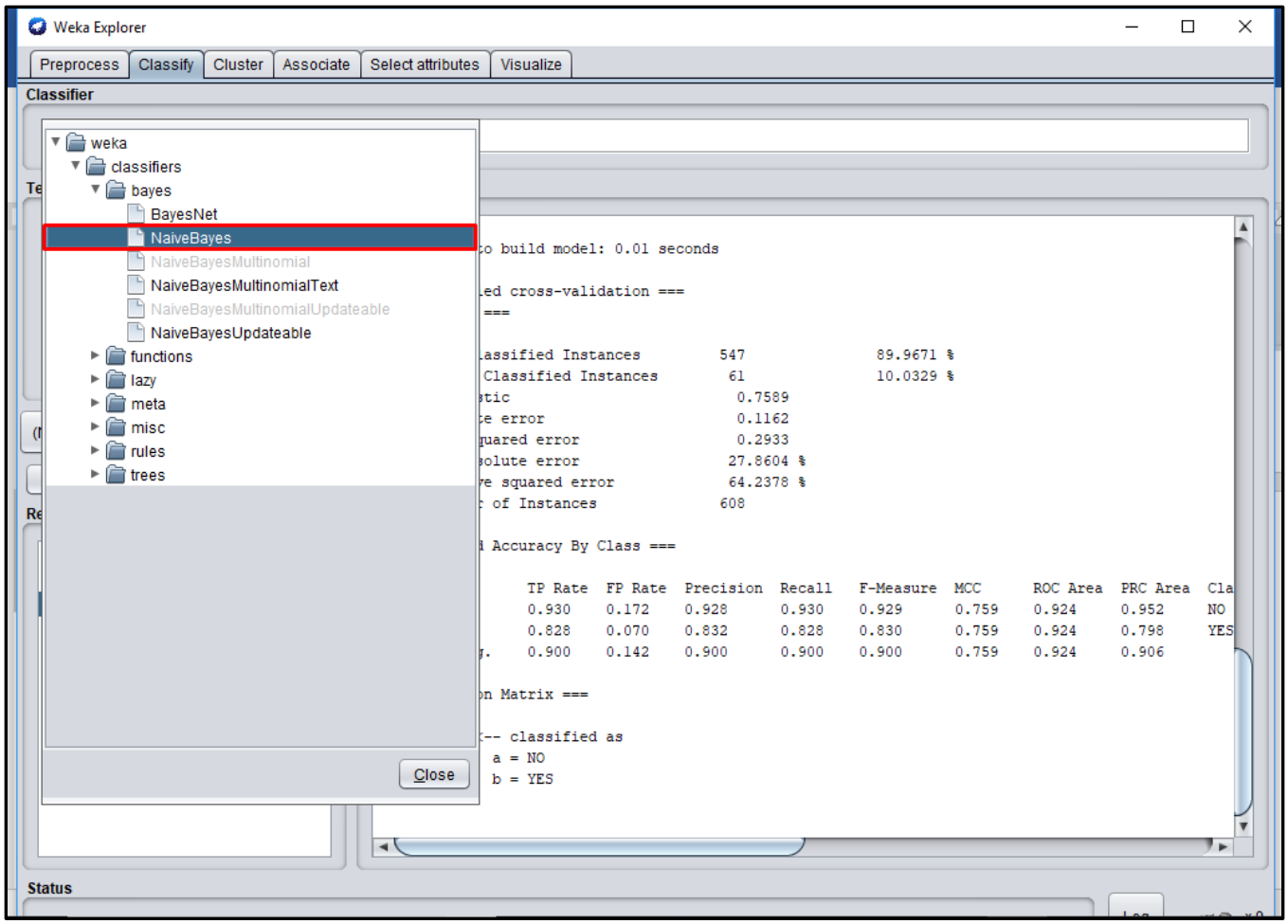
- The Threshold curve for current dataset is generated as given below. The True positive rate is plotted on the Y-axis and False Positive Rate on the X-axis



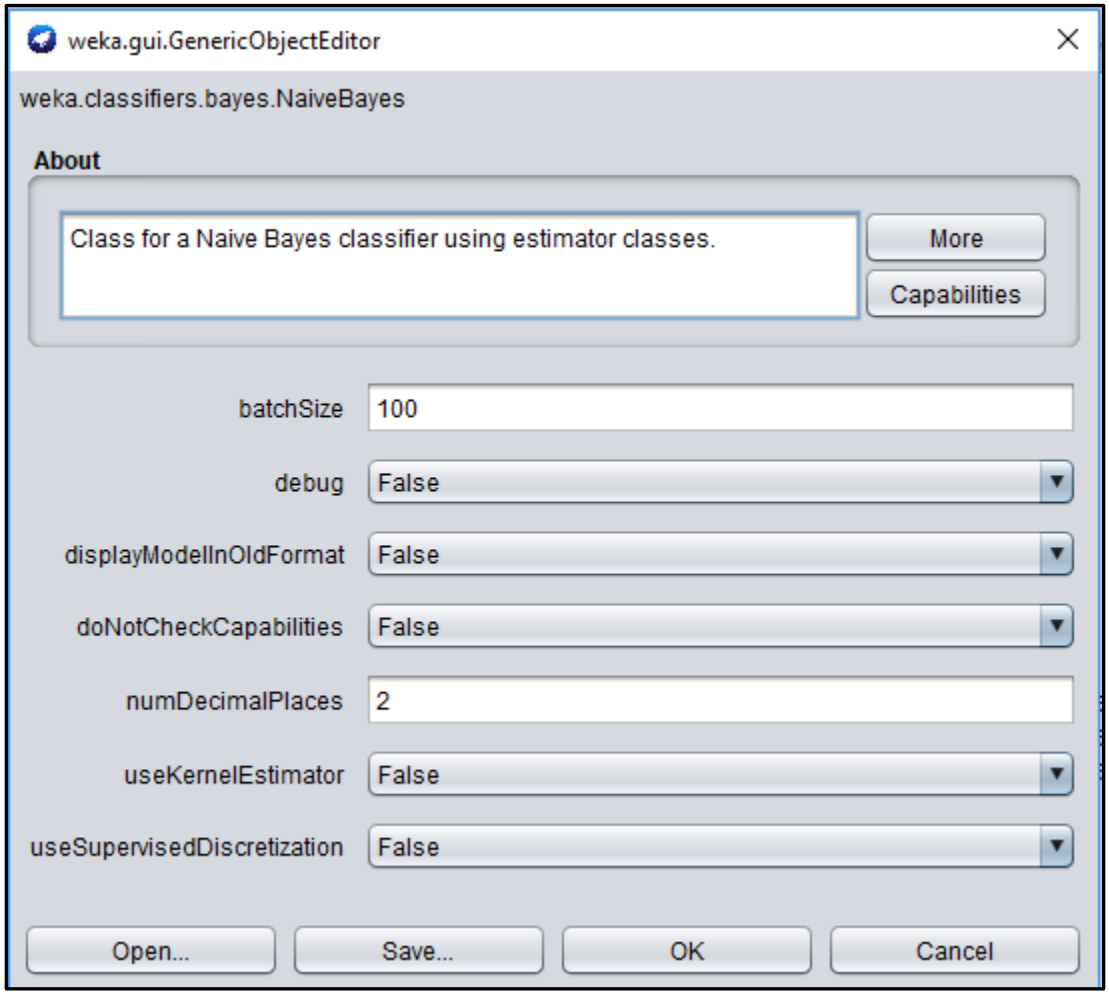
Area under ROC (AUC) = 0.9237

7.4. Implementation of Naive Bayes using WEKA

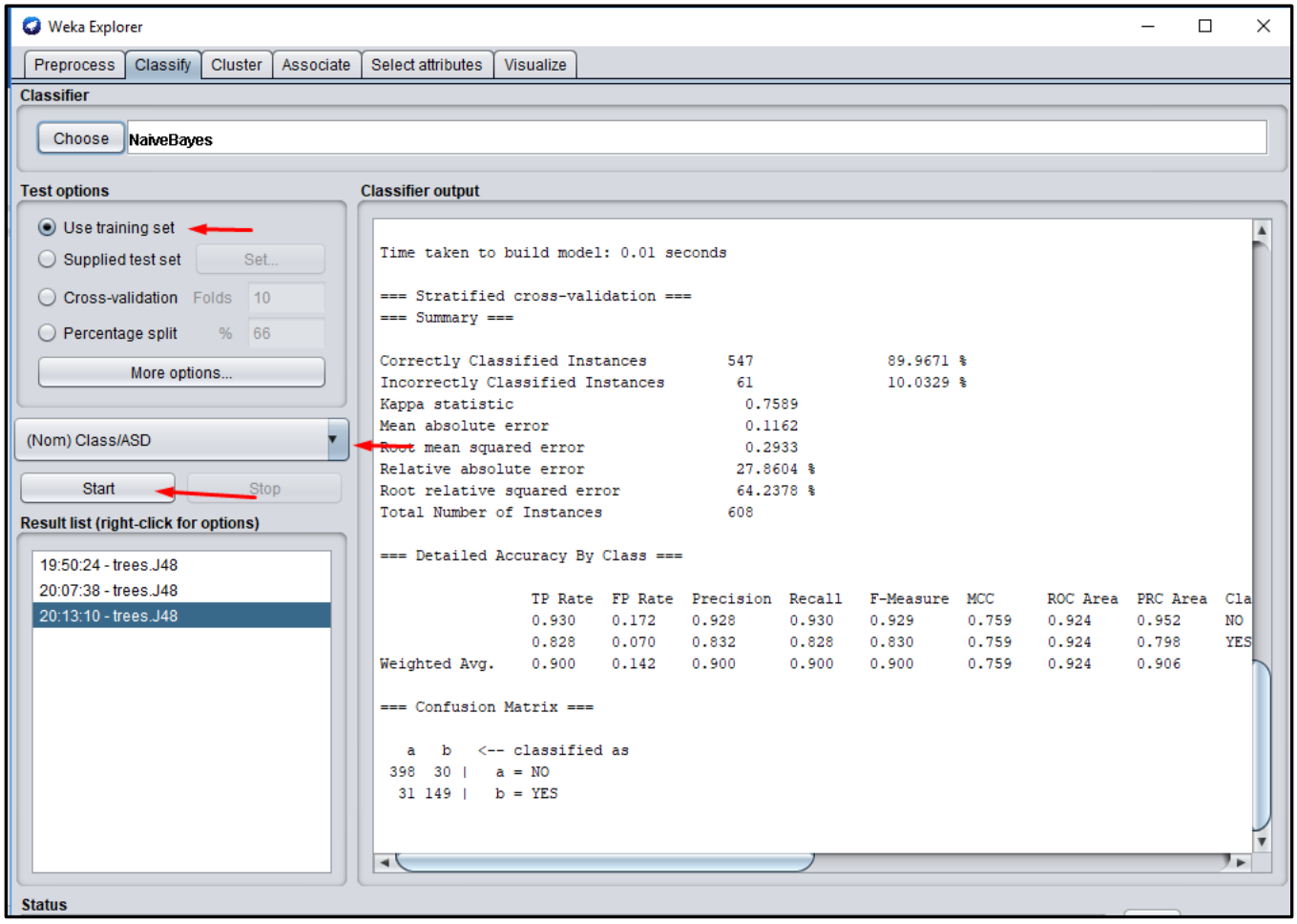
- To perform the second algorithm – Naive Bayes on the same dataset, Click **Classify** Tab on the top of the Weka Explorer window. Click "**Choose**" button under Classifier. The drop-down lists all classifier algorithms. Select "**Naive Bayes**" from the Bayes folder as shown below.


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
weka
classifiers
bayes
BayesNet
NaiveBayes
NaiveBayesMultinomial
NaiveBayesMultinomialText
NaiveBayesMultinomialUpdateable
NaiveBayesUpdateable
functions
lazy
meta
misc
rules
trees
Close
o build model: 0.01 seconds
ed cross-validation ===
===
assified Instances 547 89.9671 %
Classified Instances 61 10.0329 %
tic 0.7589
e error 0.1162
uared error 0.2933
olute error 27.8604 %
e squared error 64.2378 %
of Instances 608
Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Cla
0.930 0.172 0.928 0.930 0.929 0.759 0.924 0.952 NO
0.828 0.070 0.832 0.828 0.830 0.759 0.924 0.798 YES
0.900 0.142 0.900 0.900 0.900 0.759 0.924 0.906
n Matrix ===
-- classified as
a = NO
b = YES
Status

- Left click the field of Classifier. The property window of Naive Bayes opens. Click **OK** to save all the default settings.



