CS 634 Data Mining
Final Term Project Report

On

Option 1 : Supervised Data Mining (Classification)

Project Report by:
Aarjavi Dharaya
ard22@njit.edu
UCID : ard22
11/21/2020

Table Of Contents

Chapter 1 : Introduction

I have selected **Option 1- Supervised Data Mining (Classification)** for the Final Term Project. This project is focused on analysing the performance of two Machine Learning Algorithms and comparing these two algorithms with various performance parameters like ROC/AUC curve, accuracy, etc. Out of the many ML algorithms listed in Option 1, I have selected **Random Forest - Weka (Category 2)**, and **Naive Bayes Algorithm -Weka (Category 5).** Dataset used, is for the classification of voice samples as male/female and is taken from the website :- Kaggle. The tool which is used to carry out experiments and run analysis is **Weka (Category 9)** - open source tool implemented in Java language. At the end of this project, we are successfully able to run the two algorithms on the dataset and generate necessary performance data and graphs required for the comparison of two algorithms.

Project Details summary:-

Project Option:-	Option 1- Supervised Data Mining (Classification)
Algorithms:-	Random Forest - Weka(Category 2), and Naive Bayes Algorithm-Weka(Category 5).
Tool:-	Weka (Category 9)
DataSet:-	https://www.kaggle.com/primaryobjects/voicegender

Chapter 2 : Project Environment

Operating System	:	Mac OS Catalina x64-bit
RAM	:	8 GB 2133 MHz LPDDR3
Processor	:	2.7 GHz Intel Core i5
Programming Language	:	Java - version 1.8
Testing Tool	:	WEKA 3.8.4 (20 Dec 2019), Copyright (C) 1998-2019 University of Waikato (http://www.cs.waikato.ac.nz/~ml/weka)

Chapter 3 : Software Downloads

You can download Weka for the MaC OS from here:
https://sourceforge.net/projects/weka/files/weka-3-8/3.8.4/weka-3-8-4-azul-zulu-osx.dmg/download

Chapter 4 : Algorithms Tested in this Project

In this Project, I am applying following two algorithms on my dataset :

1. Category 2 :- Random Forests (Weka)
    - http://weka.sourceforge.net/doc/weka/classifiers/trees/RandomForest.html
    - A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

2. Category 5 :- Naive Bayes (Weka)
    - Naive Bayes is a probabilistic machine learning algorithm based on the Bayes Theorem, used in a wide variety of classification tasks.
    - The fundamental Naive Bayes assumption is that each feature makes an: independent and equal contribution to the outcome.

Chapter 5 : DataSet Information

DataSet Description (Reference from - https://www.kaggle.com/primaryobjects/voicegender):

Voice Gender

Gender Recognition by Voice and Speech Analysis

This database was created to identify a voice as male or female, based upon acoustic properties of the voice and speech. The dataset consists of 3,168 recorded voice samples, collected from male and female speakers. The voice samples are pre-processed by acoustic analysis in R using the seewave and tuneR packages, with an analyzed frequency range of 0hz-280hz (human vocal range).

The Dataset

The following acoustic properties of each voice are measured and included within the CSV:

- meanfreq: mean frequency (in kHz)
- sd: standard deviation of frequency
- median: median frequency (in kHz)
- Q25: first quantile (in kHz)
- Q75: third quantile (in kHz)
- IQR: interquantile range (in kHz)
- skew: skewness (see note in specprop description)
- kurt: kurtosis (see note in specprop description)
- sp.ent: spectral entropy
- sfm: spectral flatness
- mode: mode frequency
- centroid: frequency centroid (see specprop)
- peakf: peak frequency (frequency with highest energy)
- meanfun: average of fundamental frequency measured across acoustic signal
- minfun: minimum fundamental frequency measured across acoustic signal
- maxfun: maximum fundamental frequency measured across acoustic signal
- meandom: average of dominant frequency measured across acoustic signal
- mindom: minimum of dominant frequency measured across acoustic signal
- maxdom: maximum of dominant frequency measured across acoustic signal
- dfrange: range of dominant frequency measured across acoustic signal

- modindx: modulation index. Calculated as the accumulated absolute difference between adjacent measurements of fundamental frequencies divided by the frequency range
- label: male or female

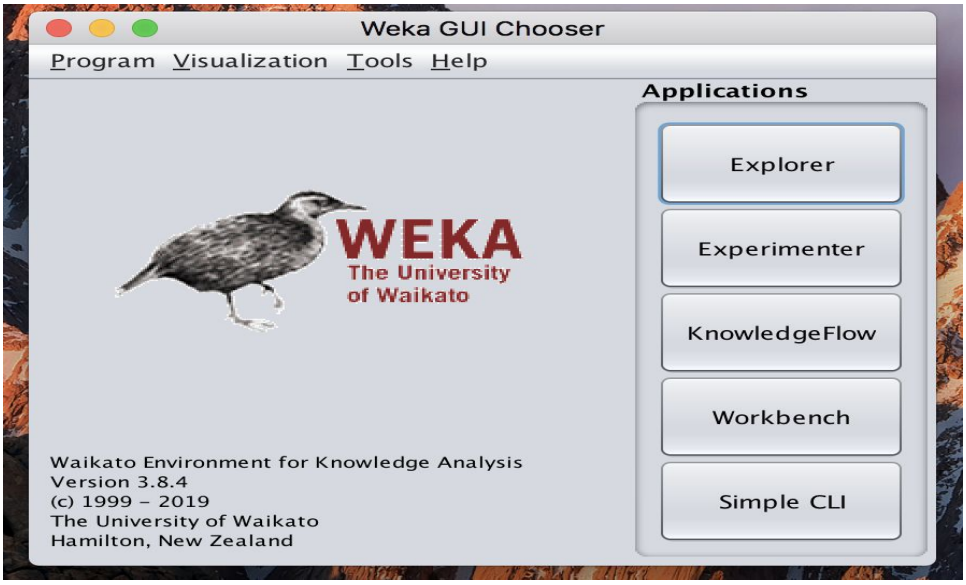
DataSet Summary:

Source : https://www.kaggle.com/primaryobjects/voicegender
File Format : .csv
Size: 1.02 MB
Instances: 3168
Attributes: 21 (Decimal - 20 , String -1)

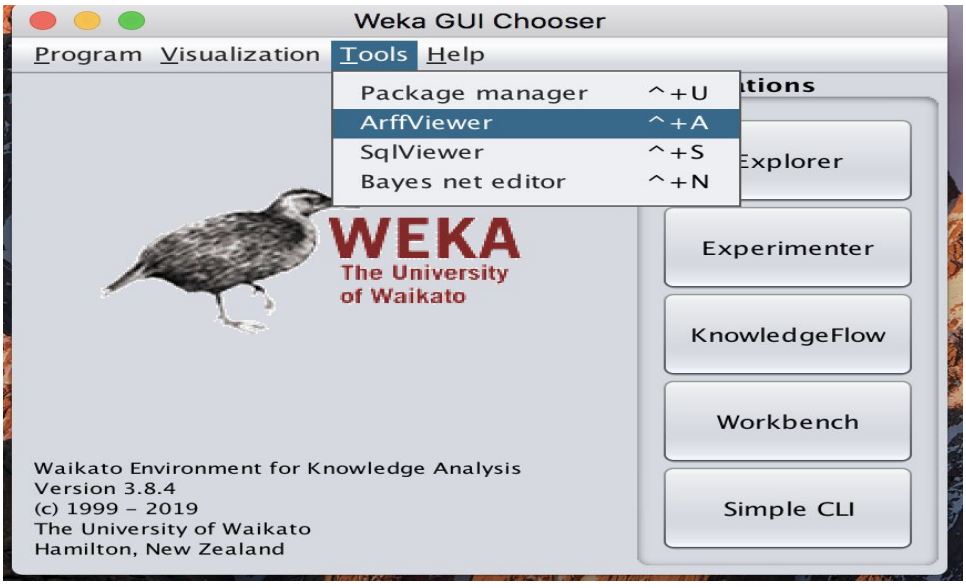
Chapter 6 : Preparing the Data for Weka

The original Dataset is in .csv format. To run algorithms in Weka on the dataset, the file format that is supported is only .arff. Hence we need to convert .csv data to .arff data by following method:

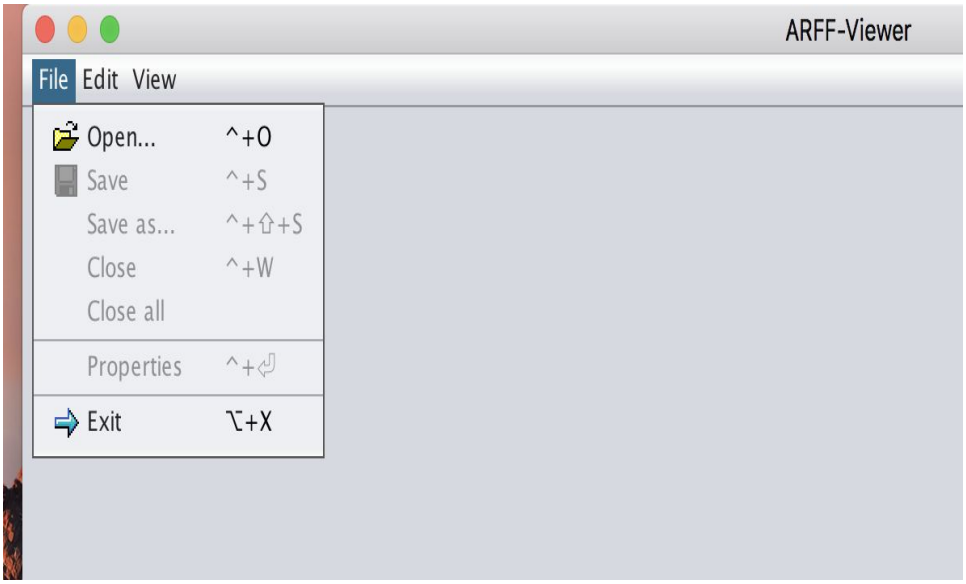
1. Open Weka



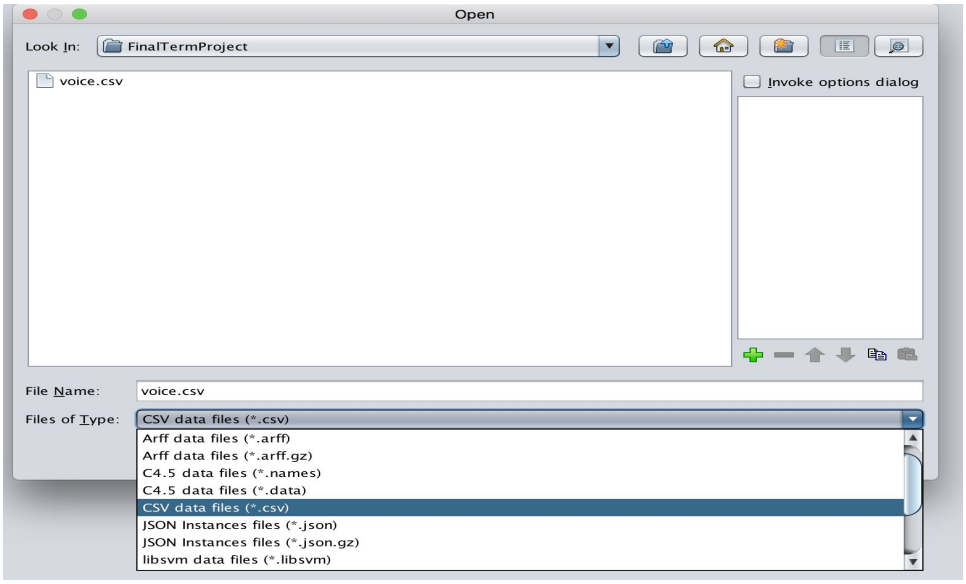
2. In Tools, click on ARFF Viewer



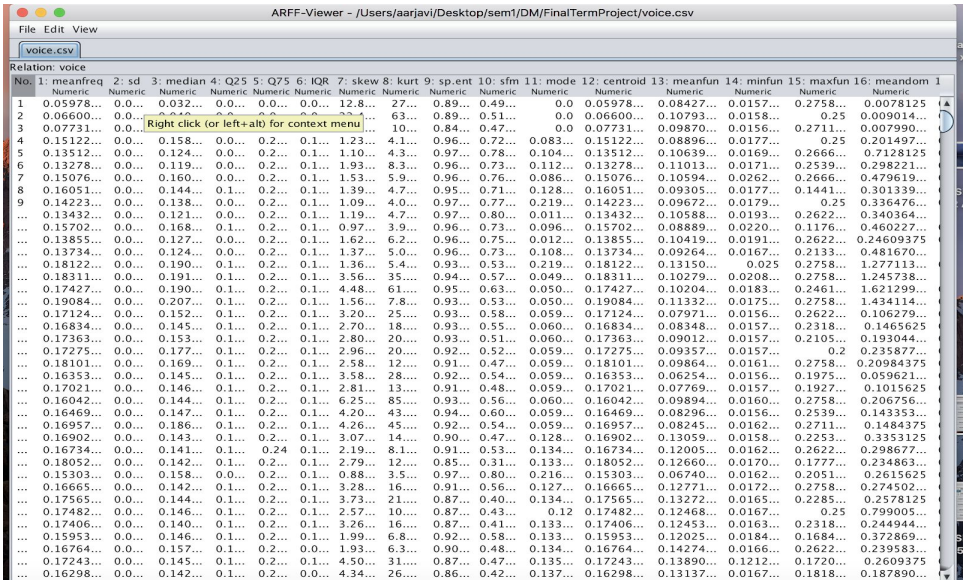
3. In ARFF Viewer, click on Files and then Click on Open