- Choose the Test options as "**Use Training Dataset**" and select "**Nom Class/ASD**" as highlighted in the screenshot below. Once the above settings are done, click "**Start**" button to implement the Naive Bayes Classifier algorithm on the dataset. The output is displayed in the Classifier Output window.



7.4.1. Output/Analysis using Naive Bayes

- The output of performing Naïve Bayes algorithm classification on Autism Dataset is as given below

```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     adult-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-10-weka.
Instances:    608
Attributes:   17
              A1_Score
              A2_Score
              A3_Score
              A4_Score
              A5_Score
              A6_Score
              A7_Score
              A8_Score
              A9_Score
              A10_Score
              age
              gender
              ethnicity
              jundice
              austim
              contry_of_res
              Class/ASD
Test mode:    evaluate on training data

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

Naive Bayes Classifier

                          Class
Attribute                   NO     YES
                          (0.7)   (0.3)
=======================================
A1_Score
  0                        148.0    12.0
  1                        282.0   170.0
  [total]                  430.0   182.0

A2_Score
  0                        274.0    50.0
```

A2_Score		
0	274.0	50.0
1	156.0	132.0
[total]	430.0	182.0
A3_Score		
0	283.0	34.0
1	147.0	148.0
[total]	430.0	182.0
A4_Score		
0	271.0	22.0
1	159.0	160.0
[total]	430.0	182.0
A5_Score		
0	280.0	10.0
1	150.0	172.0
[total]	430.0	182.0
A6_Score		
0	375.0	48.0
1	55.0	134.0
[total]	430.0	182.0
A7_Score		
0	294.0	55.0
1	136.0	127.0
[total]	430.0	182.0
A8_Score		
0	176.0	29.0
1	254.0	153.0
[total]	430.0	182.0
A9_Score		
0	367.0	35.0
1	63.0	147.0
[total]	430.0	182.0

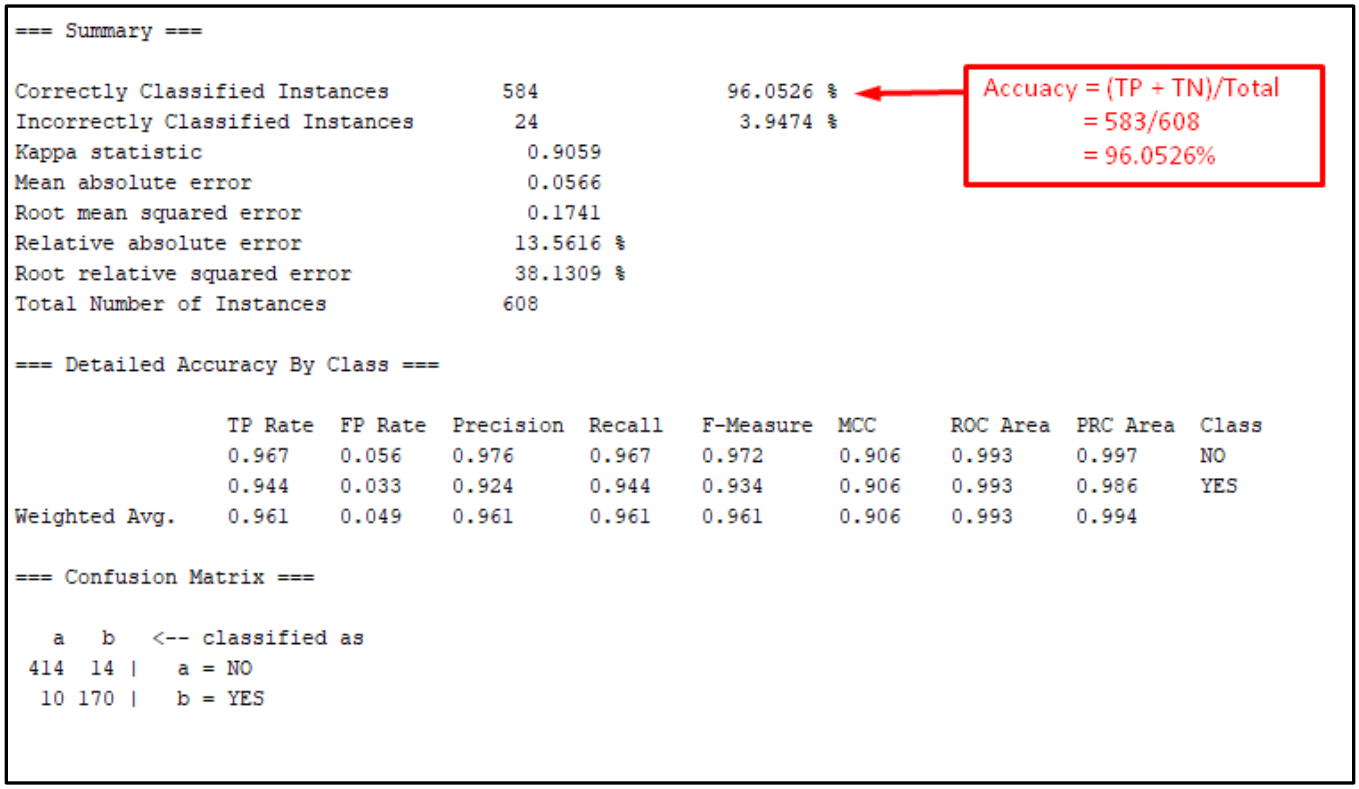
age		
mean	28.8284	31.6479
std. dev.	9.1018	10.5222
weight sum	428	180
precision	1.0682	1.0682
gender		
f	191.0	98.0
m	239.0	84.0
[total]	430.0	182.0
ethnicity		
White-European	125.0	110.0
Latino	11.0	11.0
Others	22.0	10.0
Black	26.0	19.0
Asian	108.0	17.0
Middle Eastern	85.0	9.0
Pasifika	11.0	2.0
South Asian	34.0	4.0
Hispanic	9.0	6.0
Turkish	6.0	2.0
others	2.0	1.0
[total]	439.0	191.0
jundice		
no	398.0	153.0
yes	32.0	29.0
[total]	430.0	182.0
austim		
no	384.0	140.0
yes	45.0	42.0
[total]	429.0	182.0
contry_of_res		
United States	60.0	54.0
Brazil	4.0	6.0
Spain	2.0	3.0
Egypt	3.0	1.0

New Zealand	63.0	14.0
Bahamas	1.0	2.0
Burundi	2.0	1.0
Austria	1.0	4.0
Argentina	1.0	1.0
Jordan	6.0	1.0
Ireland	4.0	3.0
United Arab Emirates	65.0	4.0
Afghanistan	9.0	3.0
Lebanon	1.0	1.0
United Kingdom	48.0	30.0
South Africa	1.0	3.0
Italy	2.0	5.0
Pakistan	3.0	1.0
Bangladesh	3.0	2.0
Chile	2.0	1.0
France	7.0	6.0
China	1.0	2.0
Australia	16.0	13.0
Canada	6.0	11.0
Saudi Arabia	3.0	1.0
Netherlands	7.0	5.0
Romania	3.0	2.0
Sweden	1.0	3.0
Tonga	2.0	1.0
Oman	2.0	1.0
India	76.0	7.0
Philippines	5.0	1.0
Sri Lanka	15.0	1.0
Sierra Leone	1.0	2.0
Ethiopia	3.0	1.0
Viet Nam	5.0	2.0
Iran	3.0	1.0
Costa Rica	2.0	1.0
Germany	3.0	3.0
Mexico	5.0	5.0
Russia	2.0	2.0
Armenia	3.0	1.0
Iceland	3.0	1.0
Nicaragua	2.0	1.0

```
  Nicaragua                    2.0     1.0
  Hong Kong                    1.0     1.0
  Japan                        1.0     1.0
  Ukraine                      2.0     1.0
  Kazakhstan                   1.0     1.0
  AmericanSamoa                2.0     2.0
  Uruguay                      1.0     2.0
  Serbia                       2.0     1.0
  Portugal                     2.0     1.0
  Malaysia                     2.0     5.0
  Ecuador                      2.0     1.0
  Niger                        2.0     1.0
  Belgium                      3.0     2.0
  Bolivia                      2.0     1.0
  Aruba                        2.0     1.0
  Finland                      1.0     2.0
  Turkey                       2.0     1.0
  Nepal                        1.0     2.0
  Indonesia                    2.0     1.0
  Angola                       2.0     1.0
  Azerbaijan                   1.0     1.0
  Iraq                         1.0     1.0
  Czech Republic               2.0     1.0
  Cyprus                       1.0     2.0
  [total]                    495.0   247.0



Time taken to build model: 0.01 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.03 seconds
```

```
=== Summary ===

Correctly Classified Instances         584               96.0526 %
Incorrectly Classified Instances        24                3.9474 %
Kappa statistic                          0.9059
Mean absolute error                      0.0566
Root mean squared error                  0.1741
Relative absolute error                 13.5616 %
Root relative squared error             38.1309 %
Total Number of Instances              608

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
                 0.967    0.056     0.976      0.967     0.972      0.906    0.993     0.997     NO
                 0.944    0.033     0.924      0.944     0.934      0.906    0.993     0.986     YES
Weighted Avg.    0.961    0.049     0.961      0.961     0.961      0.906    0.993     0.994

=== Confusion Matrix ===

   a   b   <-- classified as
 414  14 |   a = NO
  10 170 |   b = YES
```

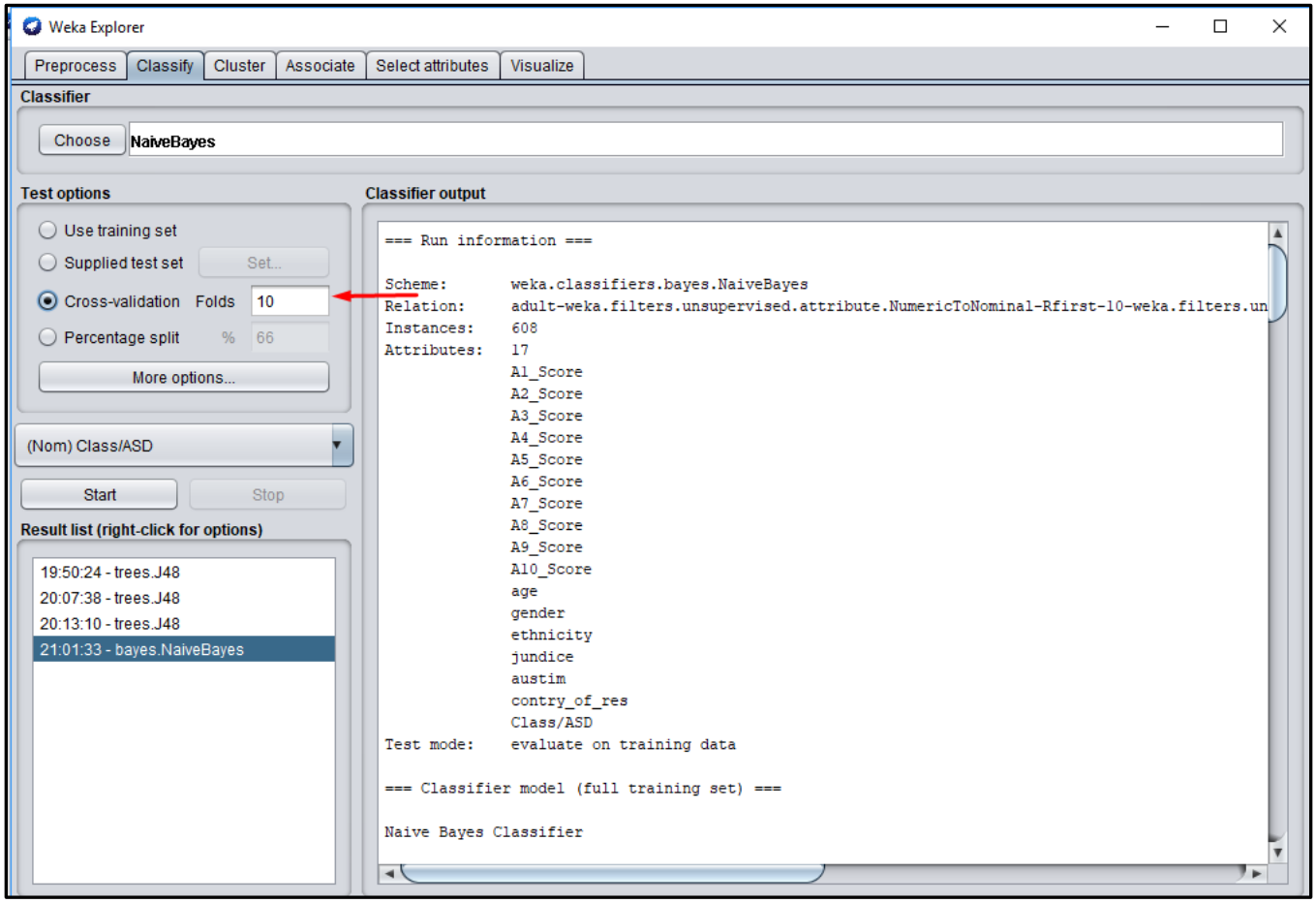
- **Output Summary**



The Accuracy of Naive Bayes algorithm using the Training set is 95.0526 % and Precision = 97.6%