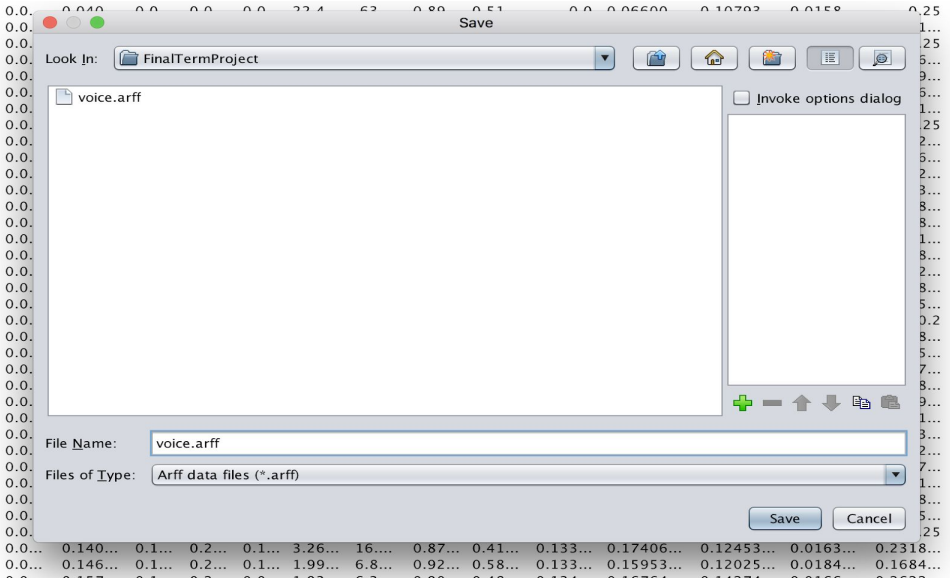
4. Browse the file : go to the directory containing dataset file
5. Click on File Type and Select .csv file format



6. Select the dataset : voice.csv , and click on Open



7. Save as the file in .arff format

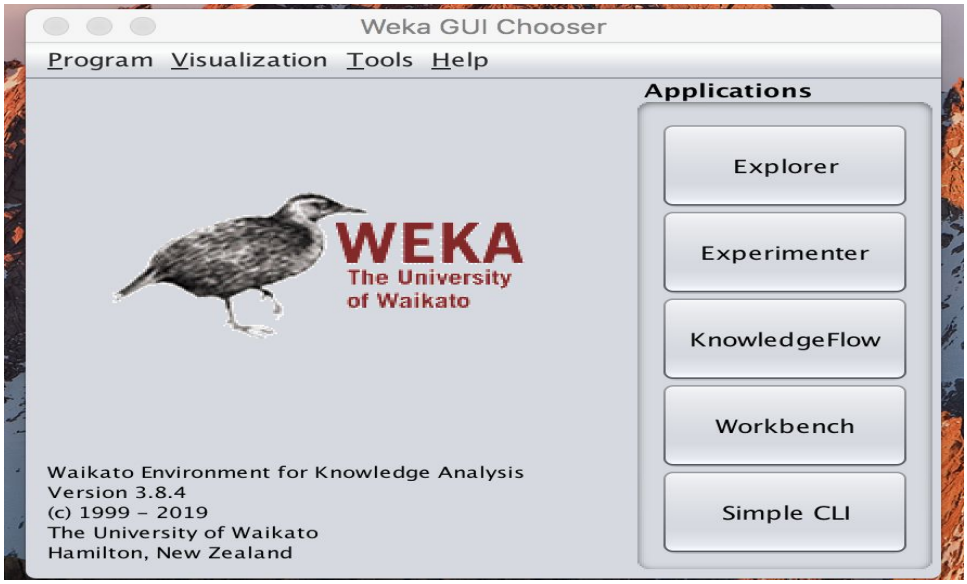


Chapter 7 : Generating the Results

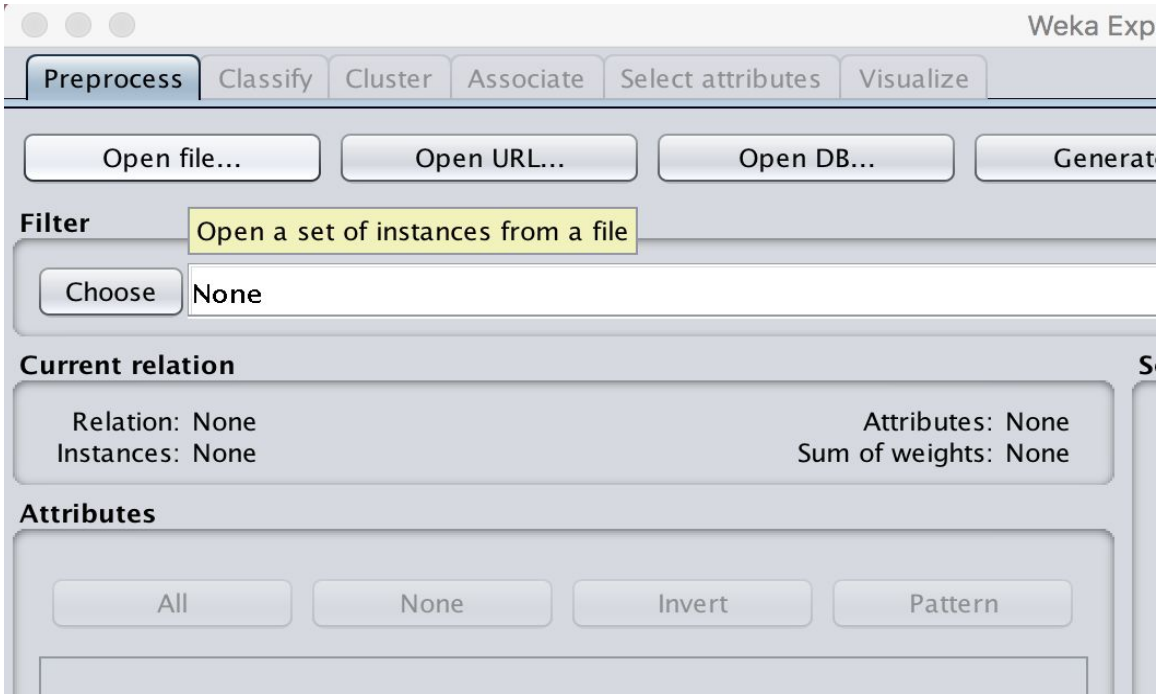
This chapter gives information and step by step guidance on how to use the Weka tool to generate results and ROC/AUC curve for the dataset by using Random Forest Algorithm (7.1) and Naive Bayes (7.2) in Weka tool. Source code for these algorithms are provided at the end of report in Appendix.

7.1 Random Forests

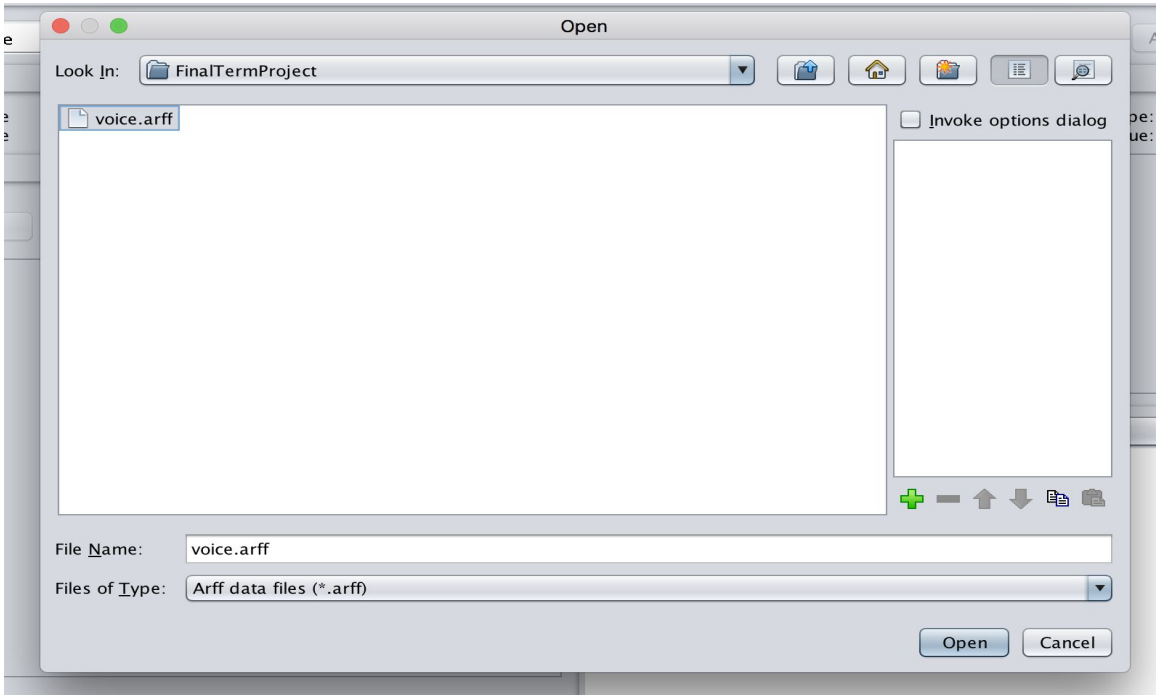
1. Open Weka and click on “Explorer”



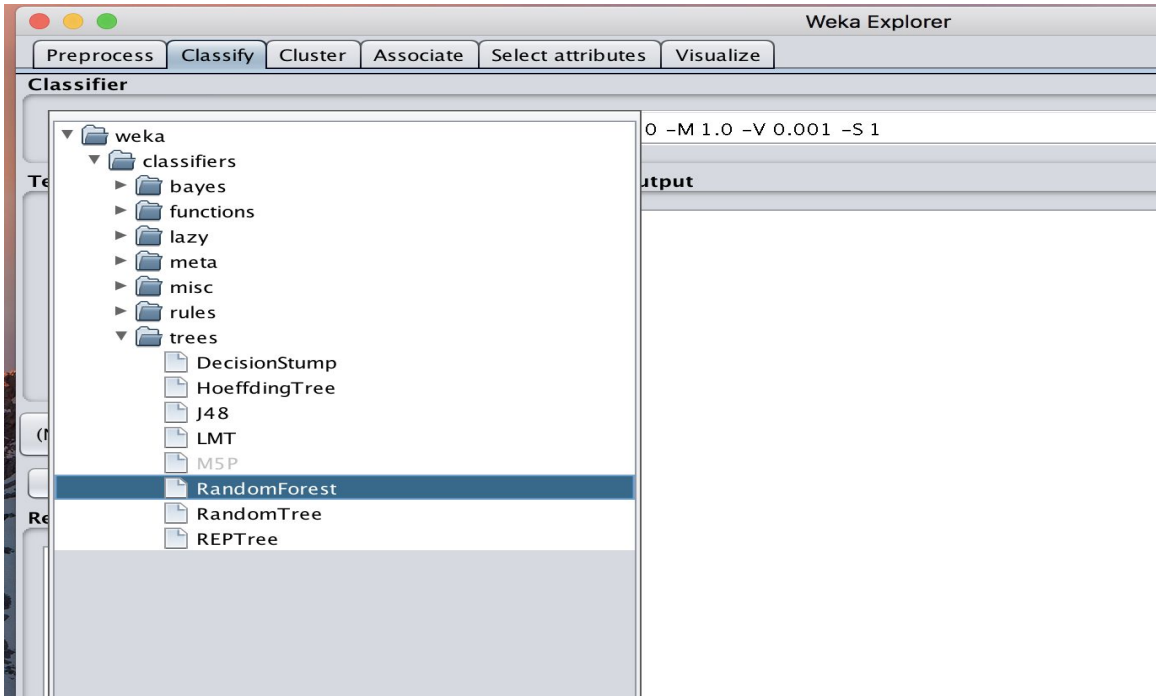
2. In the Tab “Preprocess”, click on “Open file…”



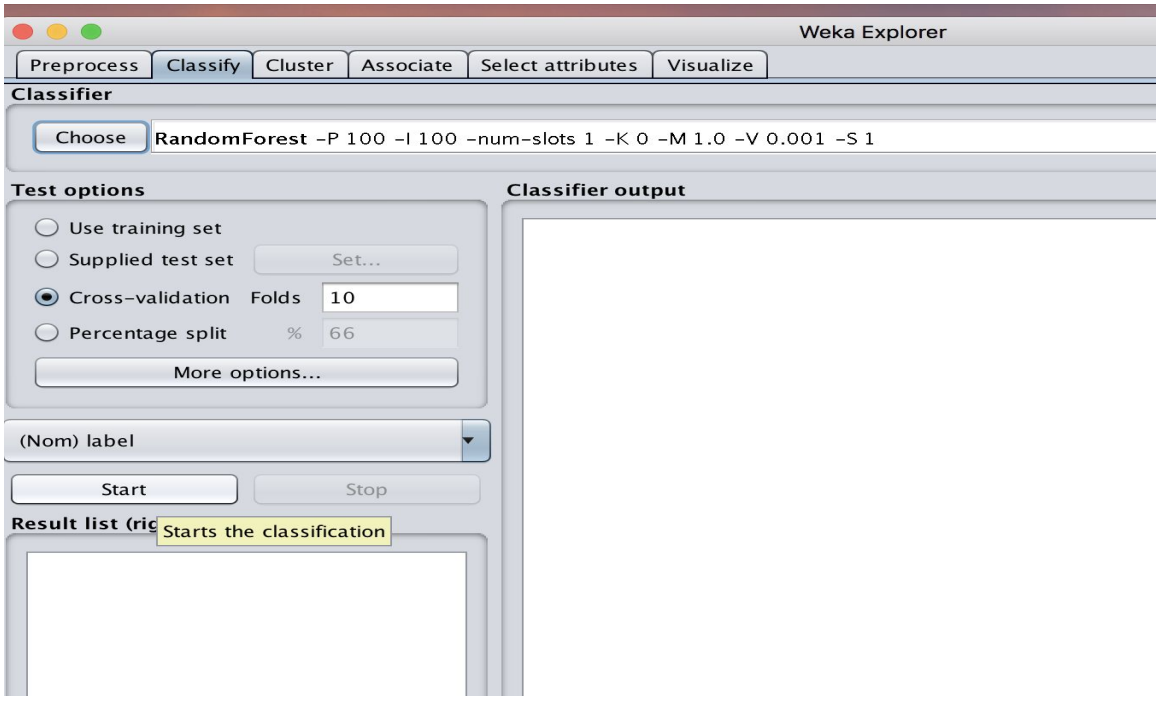
3. Browse through the file and go to the directory containing the data file: voice.arff, select the file and click on “Open”.



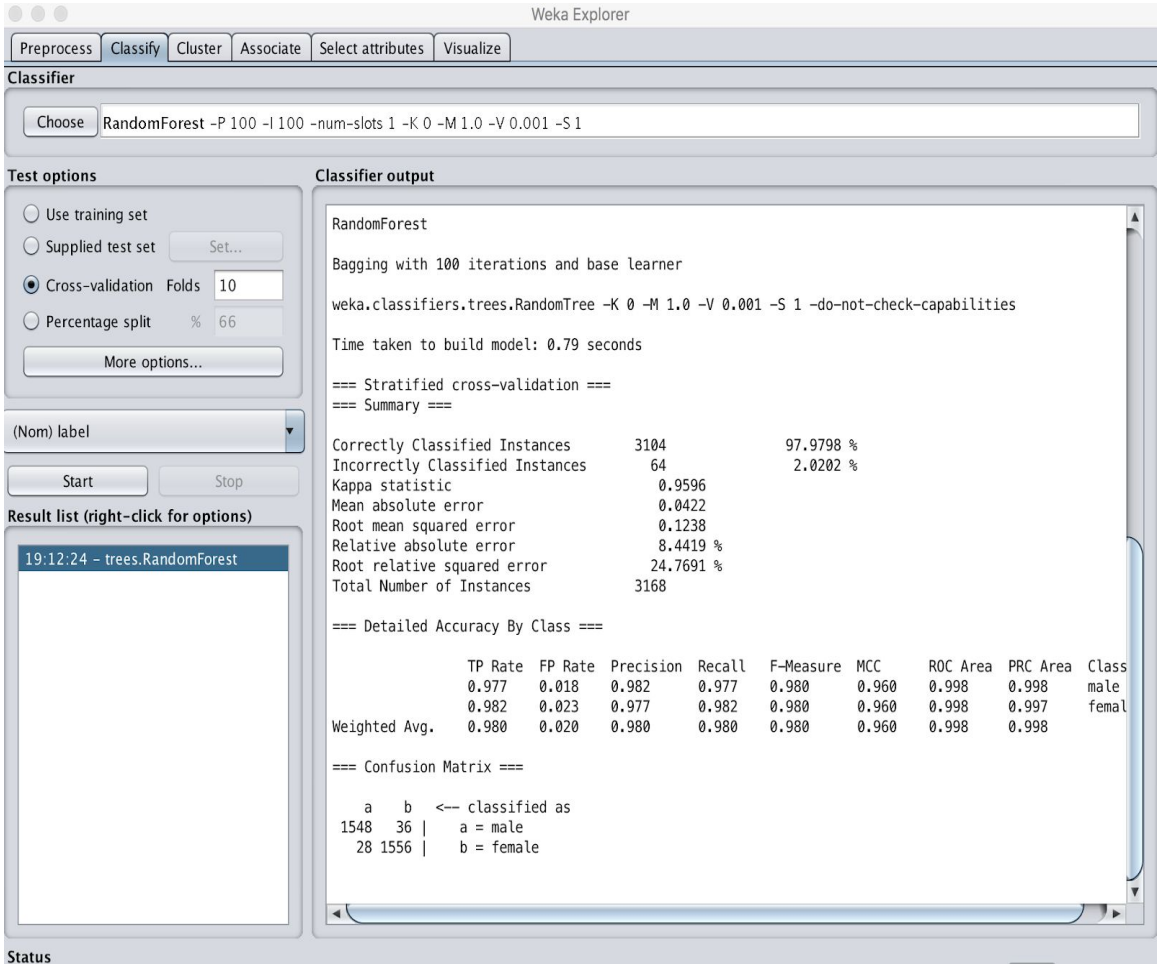
4. In “Classify” tab, click on “Choose” and select “RandomForest” in section “trees”



5. Click on “Start” button.

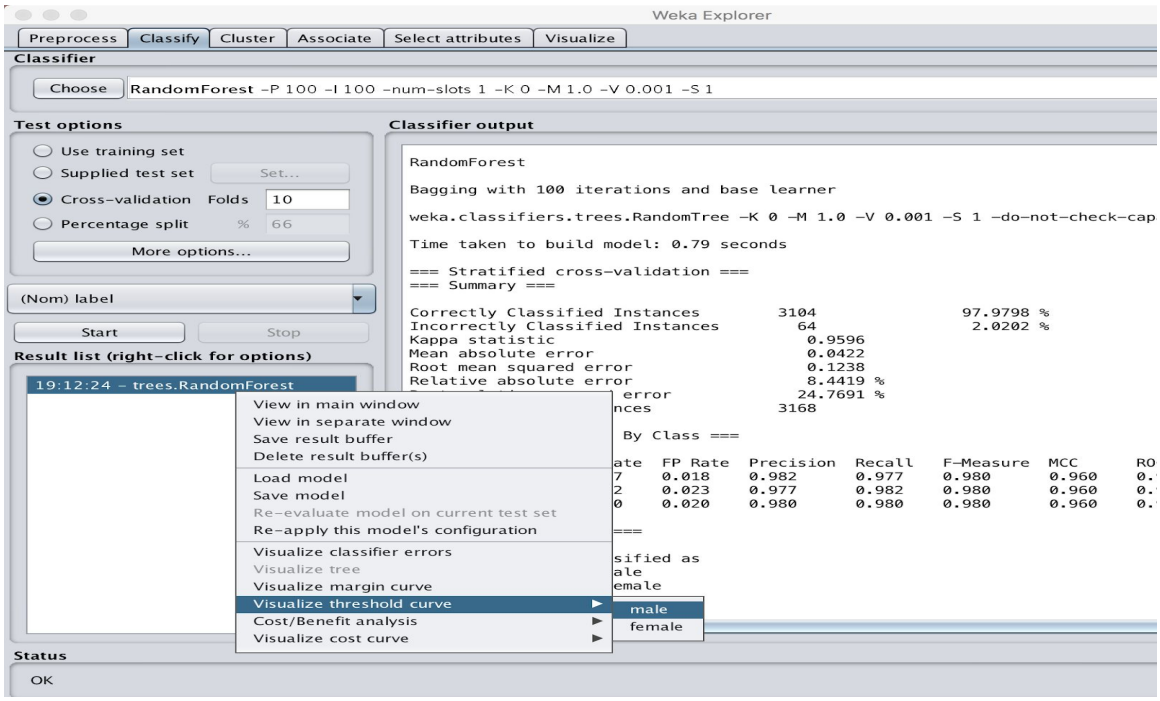


6. Output will be available in “Classifier Output” window

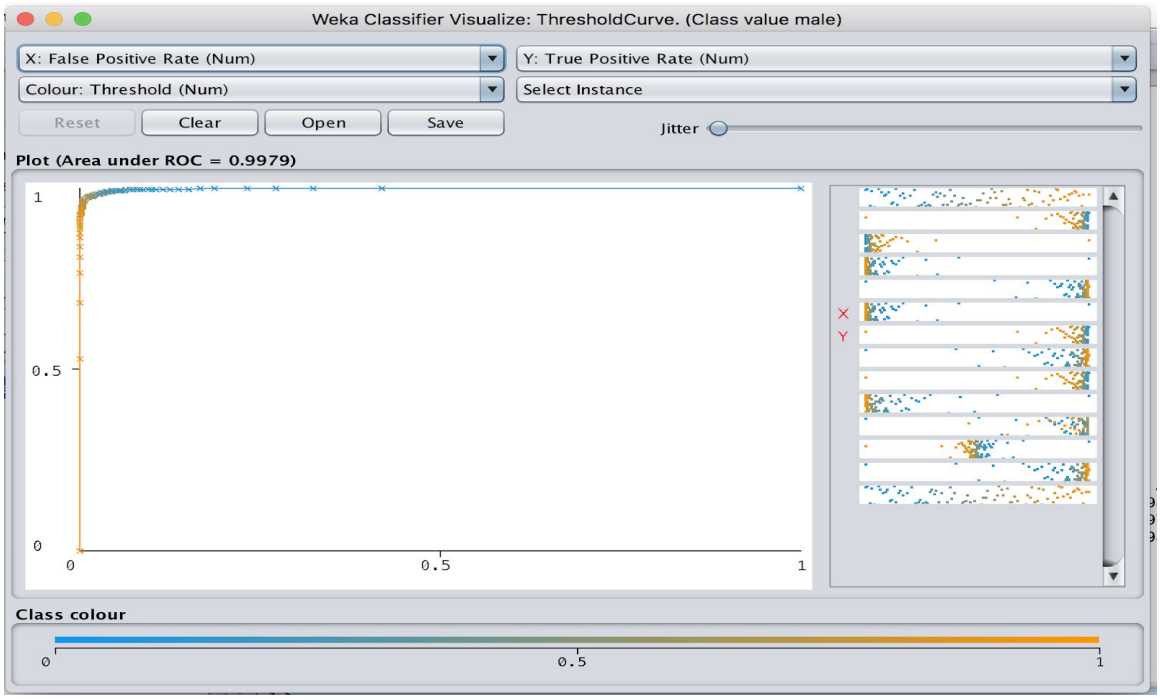


ROC / AUC Curve Generation

1. Right Click on the result in the “Result List” tab (in left-side panel) and in the option: “Visualize Threshold Curve”, click on “male” to generate ROC curve.



2. ROC curve generated with Area Under Curve as 0.9979 as shown below:-



Run information:-

Scheme: weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1
Relation: voice
Instances: 3168
Attributes: 21
meanfreq
sd
median
Q25
Q75
IQR
skew
kurt
sp.ent
sfm
mode
centroid
meanfun
minfun
maxfun
meandom
mindom
maxdom
dfrange
modindx
label
Test mode: 10-fold cross-validation

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

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.79 seconds

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

Correctly Classified Instances 3104 97.9798 %

Incorrectly Classified Instances 64 2.0202 %
Kappa statistic 0.9596
Mean absolute error 0.0422
Root mean squared error 0.1238
Relative absolute error 8.4419 %
Root relative squared error 24.7691 %
Total Number of Instances 3168

=== Detailed Accuracy By Class ===

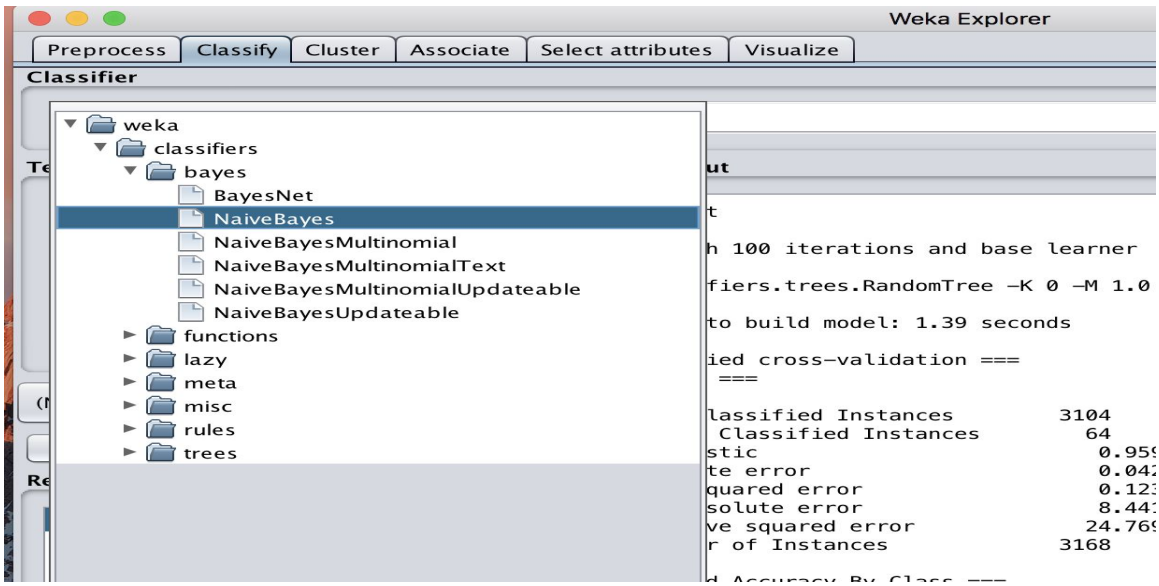
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.977	0.018	0.982	0.977	0.980	0.960	0.998	0.998	male
	0.982	0.023	0.977	0.982	0.980	0.960	0.998	0.997	female
Weighted Avg.	0.980	0.020	0.980	0.980	0.980	0.960	0.998	0.998	

=== Confusion Matrix ===

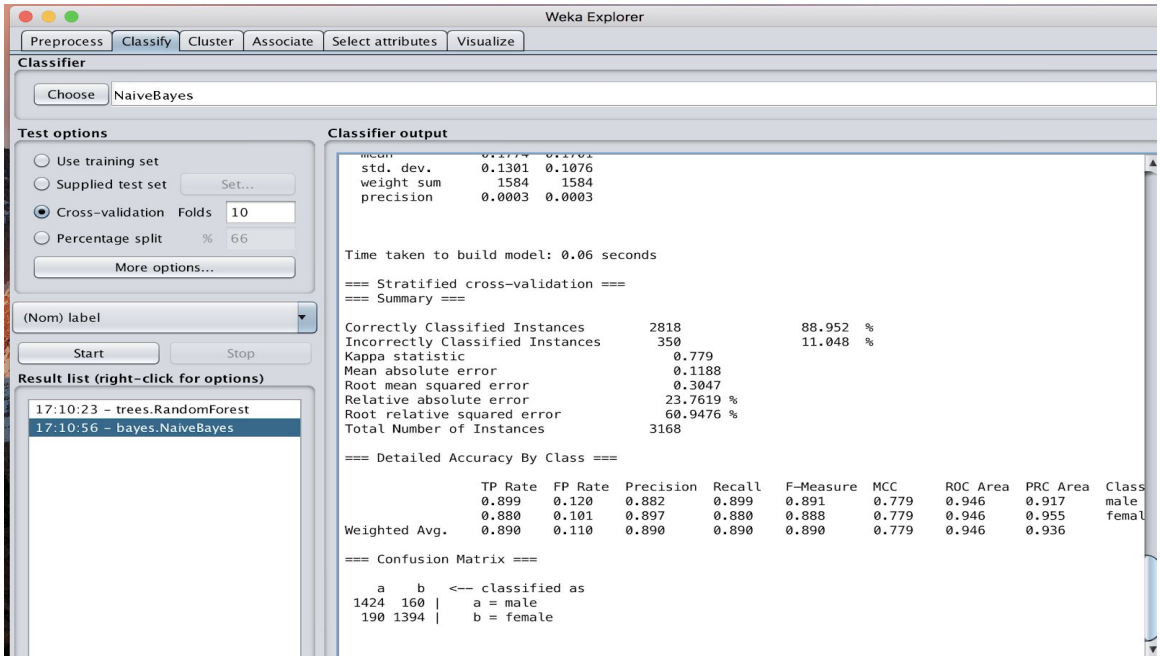
a	b	<-- classified as
1548	36	a = male
28	1556	b = female

7.2 Naive Bayes

1. Now, while our window is still open where we generated results for Random Forests. We will run Naive Bayes Algorithm in the same place with the same dataset file open. In the Classiy Tab, click on “Choose” and Select NaiveBayes under the path “bayes” as shown below:

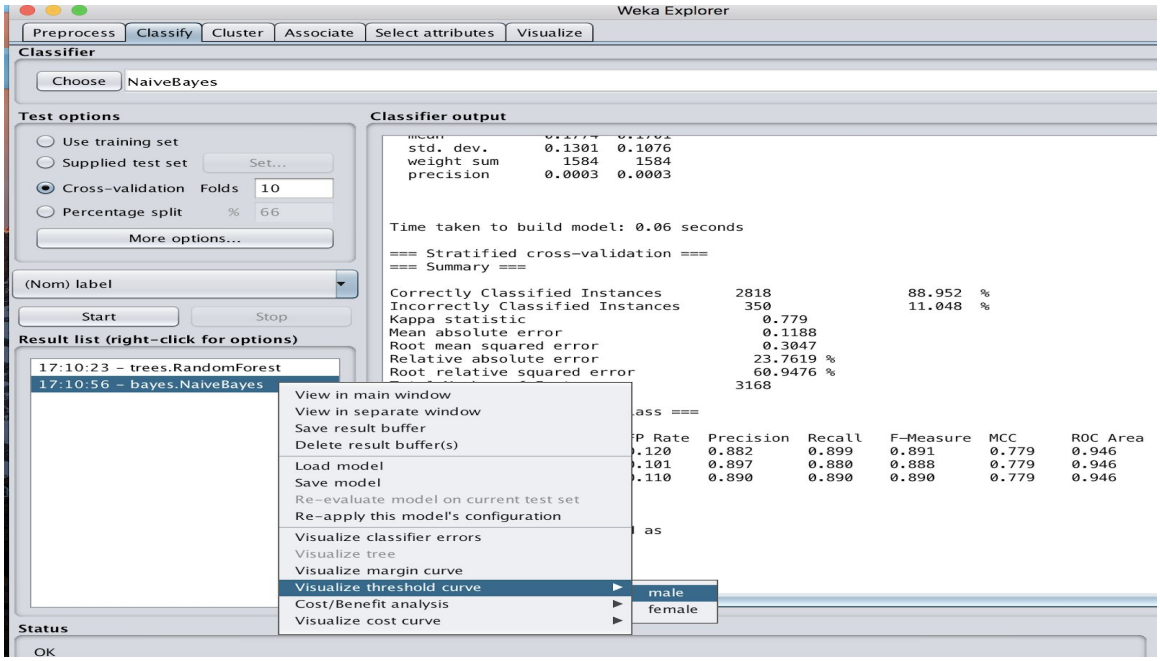


2. Click on Start and Output will be available in Classifier Window as shown below:

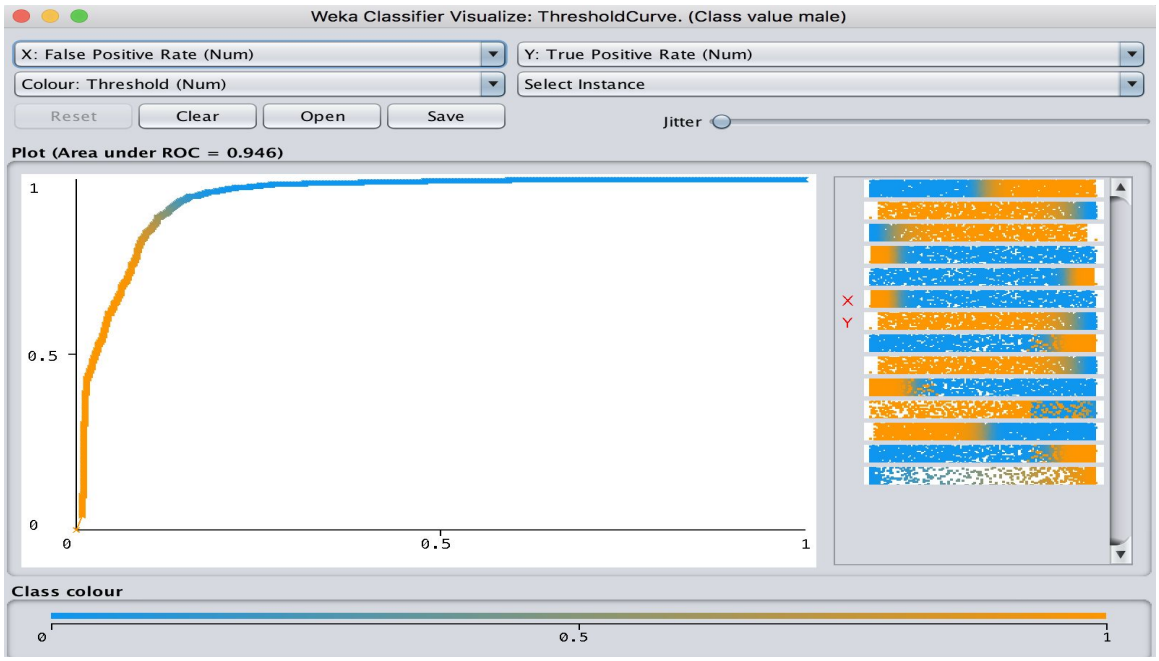


ROC / AUC Curve Generation

1. Right Click on the result in the “Result List” tab (in left-side panel) and in the option: “Visualize Threshold Curve”, click on “male” to generate ROC curve.



2. ROC curve generated with Area Under Curve as 0.946 as shown below:-



Run Information:-

```
Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     voice
Instances:    3168
Attributes:   21
              meanfreq
              sd
              median
              Q25
              Q75
              IQR
              skew
              kurt
              sp.ent
              sfm
              mode
              centroid
              meanfun
              minfun
              maxfun
              meandom
              mindom
              maxdom
              dfrange
              modindx
              label
Test mode:    10-fold cross-validation

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

Naive Bayes Classifier

                Class
Attribute         male  female
                (0.5)   (0.5)
==============================
meanfreq
  mean          0.1708   0.191
  std. dev.     0.0262    0.03
  weight sum      1584    1584
  precision     0.0001  0.0001

sd
```

	mean	0.0651 0.0491
	std. dev.	0.0095 0.0184
	weight sum	1584 1584
	precision	0 0
median		
	mean	0.1753 0.1959
	std. dev.	0.0374 0.0321
	weight sum	1584 1584
	precision	0.0001 0.0001
Q25		
	mean	0.1156 0.1653
	std. dev.	0.032 0.0498
	weight sum	1584 1584
	precision	0.0001 0.0001
Q75		
	mean	0.2263 0.2232
	std. dev.	0.024 0.0231
	weight sum	1584 1584
	precision	0.0001 0.0001
IQR		
	mean	0.1108 0.0578
	std. dev.	0.0204 0.0429
	weight sum	1584 1584
	precision	0.0001 0.0001
skew		
	mean	3.2954 2.9848
	std. dev.	5.1335 3.0905
	weight sum	1584 1584
	precision	0.0109 0.0109
kurt		
	mean	48.3345 24.7996
	std. dev.	163.0603 97.6375
	weight sum	1584 1584
	precision	0.4131 0.4131
sp.ent		
	mean	0.9172 0.8731
	std. dev.	0.0289 0.0473

	weight sum	1584 1584
	precision	0.0001 0.0001
sfm		
	mean	0.4717 0.3448
	std. dev.	0.1504 0.1798
	weight sum	1584 1584
	precision	0.0003 0.0003
mode		
	mean	0.152 0.1785
	std. dev.	0.084 0.0672
	weight sum	1584 1584
	precision	0.0001 0.0001
centroid		
	mean	0.1708 0.191
	std. dev.	0.0262 0.03
	weight sum	1584 1584
	precision	0.0001 0.0001
meanfun		
	mean	0.1159 0.1697
	std. dev.	0.0172 0.0185
	weight sum	1584 1584
	precision	0.0001 0.0001
minfun		
	mean	0.0342 0.0394
	std. dev.	0.0157 0.0218
	weight sum	1584 1584
	precision	0.0002 0.0002
maxfun		
	mean	0.2537 0.2637
	std. dev.	0.0359 0.0214
	weight sum	1584 1584
	precision	0.0014 0.0014
meandom		
	mean	0.7289 0.9295
	std. dev.	0.4458 0.5767
	weight sum	1584 1584
	precision	0.001 0.001

mindom
  mean   0.0401 0.0646
  std. dev.   0.0497 0.0726
  weight sum   1584   1584
  precision   0.006   0.006

maxdom
  mean   4.3581 5.7361
  std. dev.   2.9999 3.8528
  weight sum   1584   1584
  precision   0.0208 0.0208

dfrange
  mean   4.318 5.6712
  std. dev.   2.9998 3.8548
  weight sum   1584   1584
  precision   0.02   0.02

modindx
  mean   0.1774 0.1701
  std. dev.   0.1301 0.1076
  weight sum   1584   1584
  precision   0.0003 0.0003

Time taken to build model: 0.02 seconds

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

Correctly Classified Instances   2818   88.952 %
Incorrectly Classified Instances   350   11.048 %
Kappa statistic   0.779
Mean absolute error   0.1188
Root mean squared error   0.3047
Relative absolute error   23.7619 %
Root relative squared error   60.9476 %
Total Number of Instances   3168

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.899	0.120	0.882	0.899	0.891	0.779	0.946	0.917	male
	0.880	0.101	0.897	0.880	0.888	0.779	0.946	0.955	female
Weighted Avg.	0.890	0.110	0.890	0.890	0.890	0.779	0.946	0.936	

=== Confusion Matrix ===

a	b	<-- classified as
1424	160	a = male
190	1394	b = female

Chapter 8 : Optimization of Model Performance with Parameter tuning:-

This chapter gives step by step guidance on how to tune different parameters of an algorithm and generate comparison of the outputs to select the most optimized model for that algorithm

8.1 Optimizing Random Forest:-

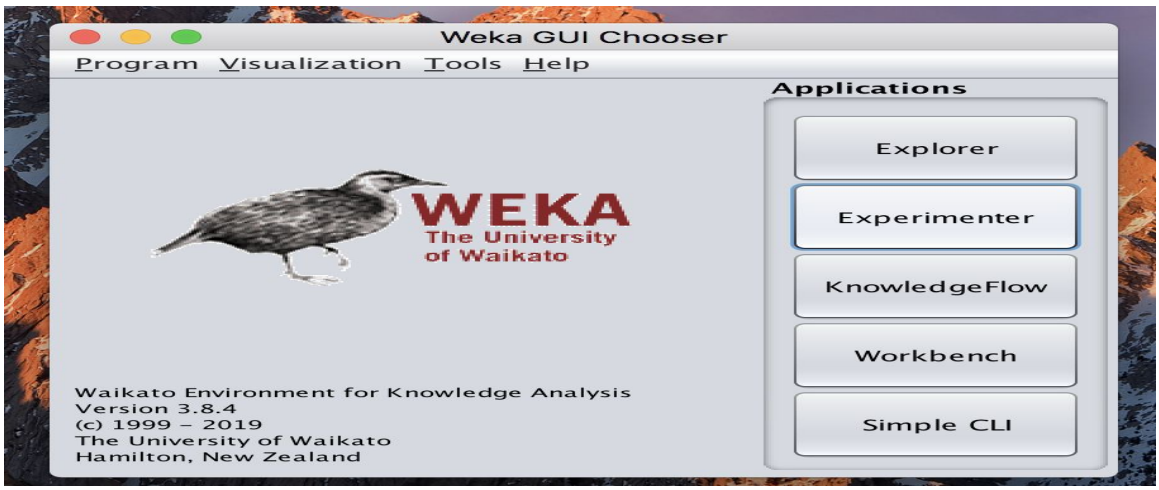
Weka gives a few input parameters for Random Forest Algorithm which can be tuned to select the best optimized model. This project demonstrates the tuning of parameter “numIterations”. Which is actually Number of trees in Random Forest. In Random Forest, the number of trees is basically how many different decision trees are used to generate results, before giving the final average output.

Increasing the number of trees improves efficiency but increases run time of the algorithm.

We have generated results for 4 different values of numIterations before selecting the most optimum model parameters.

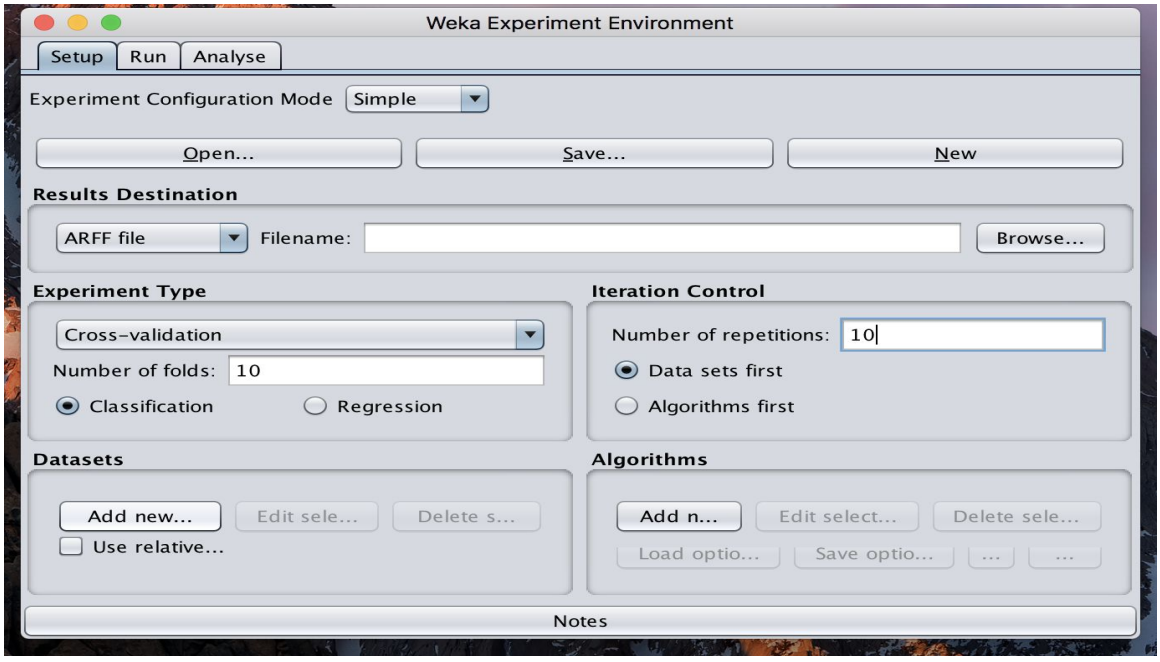
Below is the step by step guidance on the procedure.