7.4.2. 10-Fold cross Validation

- To perform Cross- Validation of Naive Bayes algorithm: Under the Classify Tab, select the radio button – **Cross-validation** and input "**10**" as value for "**Folds**" field under Test options.
- Select "**(Nom) y**" as highlighted below. click on "**Start**" button. On clicking Start button, algorithm will run on the data set and will provide output under Classifier output window as shown below



7.4.3. Cross Validation Results

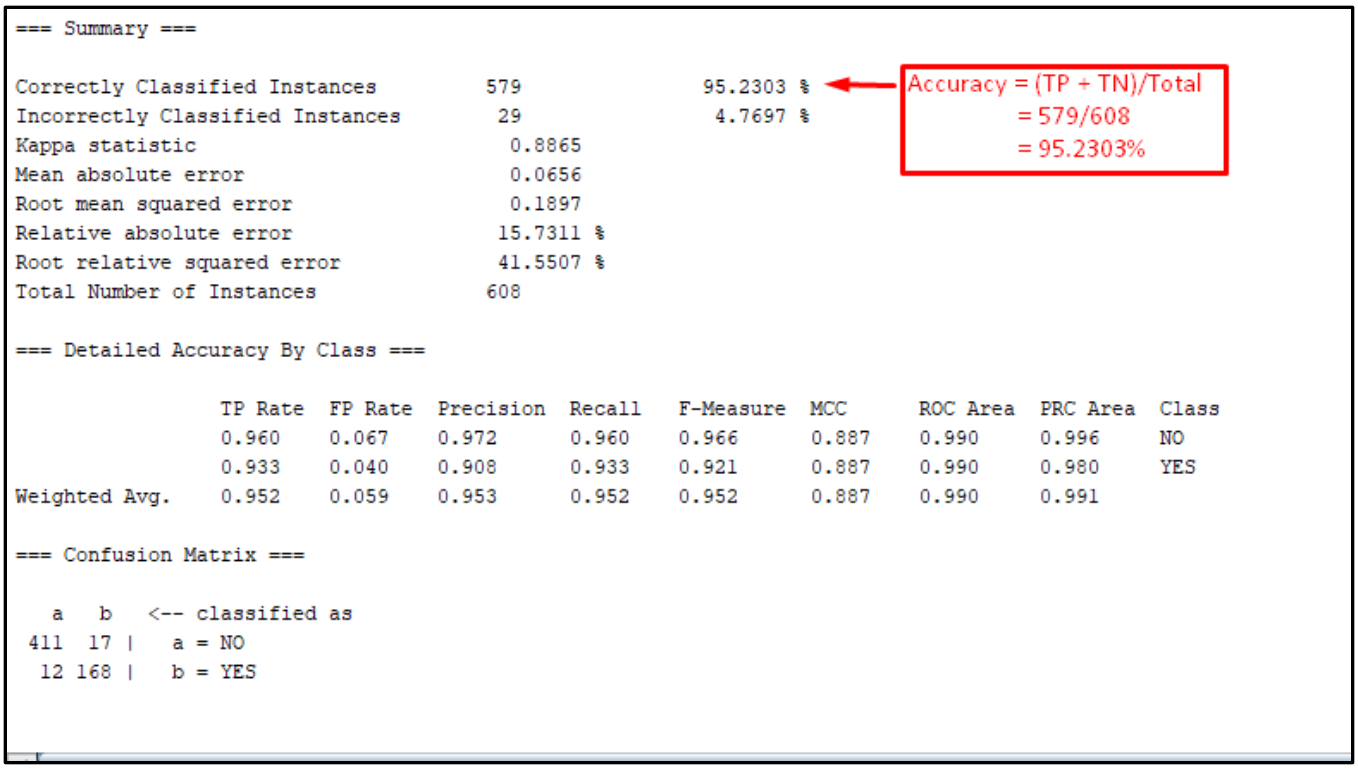
```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     adult-weka.filters.unsupervised.attribute.NumericToNominal-Rfir
Instances:    608
Attributes:   17
              A1_Score
              A2_Score
              A3_Score
              A4_Score
              A5_Score
              A6_Score
              A7_Score
              A8_Score
              A9_Score
              A10_Score
              age
              gender
              ethnicity
              jundice
              austim
              contry_of_res
              Class/ASD
Test mode:    10-fold cross-validation

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

Naive Bayes Classifier

                          Class
Attribute                   NO     YES
                          (0.7)   (0.3)
=========================================
A1_Score
  0                        148.0    12.0
  1                        282.0   170.0
  [total]                  430.0   182.0

A2_Score
  0                        274.0    50.0
```

1	156.0	132.0
[total]	430.0	182.0
A3_Score		
0	283.0	34.0
1	147.0	148.0
[total]	430.0	182.0
A4_Score		
0	271.0	22.0
1	159.0	160.0
[total]	430.0	182.0
A5_Score		
0	280.0	10.0
1	150.0	172.0
[total]	430.0	182.0
A6_Score		
0	375.0	48.0
1	55.0	134.0
[total]	430.0	182.0
A7_Score		
0	294.0	55.0
1	136.0	127.0
[total]	430.0	182.0
A8_Score		
0	176.0	29.0
1	254.0	153.0
[total]	430.0	182.0
A9_Score		
0	367.0	35.0
1	63.0	147.0
[total]	430.0	182.0
A10_Score		
0	224.0	22.0

1	206.0	160.0
[total]	430.0	182.0
age		
mean	28.8284	31.6479
std. dev.	9.1018	10.5222
weight sum	428	180
precision	1.0682	1.0682
gender		
f	191.0	98.0
m	239.0	84.0
[total]	430.0	182.0
ethnicity		
White-European	125.0	110.0
Latino	11.0	11.0
Others	22.0	10.0
Black	26.0	19.0
Asian	108.0	17.0
Middle Eastern	85.0	9.0
Pasifika	11.0	2.0
South Asian	34.0	4.0
Hispanic	9.0	6.0
Turkish	6.0	2.0
others	2.0	1.0
[total]	439.0	191.0
jundice		
no	398.0	153.0
yes	32.0	29.0
[total]	430.0	182.0
austim		
no	384.0	140.0
yes	45.0	42.0
[total]	429.0	182.0
contry_of_res		

contry_of_res		
United States	60.0	54.0
Brazil	4.0	6.0
Spain	2.0	3.0
Egypt	3.0	1.0
New Zealand	63.0	14.0
Bahamas	1.0	2.0
Burundi	2.0	1.0
Austria	1.0	4.0
Argentina	1.0	1.0
Jordan	6.0	1.0
Ireland	4.0	3.0
United Arab Emirates	65.0	4.0
Afghanistan	9.0	3.0
Lebanon	1.0	1.0
United Kingdom	48.0	30.0
South Africa	1.0	3.0
Italy	2.0	5.0
Pakistan	3.0	1.0
Bangladesh	3.0	2.0
Chile	2.0	1.0
France	7.0	6.0
China	1.0	2.0
Australia	16.0	13.0
Canada	6.0	11.0
Saudi Arabia	3.0	1.0
Netherlands	7.0	5.0
Romania	3.0	2.0
Sweden	1.0	3.0
Tonga	2.0	1.0
Oman	2.0	1.0
India	76.0	7.0
Philippines	5.0	1.0
Sri Lanka	15.0	1.0
Sierra Leone	1.0	2.0
Ethiopia	3.0	1.0
Viet Nam	5.0	2.0
Iran	3.0	1.0
Costa Rica	2.0	1.0
Germany	3.0	3.0

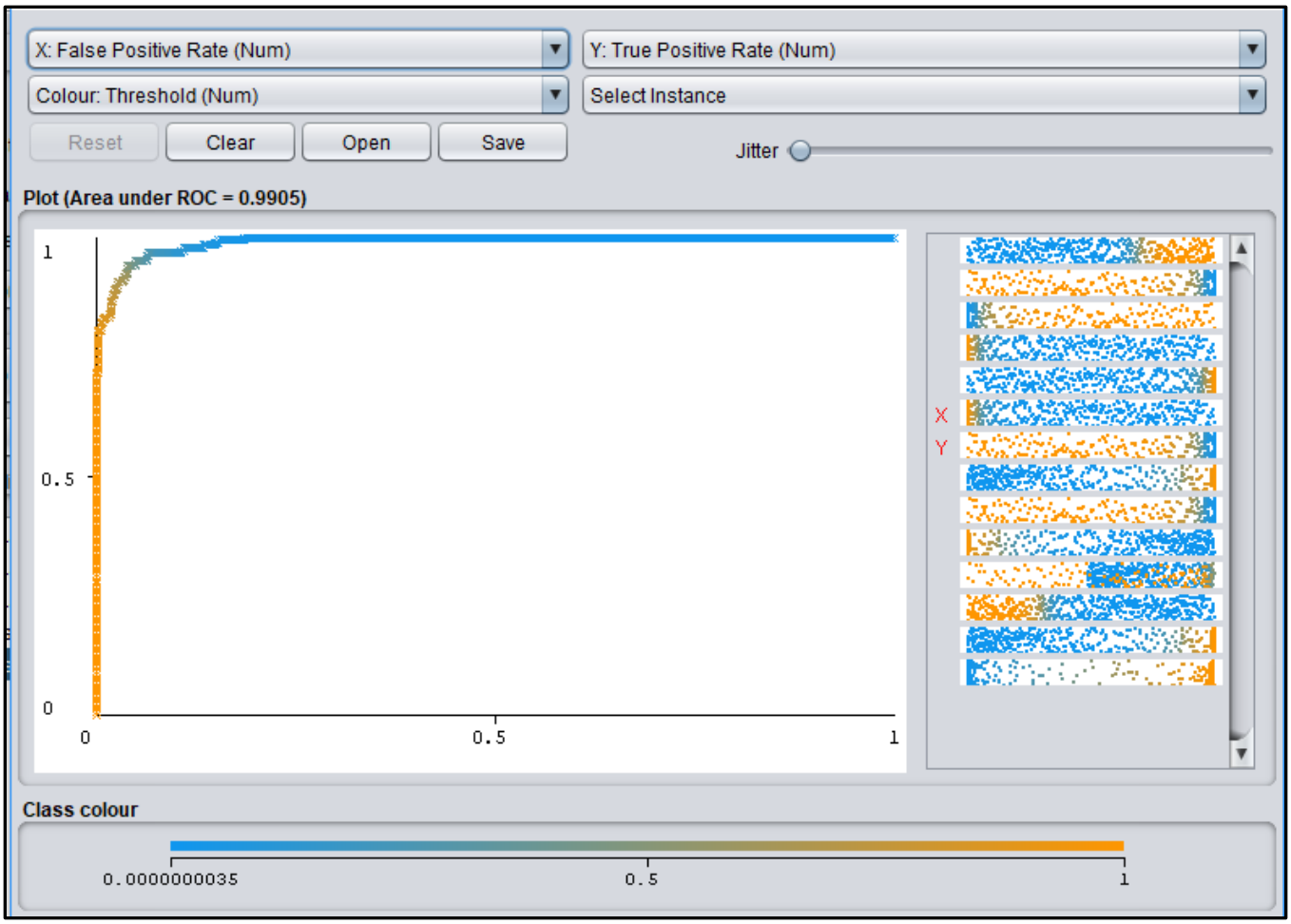
```
  Mexico                     5.0     5.0
  Russia                     2.0     2.0
  Armenia                    3.0     1.0
  Iceland                    3.0     1.0
  Nicaragua                  2.0     1.0
  Hong Kong                  1.0     1.0
  Japan                      1.0     1.0
  Ukraine                    2.0     1.0
  Kazakhstan                 1.0     1.0
  AmericanSamoa              2.0     2.0
  Uruguay                    1.0     2.0
  Serbia                     2.0     1.0
  Portugal                   2.0     1.0
  Malaysia                   2.0     5.0
  Ecuador                    2.0     1.0
  Niger                      2.0     1.0
  Belgium                    3.0     2.0
  Bolivia                    2.0     1.0
  Aruba                      2.0     1.0
  Finland                    1.0     2.0
  Turkey                     2.0     1.0
  Nepal                      1.0     2.0
  Indonesia                  2.0     1.0
  Angola                     2.0     1.0
  Azerbaijan                 1.0     1.0
  Iraq                       1.0     1.0
  Czech Republic             2.0     1.0
  Cyprus                     1.0     2.0
  [total]                  495.0   247.0



Time taken to build model: 0 seconds

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



The accuracy of the Naive Bayes algorithm using 10-fold cross validation method is 95.2303% and the precision of the predicted outcome is 97.2%

7.4.4. Threshold Curve – Naive Bayes

- To visualize the Threshold curve, right-click on the Naive Bayes option in the Results list pane and select "**view Threshold Curve**" and click on YES.
- The Threshold curve for current dataset is generated as given below. The True positive rate is plotted on the Y-axis and False Positive Rate on the X-axis

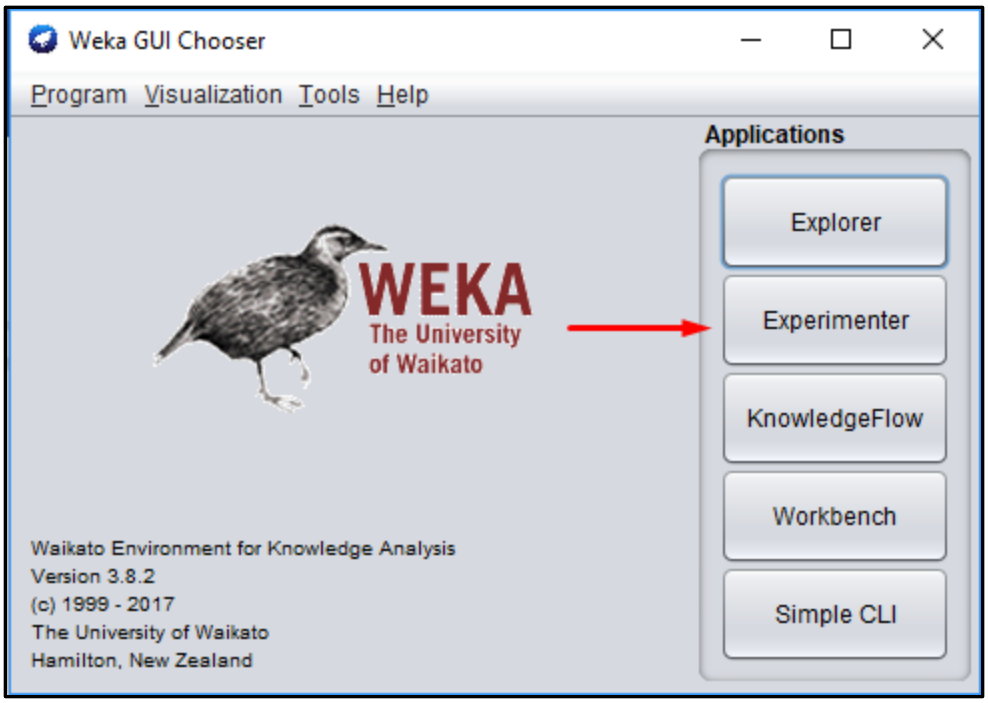


- **Area under ROC (AUC) = 0.9905**

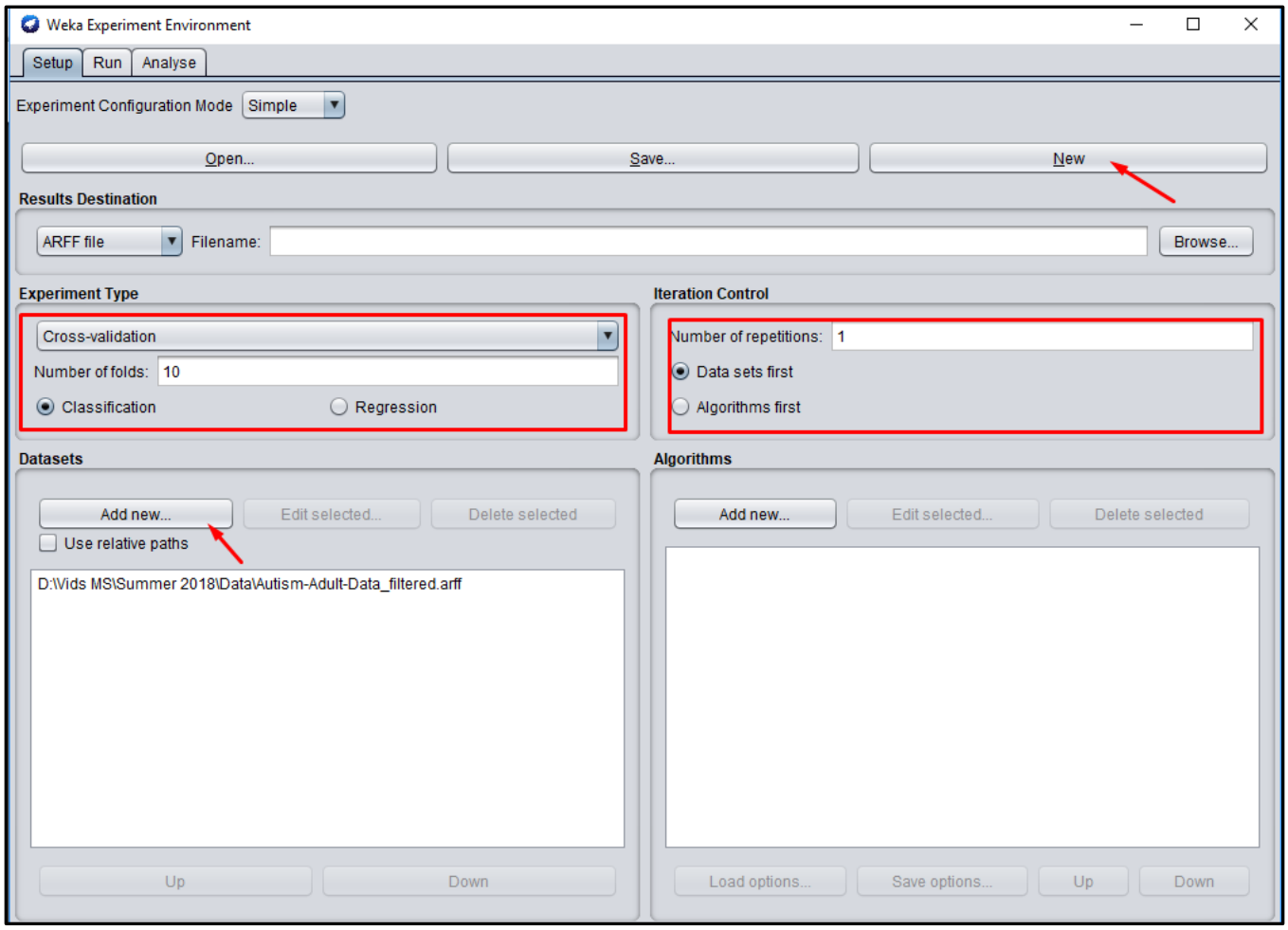
8. Comparison of Classification Algorithm Accuracies

The accuracies of the classification accuracies can be compared using WEKA, to identify which classification algorithm gives the best results for a given dataset. To compare and evaluate the classification accuracies using WEKA