1. Click on “Experimenter” in the Weka Opening Page.

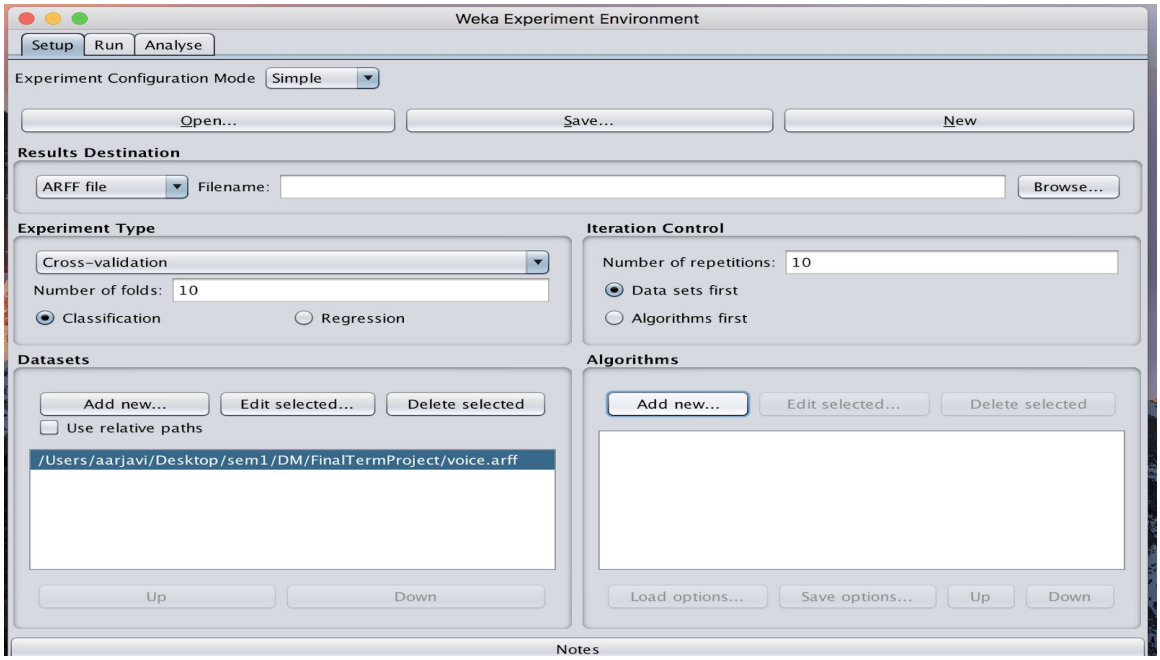


2. In the “Setup” Tab, click on “New” Button on the right most side of the screen.

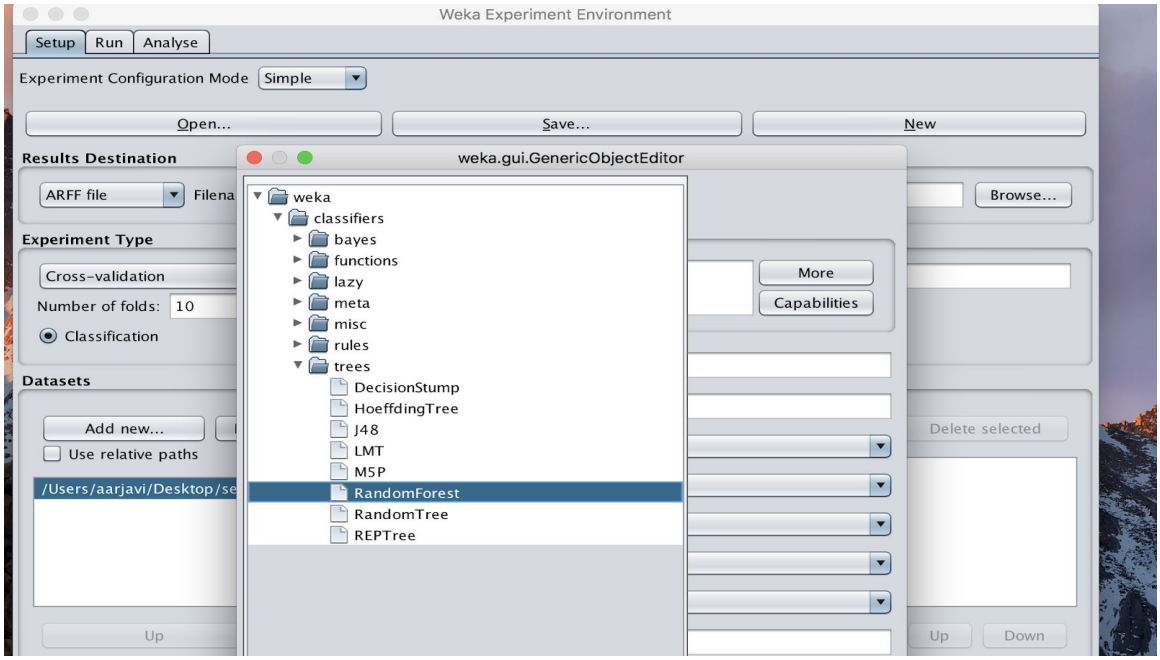
3. Set “Number of folds” to 10 in “Experiment Type” section and set “Number of repetitions” to 10 in “Iteration Control” window.



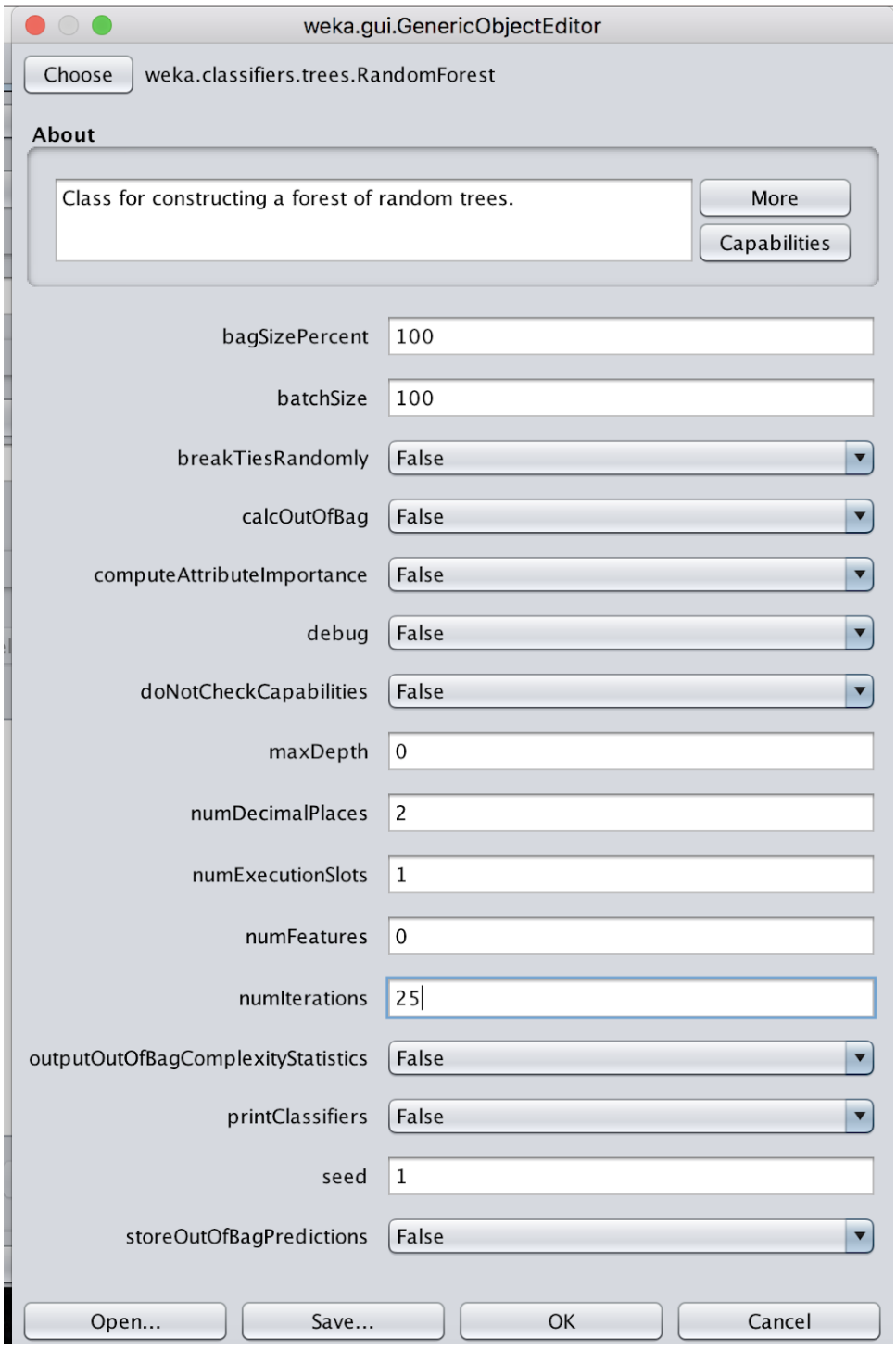
4. In the “Datasets” section, click on “Add new..” button and browse through the dataset file :- voice.arff as shown below.



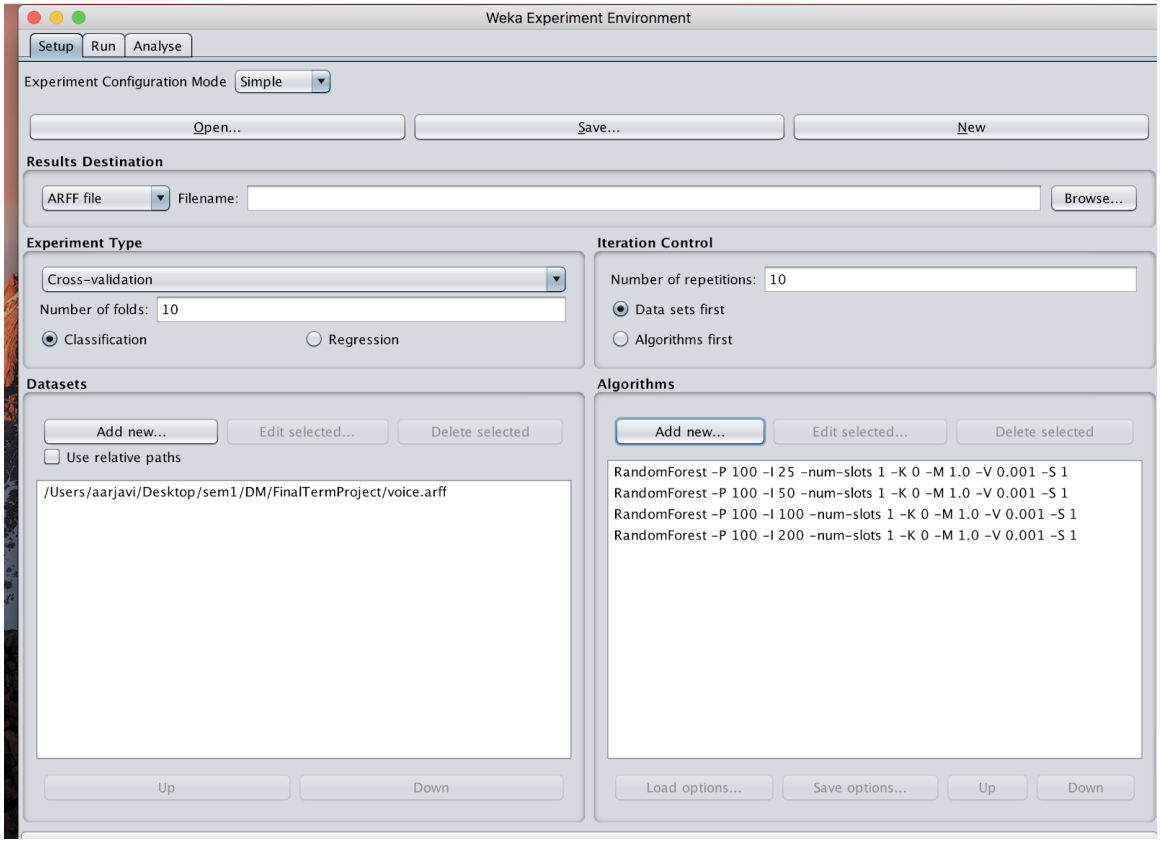
5. In “Algorithm” section, click on “Add new..” button and select “RandomForest” algorithm.



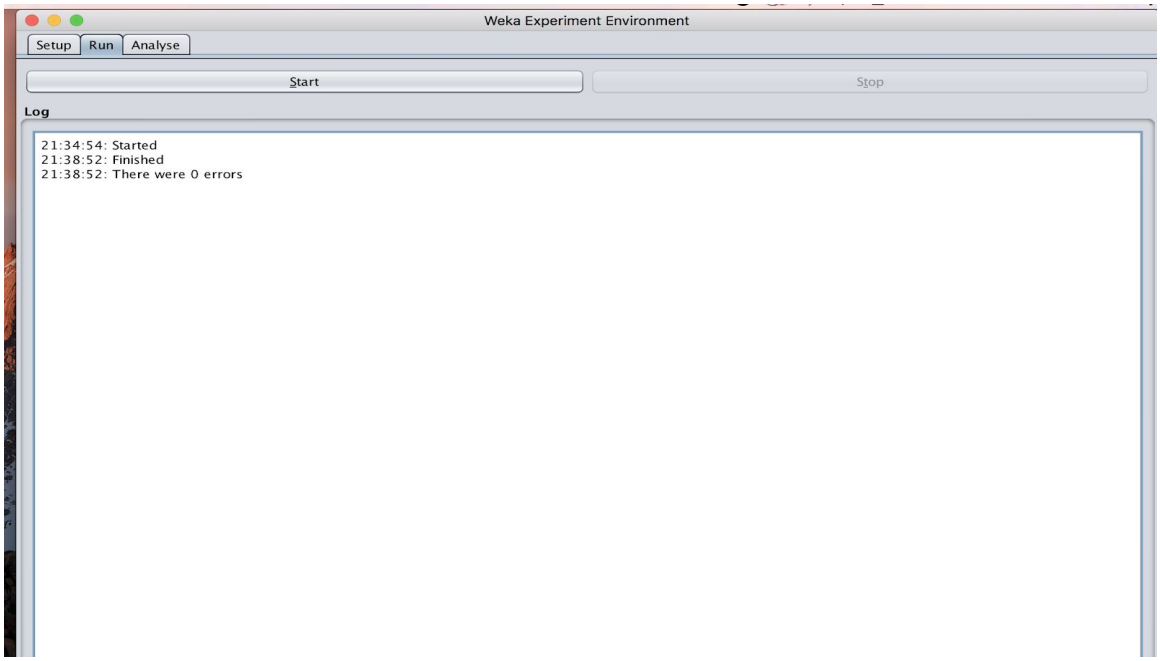
6. Change the value of “numIterations” to 25 and click on OK.