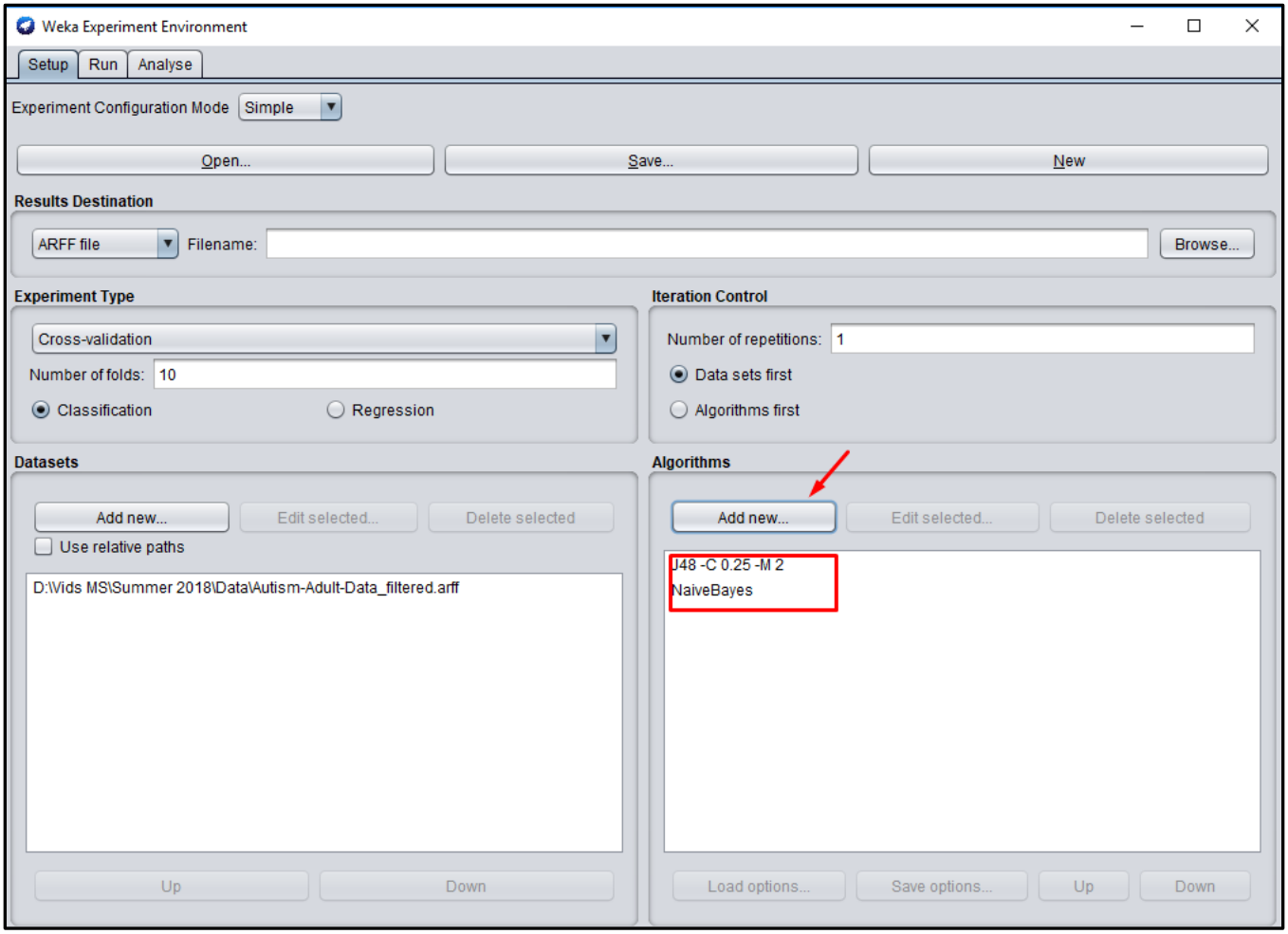
- Click on “Experimenter” tab under Weka GUI Chooser as given below.



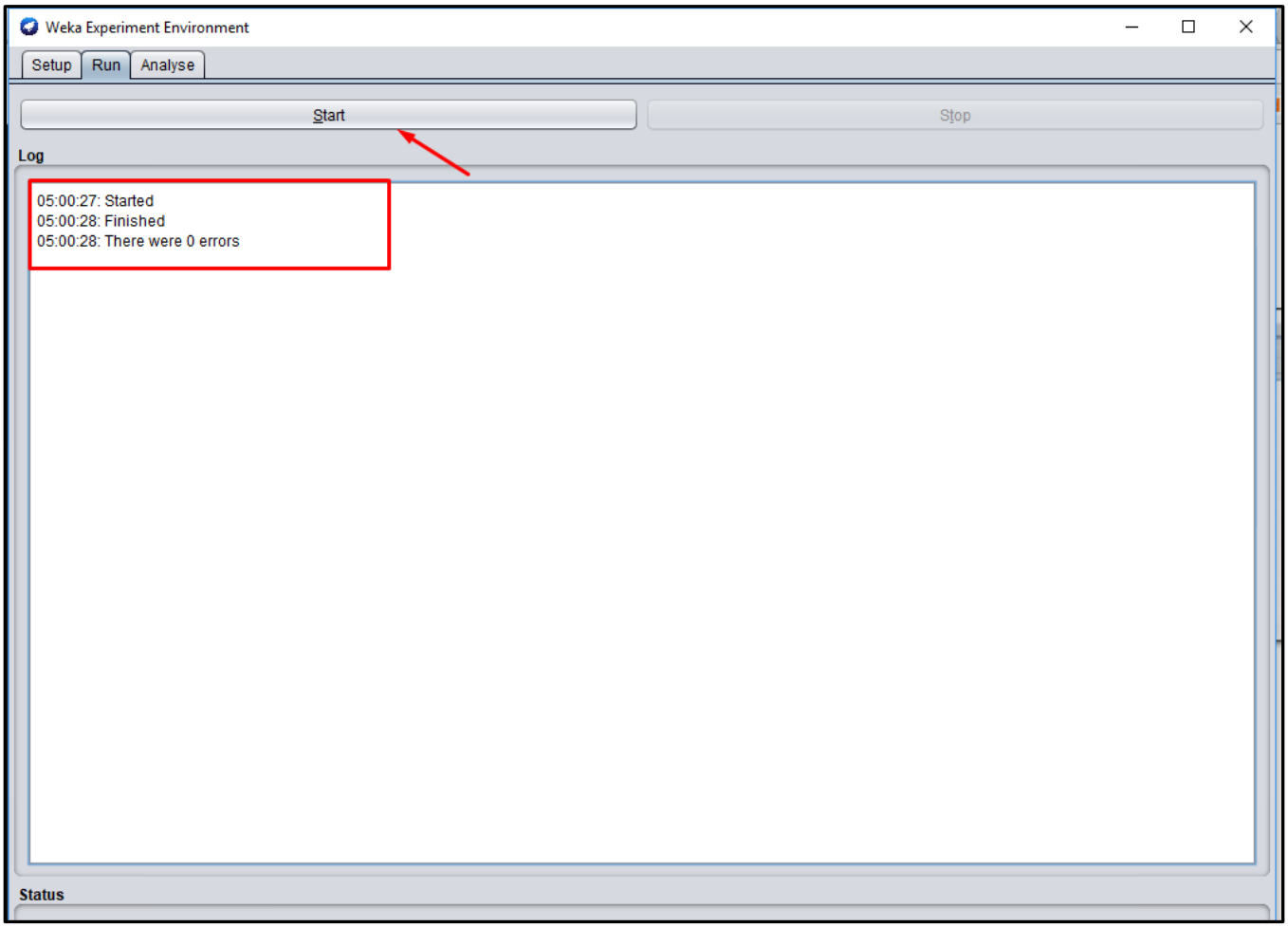
- Click on "**New**" and set the following parameters, as given in the screenshot
  - Select "**Cross-validation**" under Experiment Type. Make sure that the Classification option is selected
  - Enter "**10**" as the value for Number of folds under Experiment Type.
  - Enter the number of iterations as"1" and make sure "Data Sets first" is selected
  - Click on "**Add New**" under Datasets and select the training data set (In this case: Autism dataset)


Weka Experiment Environment
Setup
Run
Analyse
Experiment Configuration Mode
Simple
Open...
Save...
New
Results Destination
ARFF file
Filename:
Browse...
Experiment Type
Cross-validation
Number of folds:
10
Classification
Regression
Iteration Control
Number of repetitions:
1
Data sets first
Algorithms first
Datasets
Add new...
Edit selected...
Delete selected
Use relative paths
D:\Vids MS\Summer 2018\Data\Autism-Adult-Data_filtered.arff
Up
Down
Algorithms
Add new...
Edit selected...
Delete selected
Load options...
Save options...
Up
Down

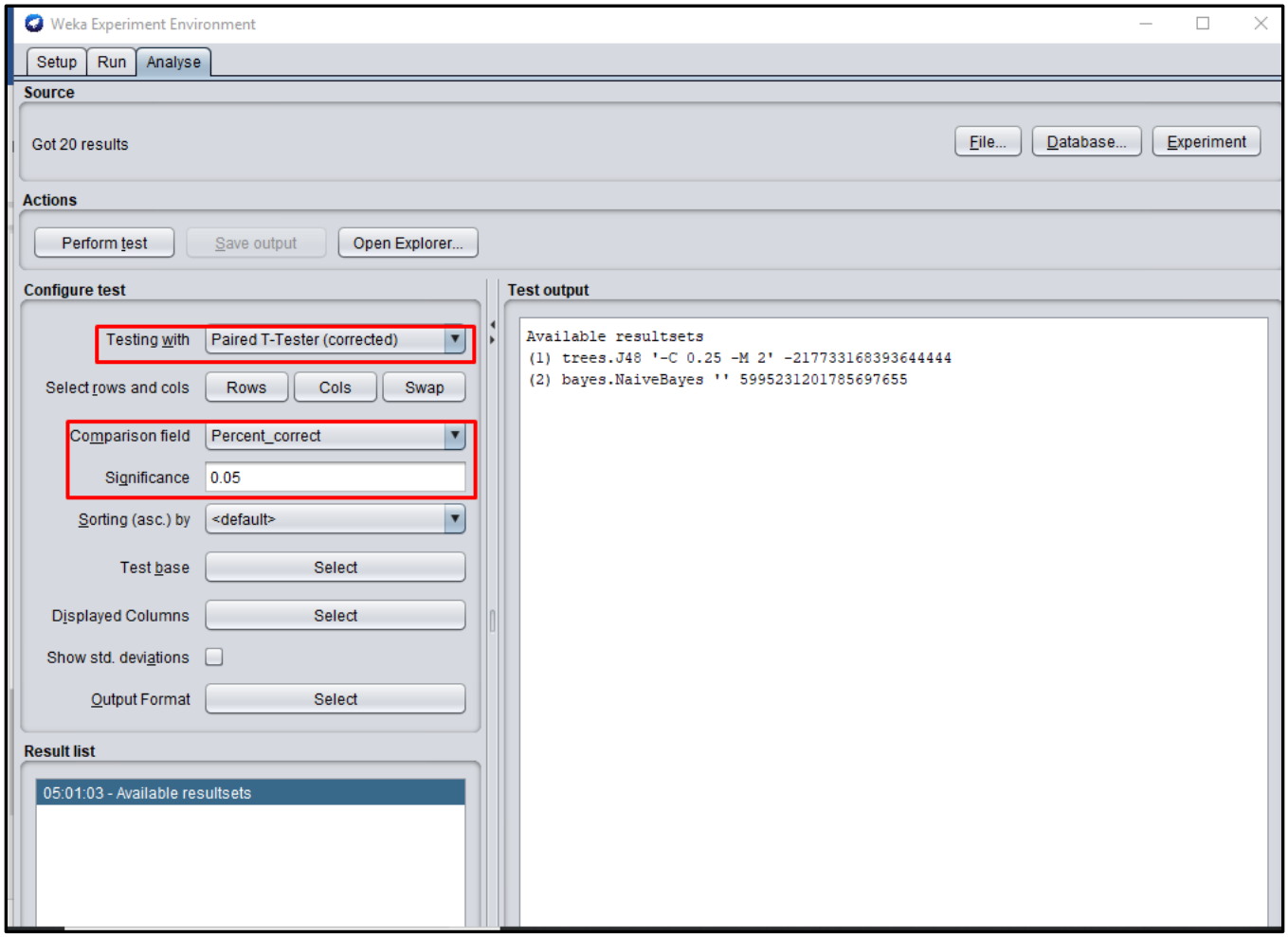
- In the same window, click on "**Add New**" under **Algorithms** to select the classifier algorithm. Select the **J48** algorithm from the trees folder. Repeat the same to add **Naive Bayes** Algorithm



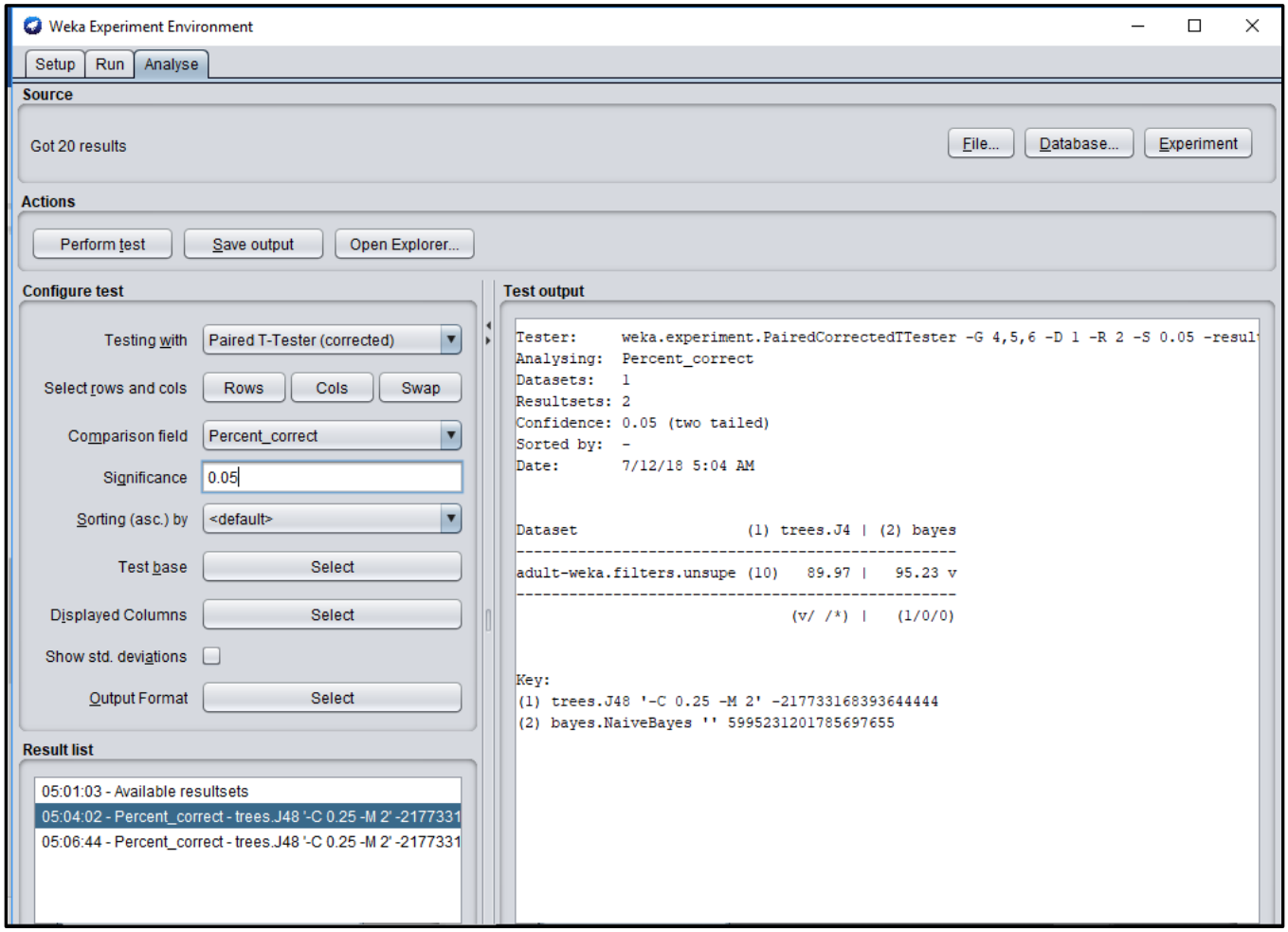
- Once done with the above settings, go to the second tab “**Run**”, and click on “**Start**”.



- Now click on the third tab “Analyse” to analyse the comparisons between the two selected algorithms – J48 and Naive Bayes. On clicking Analyse tab, the following window appears. Click on “**Experiment”** button. Under Configure test, Set the fields
  - “Testing with” to “**Paired T-Tester”** (corrected)
  - Comparison filed to “**Percent_correct**”
  - Significance to “**0.05**”
- Now click on Perform test to compare the accuracies of the two classifiers



- The below screenshot displays the output of the comparison of classifier algorithms – J48 & Naive Bayes.



8.1. Experimental Results:

```
Tester:     weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -result-matrix "weka.experiment.P
Analysing:  Percent_correct
Datasets:   1
Resultsets: 2
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       7/12/18 5:04 AM


Dataset                   (1) trees.J4 | (2) bayes
----------------------------------------------------
adult-weka.filters.unsupe (10)   89.97 |   95.23 v
----------------------------------------------------
                               (v/ /*) |   (1/0/0)


Key:
(1) trees.J48 '-C 0.25 -M 2' -217733168393644444
(2) bayes.NaiveBayes '' 5995231201785697655
```

From the Experimenter results we find that

- **The classifier accuracy of J48 algorithm on Autism dataset is: 89.97%**
- **The classifier accuracy of Naive Bayes Algorithm Autism dataset is: 95.23%**

Hence, the comparison between 10 -fold cross fold validation accuracies of J48 and Naive Bayes on Autism Dataset indicates that Naive Bayes performs better and it is the best suited algorithm to perform classification on the given dataset.

9. Source Code

9.1. Decision Tree (J48)

The source code for WEKA Decision tree algorithm in JAVA can be obtained from the below link.

https://svn.cms.waikato.ac.nz/svn/weka/branches/book2ndEd-branch/weka/src/main/java/weka/classifiers/trees/J48.java

```
/*
 *    This program is free software; you can redistribute it and/or modify
 *    it under the terms of the GNU General Public License as published by
 *    the Free Software Foundation; either version 2 of the License, or
 *    (at your option) any later version.
 *
 *    This program is distributed in the hope that it will be useful,
 *    but WITHOUT ANY WARRANTY; without even the implied warranty of
 *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *    GNU General Public License for more details.
 *
 *    You should have received a copy of the GNU General Public License
 *    along with this program; if not, write to the Free Software
 *    Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

/*
 *    J48.java
 *    Copyright (C) 1999 Eibe Frank
 *
 */

package weka.classifiers.trees;

import weka.classifiers.trees.j48.*;
import java.util.*;
import weka.core.*;
import weka.classifiers.*;

/**
 * Class for generating an unpruned or a pruned C4.5 decision tree.
 * For more information, see<p>
 *
 * Ross Quinlan (1993). <i>C4.5: Programs for Machine Learning</i>,
 * Morgan Kaufmann Publishers, San Mateo, CA. </p>
 *
 * Valid options are: <p>
 *
 * -U <br>
 * Use unpruned tree.<p>
 *
 * -C confidence <br>
 * Set confidence threshold for pruning. (Default: 0.25) <p>
 *
 * -M number <br>
 * Set minimum number of instances per leaf. (Default: 2) <p>
 *
 * -R <br>
```

```
 * Use reduced error pruning. No subtree raising is performed. <p>
 *
 * -N number <br>
 * Set number of folds for reduced error pruning. One fold is
 * used as the pruning set. (Default: 3) <p>
 *
 * -B <br>
 * Use binary splits for nominal attributes. <p>
 *
 * -S <br>
 * Don't perform subtree raising. <p>
 *
 * -L <br>
 * Do not clean up after the tree has been built. <p>
 *
 * -A <br>
 * If set, Laplace smoothing is used for predicted probabilites. <p>
 *
 * -Q <br>
 * The seed for reduced-error pruning. <p>
 *
 * @author Eibe Frank (eibe@cs.waikato.ac.nz)
 * @version $Revision: 1.2 $
 */
public class J48 extends Classifier implements OptionHandler,
  Drawable, Matchable, Sourcable, WeightedInstancesHandler, Summarizable,
  AdditionalMeasureProducer {

  // To maintain the same version number after adding m_ClassAttribute
  static final long serialVersionUID = -217733168393644444L;

  /** The decision tree */
  private ClassifierTree m_root;

  /** Unpruned tree? */
  private boolean m_unpruned = false;

  /** Confidence level */
  private float m_CF = 0.25f;

  /** Minimum number of instances */
  private int m_minNumObj = 2;

  /** Determines whether probabilities are smoothed using
      Laplace correction when predictions are generated */
  private boolean m_useLaplace = false;

  /** Use reduced error pruning? */
  private boolean m_reducedErrorPruning = false;

  /** Number of folds for reduced error pruning. */
  private int m_numFolds = 3;

  /** Binary splits on nominal attributes? */
  private boolean m_binarySplits = false;

  /** Subtree raising to be performed? */
```

```
  private boolean m_subtreeRaising = true;

  /** Cleanup after the tree has been built. */
  private boolean m_noCleanup = false;

  /** Random number seed for reduced-error pruning. */
  private int m_Seed = 1;

  /**
   * Returns a string describing classifier
   * @return a description suitable for
   * displaying in the explorer/experimenter gui
   */
  public String globalInfo() {

    return  "Class for generating a pruned or unpruned C4.5 decision tree. For
more "
      + "information, see\n\n"
      + "Ross Quinlan (1993). \"C4.5: Programs for Machine Learning\", "
      + "Morgan Kaufmann Publishers, San Mateo, CA.\n\n";
  }

  /**
   * Generates the classifier.
   *
   * @exception Exception if classifier can't be built successfully
   */
  public void buildClassifier(Instances instances)
       throws Exception {

    ModelSelection modSelection;

    if (m_binarySplits)
      modSelection = new BinC45ModelSelection(m_minNumObj, instances);
    else
      modSelection = new C45ModelSelection(m_minNumObj, instances);
    if (!m_reducedErrorPruning)
      m_root = new C45PruneableClassifierTree(modSelection, !m_unpruned, m_CF,
                                              m_subtreeRaising, !m_noCleanup);
    else
      m_root = new PruneableClassifierTree(modSelection, !m_unpruned, m_numFolds,
                                           !m_noCleanup, m_Seed);
    m_root.buildClassifier(instances);
    if (m_binarySplits) {
      ((BinC45ModelSelection)modSelection).cleanup();
    } else {
      ((C45ModelSelection)modSelection).cleanup();
    }
  }

  /**
   * Classifies an instance.
   *
   * @exception Exception if instance can't be classified successfully
   */
  public double classifyInstance(Instance instance) throws Exception {
```

```
  return m_root.classifyInstance(instance);
}

/**
 * Returns class probabilities for an instance.
 *
 * @exception Exception if distribution can't be computed successfully
 */
public final double [] distributionForInstance(Instance instance)
     throws Exception {

  return m_root.distributionForInstance(instance, m_useLaplace);
}

/**
 *  Returns the type of graph this classifier
 *  represents.
 *  @return Drawable.TREE
 */
public int graphType() {
    return Drawable.TREE;
}

/**
 * Returns graph describing the tree.
 *
 * @exception Exception if graph can't be computed
 */
public String graph() throws Exception {

  return m_root.graph();
}

/**
 * Returns tree in prefix order.
 *
 * @exception Exception if something goes wrong
 */
public String prefix() throws Exception {

  return m_root.prefix();
}


/**
 * Returns tree as an if-then statement.
 *
 * @return the tree as a Java if-then type statement
 * @exception Exception if something goes wrong
 */
public String toSource(String className) throws Exception {

  StringBuffer [] source = m_root.toSource(className);
  return
  "class " + className + " {\n\n"
  +"  public static double classify(Object [] i)\n"
  +"    throws Exception {\n\n"
```

```
  +"    double p = Double.NaN;\n"
  + source[0]  // Assignment code
  +"    return p;\n"
  +"  }\n"
  + source[1]  // Support code
  +"}\n";
}

/**
 * Returns an enumeration describing the available options.
 *
 * Valid options are: <p>
 *
 * -U <br>
 * Use unpruned tree.<p>
 *
 * -C confidence <br>
 * Set confidence threshold for pruning. (Default: 0.25) <p>
 *
 * -M number <br>
 * Set minimum number of instances per leaf. (Default: 2) <p>
 *
 * -R <br>
 * Use reduced error pruning. No subtree raising is performed. <p>
 *
 * -N number <br>
 * Set number of folds for reduced error pruning. One fold is
 * used as the pruning set. (Default: 3) <p>
 *
 * -B <br>
 * Use binary splits for nominal attributes. <p>
 *
 * -S <br>
 * Don't perform subtree raising. <p>
 *
 * -L <br>
 * Do not clean up after the tree has been built.
 *
 * -A <br>
 * If set, Laplace smoothing is used for predicted probabilites. <p>
 *
 * -Q <br>
 * The seed for reduced-error pruning. <p>
 *
 * @return an enumeration of all the available options.
 */
public Enumeration listOptions() {

  Vector newVector = new Vector(9);

  newVector.
      addElement(new Option("\tUse unpruned tree.",
                            "U", 0, "-U"));
  newVector.
      addElement(new Option("\tSet confidence threshold for pruning.\n" +
                            "\t(default 0.25)",
                            "C", 1, "-C <pruning confidence>"));
```

```
    newVector.
        addElement(new Option("\tSet minimum number of instances per leaf.\n" +
                              "\t(default 2)",
                              "M", 1, "-M <minimum number of instances>"));
    newVector.
        addElement(new Option("\tUse reduced error pruning.",
                              "R", 0, "-R"));
    newVector.
        addElement(new Option("\tSet number of folds for reduced error\n" +
                              "\tpruning. One fold is used as pruning set.\n" +
                              "\t(default 3)",
                              "N", 1, "-N <number of folds>"));
    newVector.
        addElement(new Option("\tUse binary splits only.",
                              "B", 0, "-B"));
    newVector.
         addElement(new Option("\tDon't perform subtree raising.",
                              "S", 0, "-S"));
    newVector.
         addElement(new Option("\tDo not clean up after the tree has been built.",
                              "L", 0, "-L"));
   newVector.
         addElement(new Option("\tLaplace smoothing for predicted probabilities.",
                              "A", 0, "-A"));
    newVector.
      addElement(new Option("\tSeed for random data shuffling (default 1).",
                            "Q", 1, "-Q <seed>"));

    return newVector.elements();
  }

  /**
   * Parses a given list of options.
   *
   * @param options the list of options as an array of strings
   * @exception Exception if an option is not supported
   */
  public void setOptions(String[] options) throws Exception {

    // Other options
    String minNumString = Utils.getOption('M', options);
    if (minNumString.length() != 0) {
      m_minNumObj = Integer.parseInt(minNumString);
    } else {
      m_minNumObj = 2;
    }
    m_binarySplits = Utils.getFlag('B', options);
    m_useLaplace = Utils.getFlag('A', options);

    // Pruning options
    m_unpruned = Utils.getFlag('U', options);
    m_subtreeRaising = !Utils.getFlag('S', options);
    m_noCleanup = Utils.getFlag('L', options);
    if ((m_unpruned) && (!m_subtreeRaising)) {
      throw new Exception("Subtree raising doesn't need to be unset for unpruned
tree!");
    }
```

```
    m_reducedErrorPruning = Utils.getFlag('R', options);
    if ((m_unpruned) && (m_reducedErrorPruning)) {
      throw new Exception("Unpruned tree and reduced error pruning can't be
selected " +
                          "simultaneously!");
    }
    String confidenceString = Utils.getOption('C', options);
    if (confidenceString.length() != 0) {
      if (m_reducedErrorPruning) {
        throw new Exception("Setting the confidence doesn't make sense " +
                            "for reduced error pruning.");
      } else if (m_unpruned) {
        throw new Exception("Doesn't make sense to change confidence for unpruned
"
                            +"tree!");
      } else {
        m_CF = (new Float(confidenceString)).floatValue();
        if ((m_CF <= 0) || (m_CF >= 1)) {
          throw new Exception("Confidence has to be greater than zero and smaller
" +
                              "than one!");
        }
      }
    } else {
      m_CF = 0.25f;
    }
    String numFoldsString = Utils.getOption('N', options);
    if (numFoldsString.length() != 0) {
      if (!m_reducedErrorPruning) {
        throw new Exception("Setting the number of folds" +
                            " doesn't make sense if" +
                            " reduced error pruning is not selected.");
      } else {
        m_numFolds = Integer.parseInt(numFoldsString);
      }
    } else {
      m_numFolds = 3;
    }
    String seedString = Utils.getOption('Q', options);
    if (seedString.length() != 0) {
      m_Seed = Integer.parseInt(seedString);
    } else {
      m_Seed = 1;
    }
  }

  /**
   * Gets the current settings of the Classifier.
   *
   * @return an array of strings suitable for passing to setOptions
   */
  public String [] getOptions() {

    String [] options = new String [14];
    int current = 0;

    if (m_noCleanup) {
```

```
    options[current++] = "-L";
  }
  if (m_unpruned) {
    options[current++] = "-U";
  } else {
    if (!m_subtreeRaising) {
      options[current++] = "-S";
    }
    if (m_reducedErrorPruning) {
      options[current++] = "-R";
      options[current++] = "-N"; options[current++] = "" + m_numFolds;
      options[current++] = "-Q"; options[current++] = "" + m_Seed;
    } else {
      options[current++] = "-C"; options[current++] = "" + m_CF;
    }
  }
  if (m_binarySplits) {
    options[current++] = "-B";
  }
  options[current++] = "-M"; options[current++] = "" + m_minNumObj;
  if (m_useLaplace) {
    options[current++] = "-A";
  }

  while (current < options.length) {
    options[current++] = "";
  }
  return options;
}

/**
 * Returns the tip text for this property
 * @return tip text for this property suitable for
 * displaying in the explorer/experimenter gui
 */
public String seedTipText() {
  return "The seed used for randomizing the data " +
    "when reduced-error pruning is used.";
}

/**
 * Get the value of Seed.
 *
 * @return Value of Seed.
 */
public int getSeed() {

  return m_Seed;
}

/**
 * Set the value of Seed.
 *
 * @param newSeed Value to assign to Seed.
 */
public void setSeed(int newSeed) {
```

```
    m_Seed = newSeed;
  }

  /**
   * Returns the tip text for this property
   * @return tip text for this property suitable for
   * displaying in the explorer/experimenter gui
   */
  public String useLaplaceTipText() {
    return "Whether counts at leaves are smoothed based on Laplace.";
  }

  /**
   * Get the value of useLaplace.
   *
   * @return Value of useLaplace.
   */
  public boolean getUseLaplace() {

    return m_useLaplace;
  }

  /**
   * Set the value of useLaplace.
   *
   * @param newuseLaplace Value to assign to useLaplace.
   */
  public void setUseLaplace(boolean newuseLaplace) {

    m_useLaplace = newuseLaplace;
  }

  /**
   * Returns a description of the classifier.
   */
  public String toString() {

    if (m_root == null) {
      return "No classifier built";
    }
    if (m_unpruned)
      return "J48 unpruned tree\n------------------\n" + m_root.toString();
    else
      return "J48 pruned tree\n------------------\n" + m_root.toString();
  }

  /**
   * Returns a superconcise version of the model
   */
  public String toSummaryString() {

    return "Number of leaves: " + m_root.numLeaves() + "\n"
         + "Size of the tree: " + m_root.numNodes() + "\n";
  }

  /**
   * Returns the size of the tree
```

```
   * @return the size of the tree
   */
  public double measureTreeSize() {
    return m_root.numNodes();
  }

  /**
   * Returns the number of leaves
   * @return the number of leaves
   */
  public double measureNumLeaves() {
    return m_root.numLeaves();
  }

  /**
   * Returns the number of rules (same as number of leaves)
   * @return the number of rules
   */
  public double measureNumRules() {
    return m_root.numLeaves();
  }

  /**
   * Returns an enumeration of the additional measure names
   * @return an enumeration of the measure names
   */
  public Enumeration enumerateMeasures() {
    Vector newVector = new Vector(3);
    newVector.addElement("measureTreeSize");
    newVector.addElement("measureNumLeaves");
    newVector.addElement("measureNumRules");
    return newVector.elements();
  }

  /**
   * Returns the value of the named measure
   * @param measureName the name of the measure to query for its value
   * @return the value of the named measure
   * @exception IllegalArgumentException if the named measure is not supported
   */
  public double getMeasure(String additionalMeasureName) {
    if (additionalMeasureName.compareToIgnoreCase("measureNumRules") == 0) {
      return measureNumRules();
    } else if (additionalMeasureName.compareToIgnoreCase("measureTreeSize") == 0)
{
      return measureTreeSize();
    } else if (additionalMeasureName.compareToIgnoreCase("measureNumLeaves") == 0)
{
      return measureNumLeaves();
    } else {
      throw new IllegalArgumentException(additionalMeasureName
                        + " not supported (j48)");
    }
  }

  /**
   * Returns the tip text for this property
```

```
   * @return tip text for this property suitable for
   * displaying in the explorer/experimenter gui
   */
  public String unprunedTipText() {
    return "Whether pruning is performed.";
  }

  /**
   * Get the value of unpruned.
   *
   * @return Value of unpruned.
   */
  public boolean getUnpruned() {

    return m_unpruned;
  }

  /**
   * Set the value of unpruned. Turns reduced-error pruning
   * off if set.
   * @param v  Value to assign to unpruned.
   */
  public void setUnpruned(boolean v) {

    if (v) {
      m_reducedErrorPruning = false;
    }
    m_unpruned = v;
  }

  /**
   * Returns the tip text for this property
   * @return tip text for this property suitable for
   * displaying in the explorer/experimenter gui
   */
  public String confidenceFactorTipText() {
    return "The confidence factor used for pruning (smaller values incur "
      + "more pruning).";
  }

  /**
   * Get the value of CF.
   *
   * @return Value of CF.
   */
  public float getConfidenceFactor() {

    return m_CF;
  }

  /**
   * Set the value of CF.
   *
   * @param v  Value to assign to CF.
   */
  public void setConfidenceFactor(float v) {
```

```
    m_CF = v;
  }

  /**
   * Returns the tip text for this property
   * @return tip text for this property suitable for
   * displaying in the explorer/experimenter gui
   */
  public String minNumObjTipText() {
    return "The minimum number of instances per leaf.";
  }

  /**
   * Get the value of minNumObj.
   *
   * @return Value of minNumObj.
   */
  public int getMinNumObj() {

    return m_minNumObj;
  }

  /**
   * Set the value of minNumObj.
   *
   * @param v  Value to assign to minNumObj.
   */
  public void setMinNumObj(int v) {

    m_minNumObj = v;
  }

  /**
   * Returns the tip text for this property
   * @return tip text for this property suitable for
   * displaying in the explorer/experimenter gui
   */
  public String reducedErrorPruningTipText() {
    return "Whether reduced-error pruning is used instead of C.4.5 pruning.";
  }

  /**
   * Get the value of reducedErrorPruning.
   *
   * @return Value of reducedErrorPruning.
   */
  public boolean getReducedErrorPruning() {

    return m_reducedErrorPruning;
  }

  /**
   * Set the value of reducedErrorPruning. Turns
   * unpruned trees off if set.
   *
   * @param v  Value to assign to reducedErrorPruning.
   */
```

```
public void setReducedErrorPruning(boolean v) {

  if (v) {
    m_unpruned = false;
  }
  m_reducedErrorPruning = v;
}

/**
 * Returns the tip text for this property
 * @return tip text for this property suitable for
 * displaying in the explorer/experimenter gui
 */
public String numFoldsTipText() {
  return "Determines the amount of data used for reduced-error pruning. "
    + " One fold is used for pruning, the rest for growing the tree.";
}

/**
 * Get the value of numFolds.
 *
 * @return Value of numFolds.
 */
public int getNumFolds() {

  return m_numFolds;
}

/**
 * Set the value of numFolds.
 *
 * @param v  Value to assign to numFolds.
 */
public void setNumFolds(int v) {

  m_numFolds = v;
}

/**
 * Returns the tip text for this property
 * @return tip text for this property suitable for
 * displaying in the explorer/experimenter gui
 */
public String binarySplitsTipText() {
  return "Whether to use binary splits on nominal attributes when "
    + "building the trees.";
}

/**
 * Get the value of binarySplits.
 *
 * @return Value of binarySplits.
 */
public boolean getBinarySplits() {

  return m_binarySplits;
}
```

```
/**
 * Set the value of binarySplits.
 *
 * @param v  Value to assign to binarySplits.
 */
public void setBinarySplits(boolean v) {

  m_binarySplits = v;
}

/**
 * Returns the tip text for this property
 * @return tip text for this property suitable for
 * displaying in the explorer/experimenter gui
 */
public String subtreeRaisingTipText() {
  return "Whether to consider the subtree raising operation when pruning.";
}

/**
 * Get the value of subtreeRaising.
 *
 * @return Value of subtreeRaising.
 */
public boolean getSubtreeRaising() {

  return m_subtreeRaising;
}

/**
 * Set the value of subtreeRaising.
 *
 * @param v  Value to assign to subtreeRaising.
 */
public void setSubtreeRaising(boolean v) {

  m_subtreeRaising = v;
}

/**
 * Returns the tip text for this property
 * @return tip text for this property suitable for
 * displaying in the explorer/experimenter gui
 */
public String saveInstanceDataTipText() {
  return "Whether to save the training data for visualization.";
}

/**
 * Check whether instance data is to be saved.
 *
 * @return true if instance data is saved
 */
public boolean getSaveInstanceData() {

  return m_noCleanup;
```

```
  }

  /**
   * Set whether instance data is to be saved.
   * @param v true if instance data is to be saved
   */
  public void setSaveInstanceData(boolean v) {

    m_noCleanup = v;
  }

  /**
   * Main method for testing this class
   *
   * @param String options
   */
  public static void main(String [] argv){

    try {
      System.out.println(Evaluation.evaluateModel(new J48(), argv));
    } catch (Exception e) {
      System.err.println(e.getMessage());
    }
  }
}
```

9.2. Naive Bayes

The source code for WEKA Naive Bayes algorithm in JAVA can be obtained from the below link.

https://svn.cms.waikato.ac.nz/svn/weka/branches/book2ndEd-branch/weka/src/main/java/weka/classifiers/bayes/NaiveBayes.java

```
/*
 *    This program is free software; you can redistribute it and/or modify
 *    it under the terms of the GNU General Public License as published by
 *    the Free Software Foundation; either version 2 of the License, or
 *    (at your option) any later version.
 *
 *    This program is distributed in the hope that it will be useful,
 *    but WITHOUT ANY WARRANTY; without even the implied warranty of
 *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *    GNU General Public License for more details.
 *
 *    You should have received a copy of the GNU General Public License
 *    along with this program; if not, write to the Free Software
 *    Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

/*
 *    NaiveBayes.java
 *    Copyright (C) 1999 Eibe Frank,Len Trigg
```

```
 *
 */

package weka.classifiers.bayes;


import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.UpdateableClassifier;
import java.io.*;
import java.util.*;
import weka.core.*;
import weka.estimators.*;

/**
 * Class for a Naive Bayes classifier using estimator classes. Numeric
 * estimator precision values are chosen based on analysis of the
 * training data. For this reason, the classifier is not an
 * UpdateableClassifier (which in typical usage are initialized with zero
 * training instances) -- if you need the UpdateableClassifier functionality,
 * use the NaiveBayesUpdateable classifier. The NaiveBayesUpdateable
 * classifier will  use a default precision of 0.1 for numeric attributes
 * when buildClassifier is called with zero training instances.
 * <p>
 * For more information on Naive Bayes classifiers, see<p>
 *
 * George H. John and Pat Langley (1995). <i>Estimating
 * Continuous Distributions in Bayesian Classifiers</i>. Proceedings
 * of the Eleventh Conference on Uncertainty in Artificial
 * Intelligence. pp. 338-345. Morgan Kaufmann, San Mateo.<p>
 *
 * Valid options are:<p>
 *
 * -K <br>
 * Use kernel estimation for modelling numeric attributes rather than
 * a single normal distribution.<p>
 *
 * -D <br>
 * Use supervised discretization to process numeric attributes.<p>
 *
 * @author Len Trigg (trigg@cs.waikato.ac.nz)
 * @author Eibe Frank (eibe@cs.waikato.ac.nz)
 * @version $Revision: 1.16 $
 */
public class NaiveBayes extends Classifier
  implements OptionHandler, WeightedInstancesHandler {

  /** The attribute estimators. */
  protected Estimator [][] m_Distributions;

  /** The class estimator. */
  protected Estimator m_ClassDistribution;

  /**
   * Whether to use kernel density estimator rather than normal distribution
   * for numeric attributes
   */
```

```
  protected boolean m_UseKernelEstimator = false;

  /**
   * Whether to use discretization than normal distribution
   * for numeric attributes
   */
  protected boolean m_UseDiscretization = false;

  /** The number of classes (or 1 for numeric class) */
  protected int m_NumClasses;

  /**
   * The dataset header for the purposes of printing out a semi-intelligible
   * model
   */
  protected Instances m_Instances;

  /*** The precision parameter used for numeric attributes */
  protected static final double DEFAULT_NUM_PRECISION = 0.01;