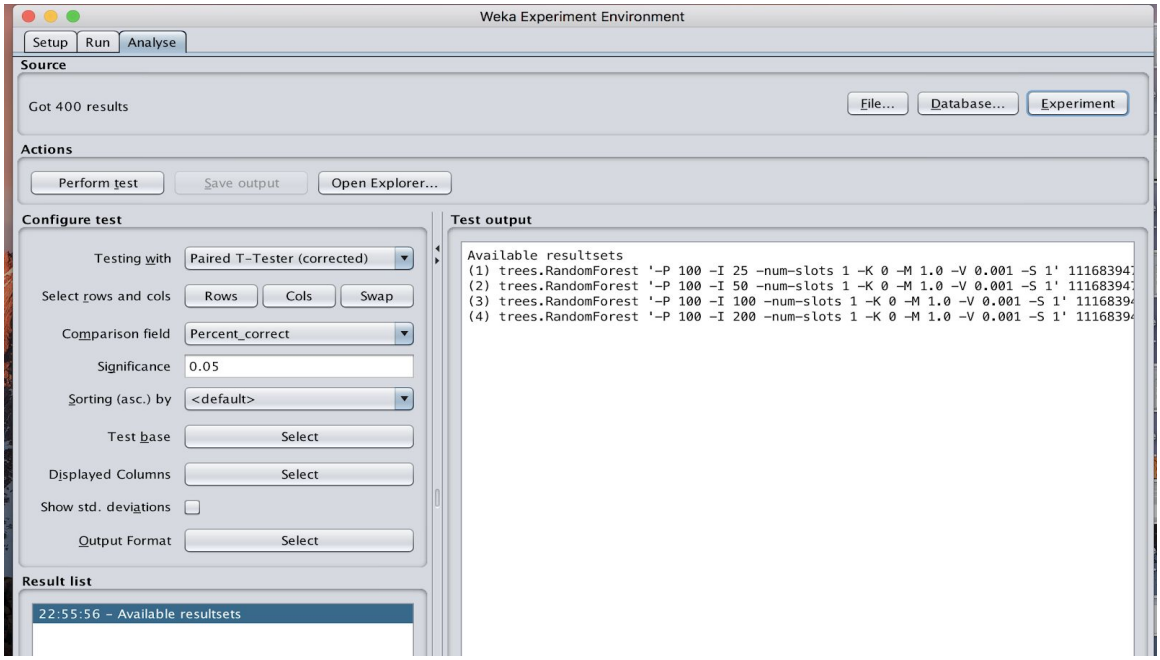
7. Similarly , repeat the process of adding a new algorithm 3 more times, every time select Random Forest Algorithm, but keep the value of “numIterations” to 50, 100 and 200 in 2nd, 3rd and 4rth time respectively. The Algorithm section will look as shown below.



8. In the “Run” tab, click on “Start” button, and wait till the process is completed. If the algorithms ran successfully on the dataset, a message with 0 errors will be printed as shown below.

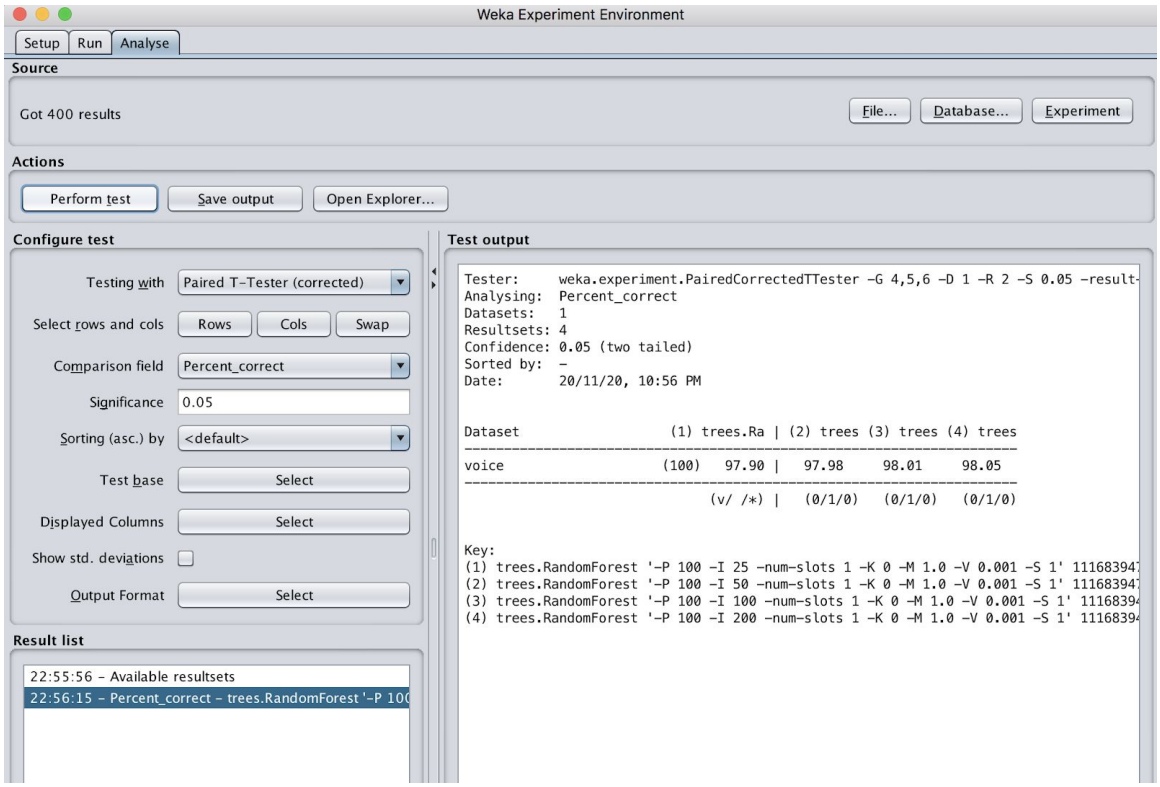


9. In the “Analyse” tab, in the “Source” section, click on “Experiment” button.

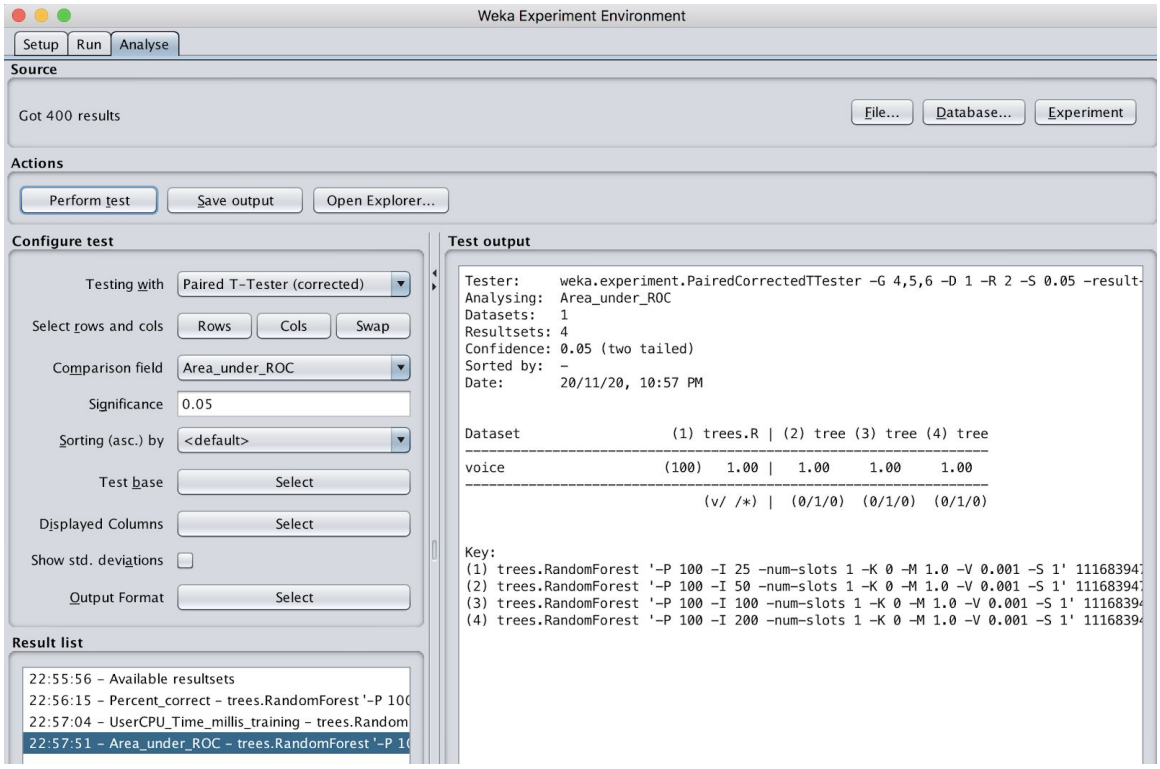


10. In the “Actions” section, click on “Perform Test” button to start the test. Once the test is successfully completed, a detailed comparison report will be available in “Test Output” window as shown below:

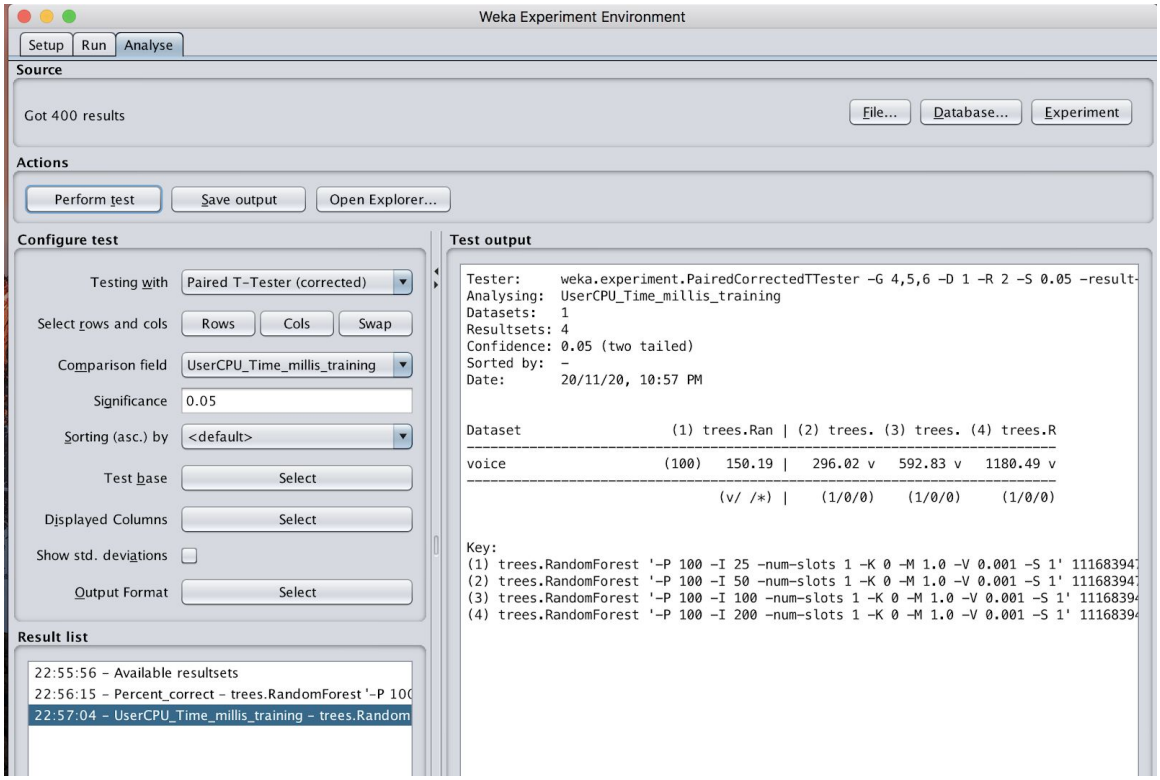
    The below report is for “Comparison filed” - Percent_correct, or in other words accuracy.



11. Now, from the drop down menu in “Comparison Field” in “Configure Test “ section, select “Area_under_ROC” and click on “Perform test” button to generate below comparison result between all the 4 models for Area under ROC curve.



12. Now, from the drop down menu in “Comparison Field” in “Configure Test “ section, select “UserCPU_Time_millis_training” and click on “Perform test” button to generate below comparison result between all the 4 models for Time required for training.



Summary of comparison for different parameters :-

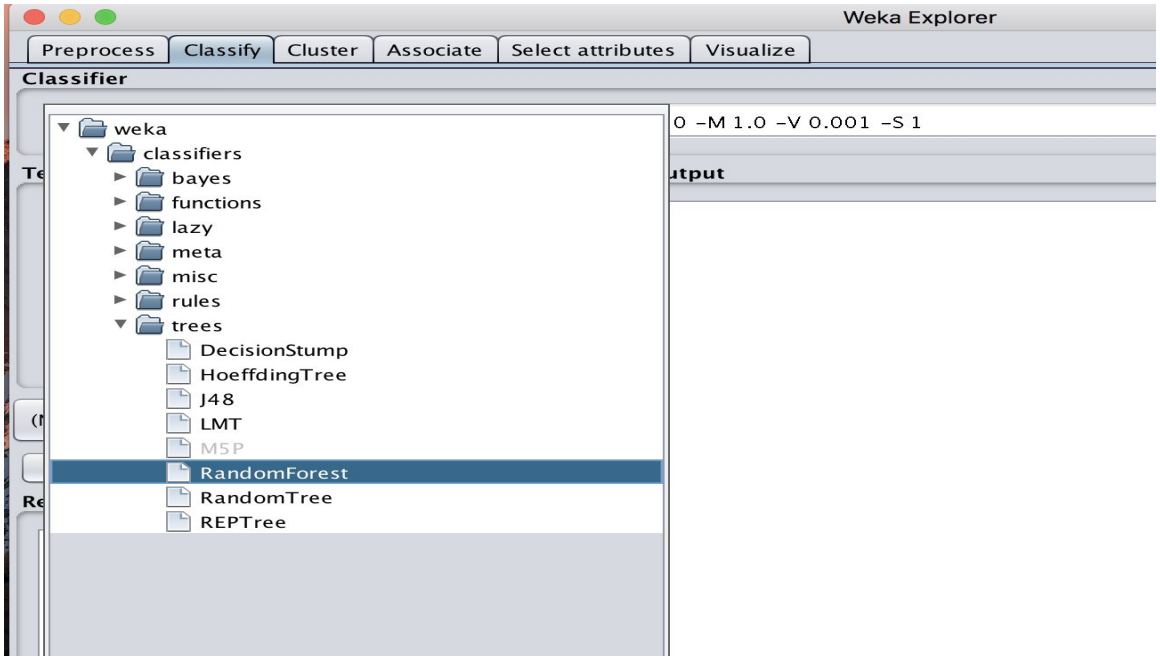
Random Forest Model with number of trees =	Accuracy (%)	Percentage Increase w.r.t base	Area under ROC	Training Time (ms)	Percentage Increase w.r.t base
25	97.9	-	1	150.19	-
50	97.98	0.08 %	1	296.02	97 %
100	98.01	0.11 %	1	592.83	294.7 %
200	98.05	0.15 %	1	1180.49	685.99 %

As shown above, the percentage increase in accuracy is negligible w.r.t to the percentage increase in training time as we increase the number of trees from 25 to a higher value. So the most optimum model parameter according to me which optimizes time and gives an almost best accuracy is with the number of trees 25. This model will be used in chapter 9 to compare against Naive Bayes algo.