  /**
   * The discretization filter.
   */
  protected weka.filters.supervised.attribute.Discretize m_Disc = null;

  /**
   * Returns a string describing this classifier
   * @return a description of the classifier suitable for
   * displaying in the explorer/experimenter gui
   */
  public String globalInfo() {
    return "Class for a Naive Bayes classifier using estimator classes. Numeric"
      +" estimator precision values are chosen based on analysis of the "
      +" training data. For this reason, the classifier is not an"
      +" UpdateableClassifier (which in typical usage are initialized with zero"
      +" training instances) -- if you need the UpdateableClassifier
functionality,"
      +" use the NaiveBayesUpdateable classifier. The NaiveBayesUpdateable"
      +" classifier will  use a default precision of 0.1 for numeric attributes"
      +" when buildClassifier is called with zero training instances.\n\n"
      +"For more information on Naive Bayes classifiers, see\n\n"
      +"George H. John and Pat Langley (1995). Estimating"
      + " Continuous Distributions in Bayesian Classifiers. Proceedings"
      +" of the Eleventh Conference on Uncertainty in Artificial"
      +" Intelligence. pp. 338-345. Morgan Kaufmann, San Mateo.";
  }

  /**
   * Generates the classifier.
   *
   * @param instances set of instances serving as training data
   * @exception Exception if the classifier has not been generated
   * successfully
   */
  public void buildClassifier(Instances instances) throws Exception {

    if (instances.checkForStringAttributes()) {
```

```
      throw new UnsupportedAttributeTypeException("Cannot handle string
attributes!");
    }
    if (instances.classAttribute().isNumeric()) {
      throw new UnsupportedClassTypeException("Naive Bayes: Class is numeric!");
    }
    m_NumClasses = instances.numClasses();
    if (m_NumClasses < 0) {
      throw new Exception ("Dataset has no class attribute");
    }

    // Copy the instances
    m_Instances = new Instances(instances);

    // Discretize instances if required
    if (m_UseDiscretization) {
      m_Disc = new weka.filters.supervised.attribute.Discretize();
      m_Disc.setInputFormat(m_Instances);
      m_Instances = weka.filters.Filter.useFilter(m_Instances, m_Disc);
    } else {
      m_Disc = null;
    }

    // Reserve space for the distributions
    m_Distributions = new Estimator[m_Instances.numAttributes() - 1]
    [m_Instances.numClasses()];
    m_ClassDistribution = new DiscreteEstimator(m_Instances.numClasses(),
                                                true);
    int attIndex = 0;
    Enumeration enu = m_Instances.enumerateAttributes();
    while (enu.hasMoreElements()) {
      Attribute attribute = (Attribute) enu.nextElement();

      // If the attribute is numeric, determine the estimator
      // numeric precision from differences between adjacent values
      double numPrecision = DEFAULT_NUM_PRECISION;
      if (attribute.type() == Attribute.NUMERIC) {
        m_Instances.sort(attribute);
        if ((m_Instances.numInstances() > 0)
            && !m_Instances.instance(0).isMissing(attribute)) {
          double lastVal = m_Instances.instance(0).value(attribute);
          double currentVal, deltaSum = 0;
          int distinct = 0;
          for (int i = 1; i < m_Instances.numInstances(); i++) {
            Instance currentInst = m_Instances.instance(i);
            if (currentInst.isMissing(attribute)) {
              break;
            }
            currentVal = currentInst.value(attribute);
            if (currentVal != lastVal) {
              deltaSum += currentVal - lastVal;
              lastVal = currentVal;
              distinct++;
            }
          }
          if (distinct > 0) {
            numPrecision = deltaSum / distinct;
```

```
          }
        }
      }


      for (int j = 0; j < m_Instances.numClasses(); j++) {
        switch (attribute.type()) {
        case Attribute.NUMERIC:
          if (m_UseKernelEstimator) {
            m_Distributions[attIndex][j] =
            new KernelEstimator(numPrecision);
          } else {
            m_Distributions[attIndex][j] =
            new NormalEstimator(numPrecision);
          }
          break;
        case Attribute.NOMINAL:
          m_Distributions[attIndex][j] =
          new DiscreteEstimator(attribute.numValues(), true);
          break;
        default:
          throw new Exception("Attribute type unknown to NaiveBayes");
        }
      }
      attIndex++;
    }

    // Compute counts
    Enumeration enumInsts = m_Instances.enumerateInstances();
    while (enumInsts.hasMoreElements()) {
      Instance instance =
        (Instance) enumInsts.nextElement();
      updateClassifier(instance);
    }

    // Save space
    m_Instances = new Instances(m_Instances, 0);
  }


  /**
   * Updates the classifier with the given instance.
   *
   * @param instance the new training instance to include in the model
   * @exception Exception if the instance could not be incorporated in
   * the model.
   */
  public void updateClassifier(Instance instance) throws Exception {

    if (!instance.classIsMissing()) {
      Enumeration enumAtts = m_Instances.enumerateAttributes();
      int attIndex = 0;
      while (enumAtts.hasMoreElements()) {
        Attribute attribute = (Attribute) enumAtts.nextElement();
        if (!instance.isMissing(attribute)) {
          m_Distributions[attIndex][(int)instance.classValue()].
            addValue(instance.value(attribute), instance.weight());
```

```
        }
        attIndex++;
      }
      m_ClassDistribution.addValue(instance.classValue(),
                                   instance.weight());
    }
  }


  /**
   * Calculates the class membership probabilities for the given test
   * instance.
   *
   * @param instance the instance to be classified
   * @return predicted class probability distribution
   * @exception Exception if there is a problem generating the prediction
   */
  public double [] distributionForInstance(Instance instance)
  throws Exception {

    if (m_UseDiscretization) {
      m_Disc.input(instance);
      instance = m_Disc.output();
    }
    double [] probs = new double[m_NumClasses];
    for (int j = 0; j < m_NumClasses; j++) {
      probs[j] = m_ClassDistribution.getProbability(j);
    }
    Enumeration enumAtts = instance.enumerateAttributes();
    int attIndex = 0;
    while (enumAtts.hasMoreElements()) {
      Attribute attribute = (Attribute) enumAtts.nextElement();
      if (!instance.isMissing(attribute)) {
        double temp, max = 0;
        for (int j = 0; j < m_NumClasses; j++) {
          temp = Math.max(1e-75, m_Distributions[attIndex][j].
          getProbability(instance.value(attribute)));
          probs[j] *= temp;
          if (probs[j] > max) {
            max = probs[j];
          }
          if (Double.isNaN(probs[j])) {
            throw new Exception("NaN returned from estimator for attribute "
                                + attribute.name() + ":\n"
                                + m_Distributions[attIndex][j].toString());
          }
        }
        if ((max > 0) && (max < 1e-75)) { // Danger of probability underflow
          for (int j = 0; j < m_NumClasses; j++) {
            probs[j] *= 1e75;
          }
        }
      }
      attIndex++;
    }

    // Display probabilities
```

```
    Utils.normalize(probs);
    return probs;
  }

  /**
   * Returns an enumeration describing the available options.
   *
   * @return an enumeration of all the available options.
   */
  public Enumeration listOptions() {

    Vector newVector = new Vector(2);

    newVector.addElement(
    new Option("\tUse kernel density estimator rather than normal\n"
               +"\tdistribution for numeric attributes",
               "K", 0,"-K"));
    newVector.addElement(
    new Option("\tUse supervised discretization to process numeric attributes\n",
               "D", 0,"-D"));
    return newVector.elements();
  }

  /**
   * Parses a given list of options. Valid options are:<p>
   *
   * -K <br>
   * Use kernel estimation for modelling numeric attributes rather than
   * a single normal distribution.<p>
   *
   * -D <br>
   * Use supervised discretization to process numeric attributes.
   *
   * @param options the list of options as an array of strings
   * @exception Exception if an option is not supported
   */
  public void setOptions(String[] options) throws Exception {

    boolean k = Utils.getFlag('K', options);
    boolean d = Utils.getFlag('D', options);
    if (k && d) {
      throw new IllegalArgumentException("Can't use both kernel density " +
                                         "estimation and discretization!");
    }
    setUseSupervisedDiscretization(d);
    setUseKernelEstimator(k);
    Utils.checkForRemainingOptions(options);
  }

  /**
   * Gets the current settings of the classifier.
   *
   * @return an array of strings suitable for passing to setOptions
   */
  public String [] getOptions() {

    String [] options = new String [2];
```

```
    int current = 0;

    if (m_UseKernelEstimator) {
      options[current++] = "-K";
    }

    if (m_UseDiscretization) {
      options[current++] = "-D";
    }

    while (current < options.length) {
      options[current++] = "";
    }
    return options;
  }

  /**
   * Returns a description of the classifier.
   *
   * @return a description of the classifier as a string.
   */
  public String toString() {

    StringBuffer text = new StringBuffer();

    text.append("Naive Bayes Classifier");
    if (m_Instances == null) {
      text.append(": No model built yet.");
    } else {
      try {
        for (int i = 0; i < m_Distributions[0].length; i++) {
          text.append("\n\nClass " + m_Instances.classAttribute().value(i) +
                      ": Prior probability = " + Utils.
                      doubleToString(m_ClassDistribution.getProbability(i),
                                     4, 2) + "\n\n");
          Enumeration enumAtts = m_Instances.enumerateAttributes();
          int attIndex = 0;
          while (enumAtts.hasMoreElements()) {
            Attribute attribute = (Attribute) enumAtts.nextElement();
            text.append(attribute.name() + ":  "
                        + m_Distributions[attIndex][i]);
            attIndex++;
          }
        }
      } catch (Exception ex) {
        text.append(ex.getMessage());
      }
    }

    return text.toString();
  }

  /**
   * Returns the tip text for this property
   * @return tip text for this property suitable for
   * displaying in the explorer/experimenter gui
   */
```

```
  public String useKernelEstimatorTipText() {
    return "Use a kernel estimator for numeric attributes rather than a "
      +"normal distribution.";
  }
  /**
   * Gets if kernel estimator is being used.
   *
   * @return Value of m_UseKernelEstimatory.
   */
  public boolean getUseKernelEstimator() {

    return m_UseKernelEstimator;
  }

  /**
   * Sets if kernel estimator is to be used.
   *
   * @param v  Value to assign to m_UseKernelEstimatory.
   */
  public void setUseKernelEstimator(boolean v) {

    m_UseKernelEstimator = v;
    if (v) {
      setUseSupervisedDiscretization(false);
    }
  }

  /**
   * Returns the tip text for this property
   * @return tip text for this property suitable for
   * displaying in the explorer/experimenter gui
   */
  public String useSupervisedDiscretizationTipText() {
    return "Use supervised discretization to convert numeric attributes to nominal
"
      +"ones.";
  }

  /**
   * Get whether supervised discretization is to be used.
   *
   * @return true if supervised discretization is to be used.
   */
  public boolean getUseSupervisedDiscretization() {

    return m_UseDiscretization;
  }

  /**
   * Set whether supervised discretization is to be used.
   *
   * @param newblah true if supervised discretization is to be used.
   */
  public void setUseSupervisedDiscretization(boolean newblah) {

    m_UseDiscretization = newblah;
    if (newblah) {
```

```
      setUseKernelEstimator(false);
    }
  }

  /**
   * Main method for testing this class.
   *
   * @param argv the options
   */
  public static void main(String [] argv) {

    try {
      System.out.println(Evaluation.evaluateModel(new NaiveBayes(), argv));
    } catch (Exception e) {
      e.printStackTrace();
      System.err.println(e.getMessage());
    }
  }
}
```

10. References

1. http://webdocs.cs.ualberta.ca/~zaiane/courses/cmput690/notes/Chapter1/
2. https://cloudtweaks.com/2014/09/use-supervised-unsupervised-data-mining/
3. https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/
4. https://www.cs.waikato.ac.nz/ml/weka/
5. https://en.wikipedia.org/wiki/Weka_(machine_learning)
6. http://www.saedsayad.com/decision_tree.htm
7. https://en.wikipedia.org/wiki/Decision_tree_learning
8. http://scikit-learn.org/stable/modules/naive_bayes.html
9. https://www.futurelearn.com/courses/data-mining-with-weka/0/steps/25384
10. https://svn.cms.waikato.ac.nz/svn/weka/branches/book2ndEd-branch/weka/src/main/java/weka/classifiers/trees/J48.java
11. https://svn.cms.waikato.ac.nz/svn/weka/branches/book2ndEd-branch/weka/src/main/java/weka/classifiers/bayes/NaiveBayes.java