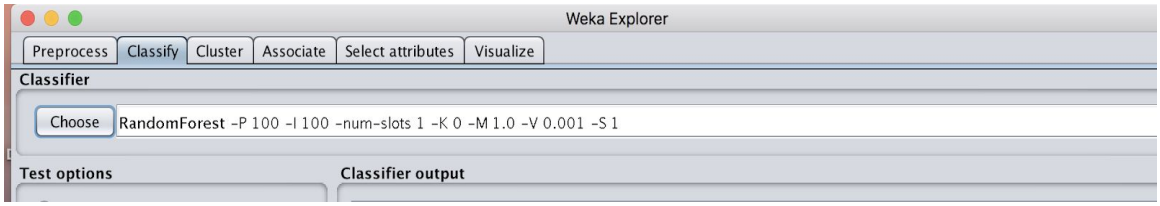
Generating results and ROC curves for this optimized Random Forest Model (numIterations = 25) :-

1. Add dataset file: voice.arff to Explorer in Weka, in Preprocess tab as shown by steps 1-3 in section 7.1.

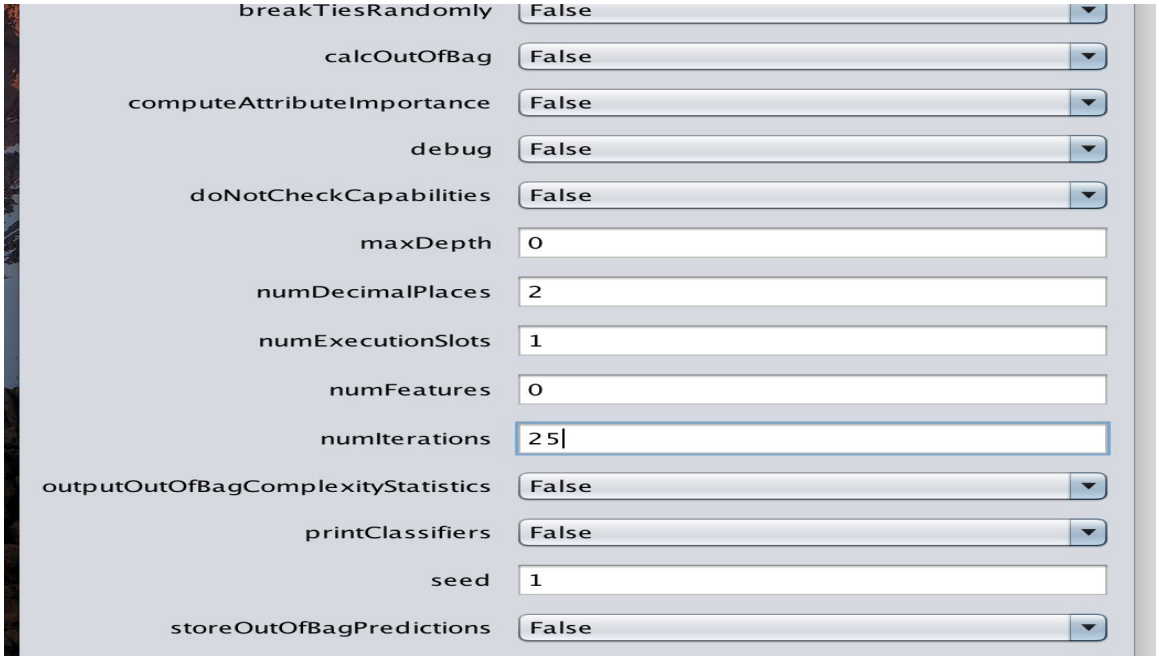
2. In “Classify” tab, click on Choose and Select RandomForest algorithm.



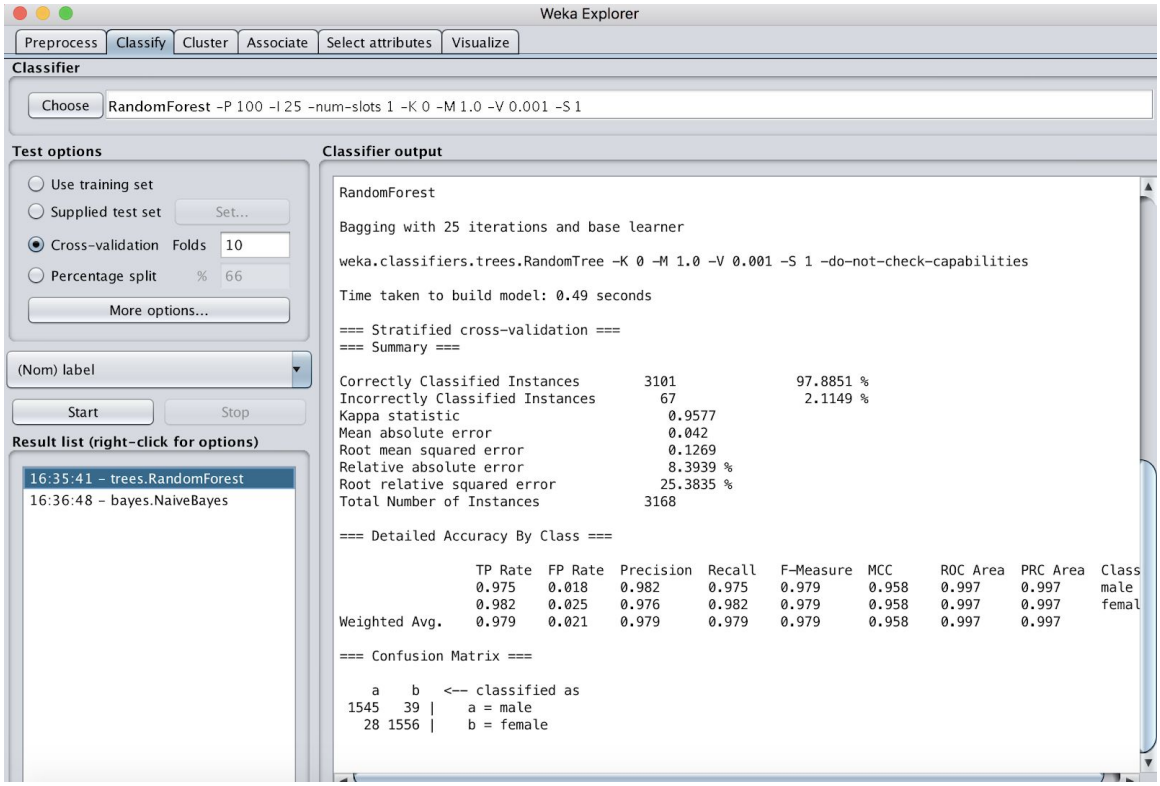
3. In the small window next to “Choose” , click on “RandomForest -P 100……” line



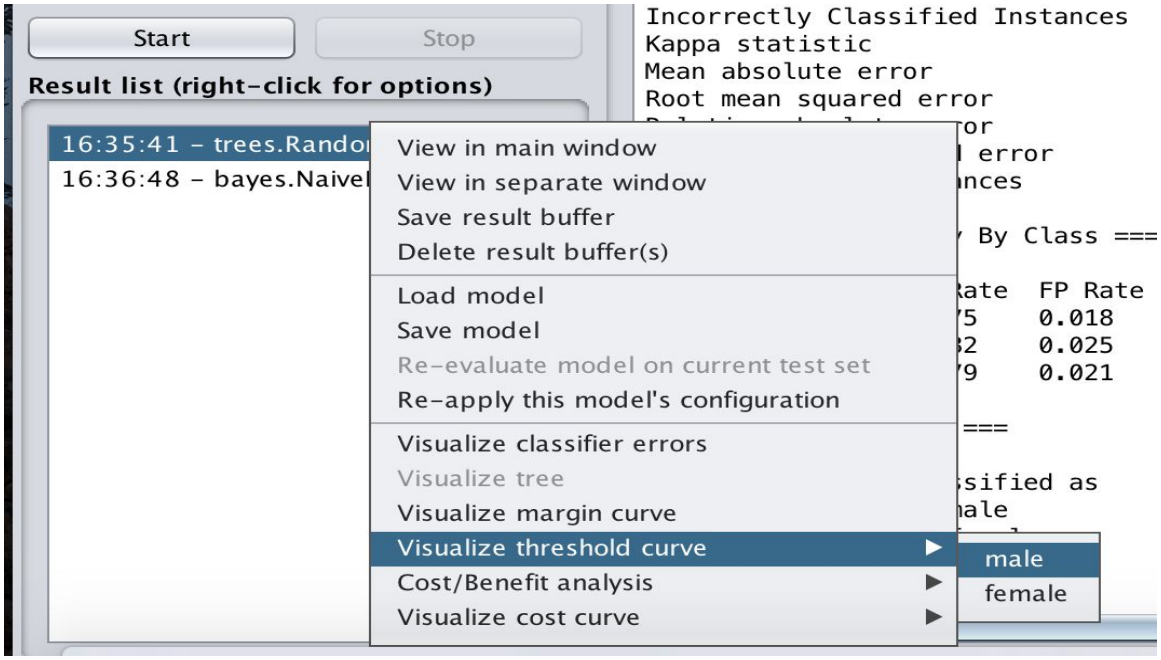
4. It will open parameter selection window for this algo. Change the value of numIterations to 25 and click on OK.

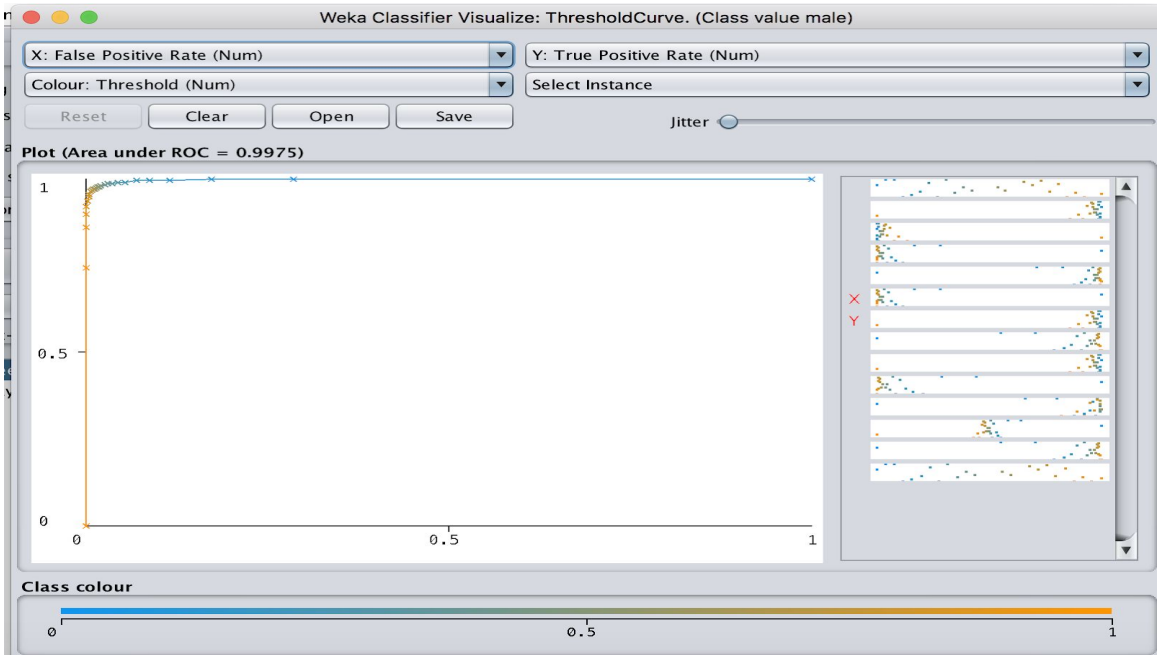


5. Click on Start button and result will be available in “Classifier output” window as below:



6. Right click on Random Forest result, in Results List, and in “Visualize threshold curve” click on male and ROC curve will be generated as below.





Result Summary for the optimized model where parameter numIterations = 25:-

Accuracy:-	97.885 %
Area under ROC:-	0.997
Precision :-	0.982
Recall:-	0.975
F1:-	0.979
Time Taken to build the Model:-	0.49 sec

8.2 Optimizing Naive Bayes:-

Weka gives 2 input parameters for Naive Bayes Algorithm which can be tuned to select the best optimized model. This project demonstrates the tuning of parameters “useKernelEstimator” and “useSupervisedDiscretization” for Naive Bayes Algorithm.

A kernel is a weighting function used in non-parametric estimation techniques. Kernels are used in kernel density estimation to estimate random variables' density functions, or in kernel regression to estimate the conditional expectation of a random variable.

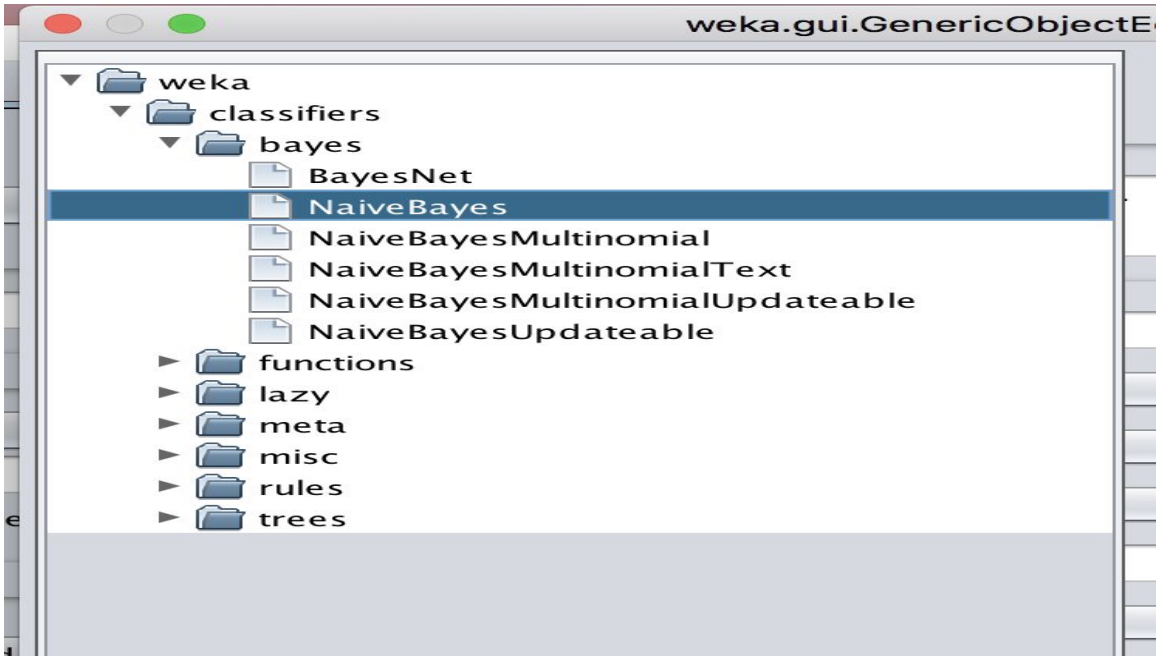
Supervised discretization methods take the class into account when setting discretization boundaries

Before the selection of most optimum model, we have generated results for 3 different models where:-

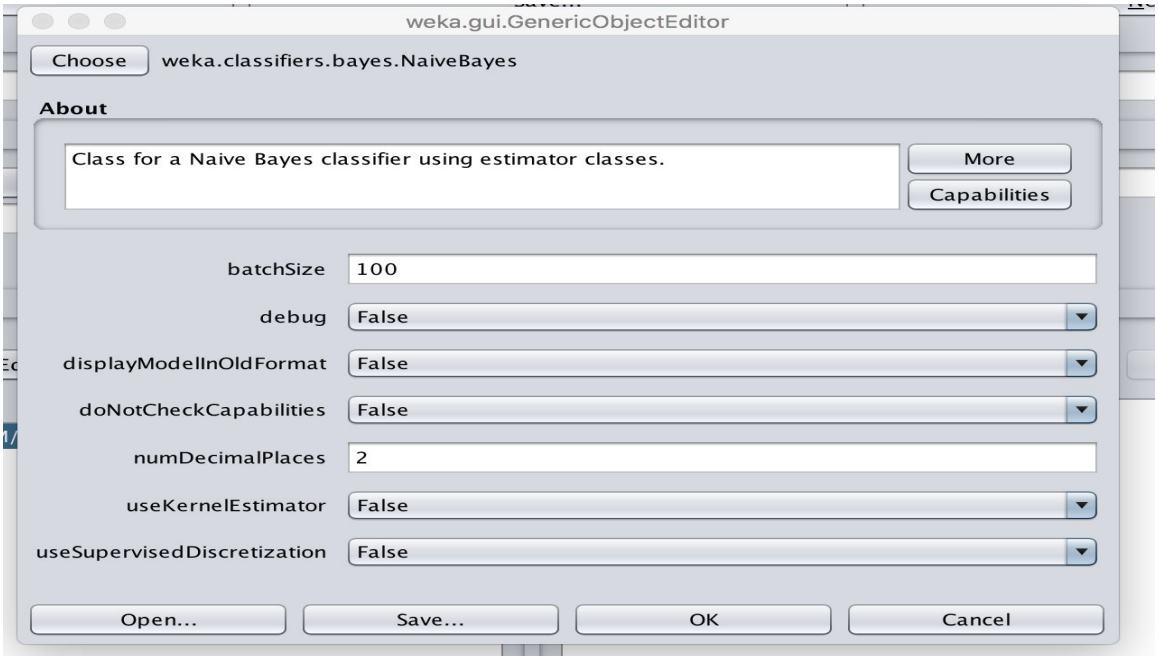
1. useKernelEstimator =False and useSupervisedDiscretization =False
2. useKernelEstimator =True and useSupervisedDiscretization =False
3. useKernelEstimator =False and useSupervisedDiscretization =True

Below is the step by step guidance on the procedure.

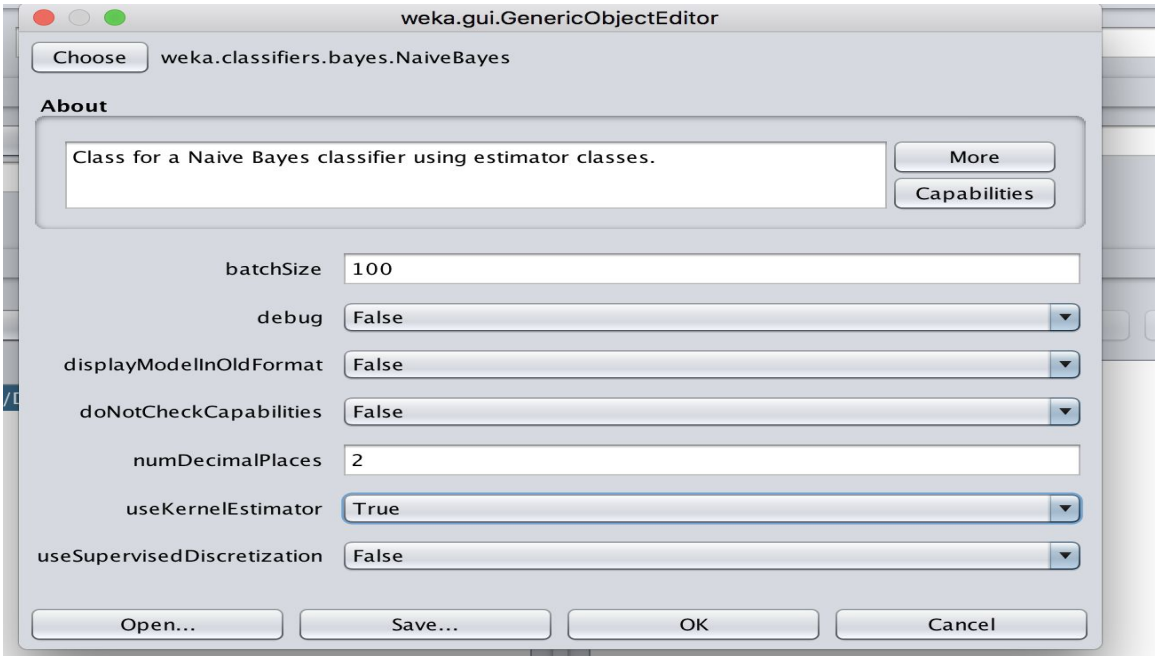
1. Repeat the steps 1 to 4 from section 8.1 to add the dataset file to the Experimenter in Weka.
2. In “Algorithm” section, click on “Add new..” button and select “Naive Bayes” Algorithm



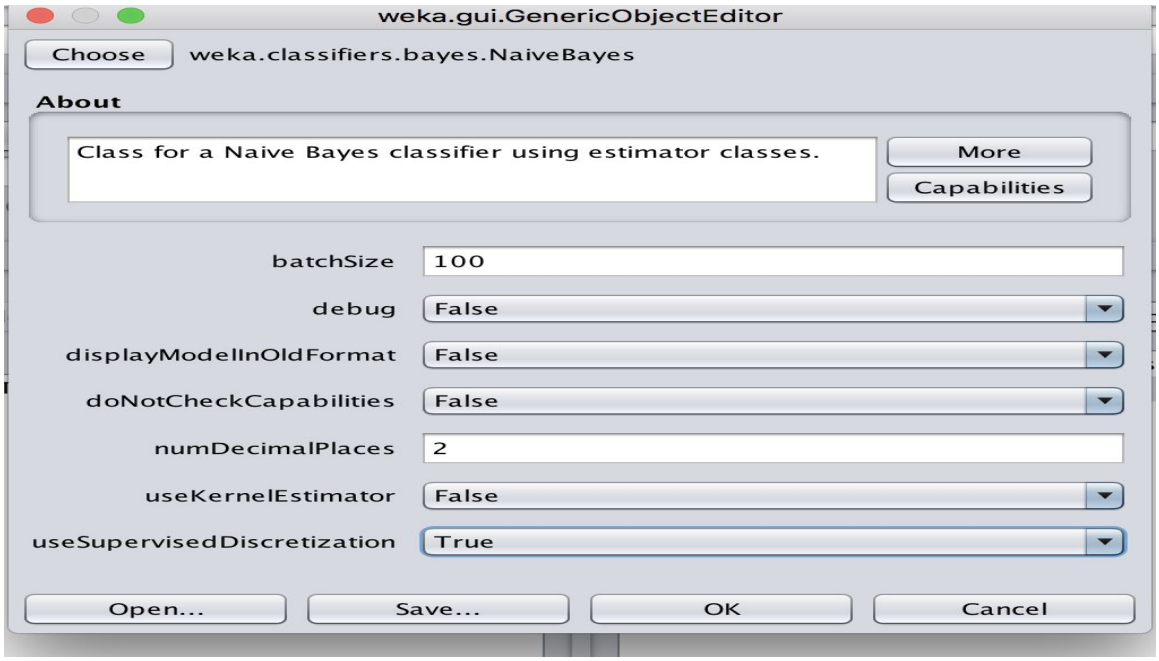
3. We will keep the default parameters for “useKernelEstimator” = False and “useSupervisedDiscretization” = False for the first model and click on OK.



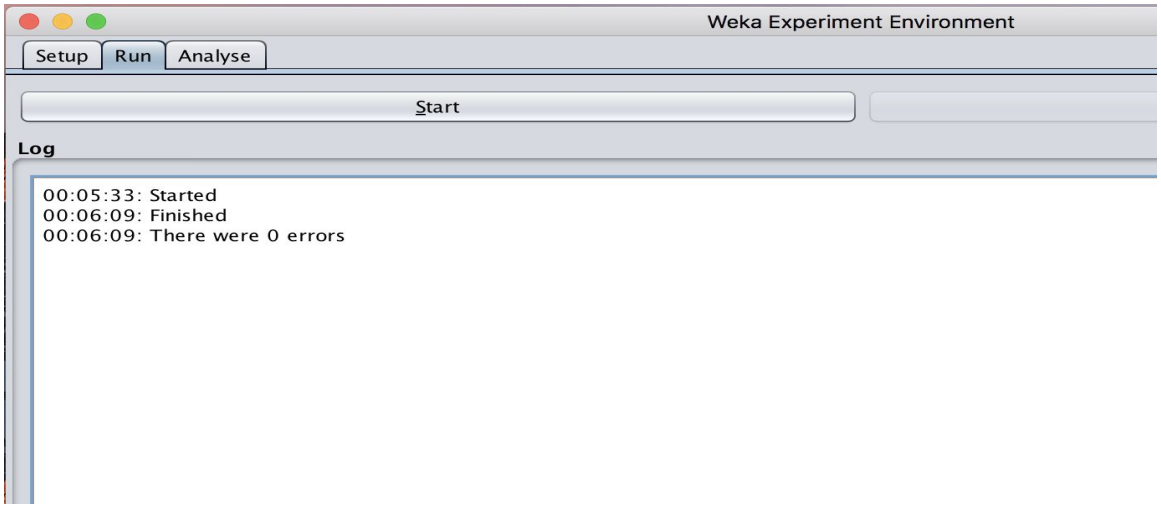
4. Click on “Add new” button again, select Naive Bayes algorithm, change value of “useKernelEstimator” to “True” and click on OK.



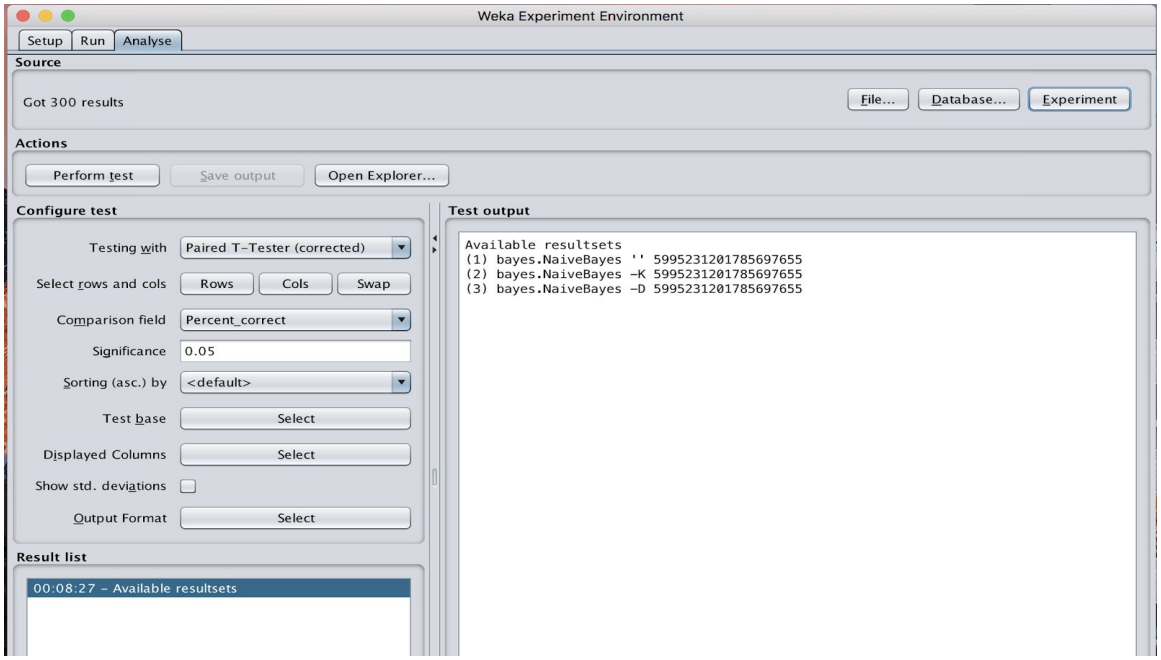
5. Click on “Add new” button again, select Naive Bayes algorithm, change value of “useKernelEstimat” to “False” and value of “useSupervisedDiscretization” to True and click on OK.



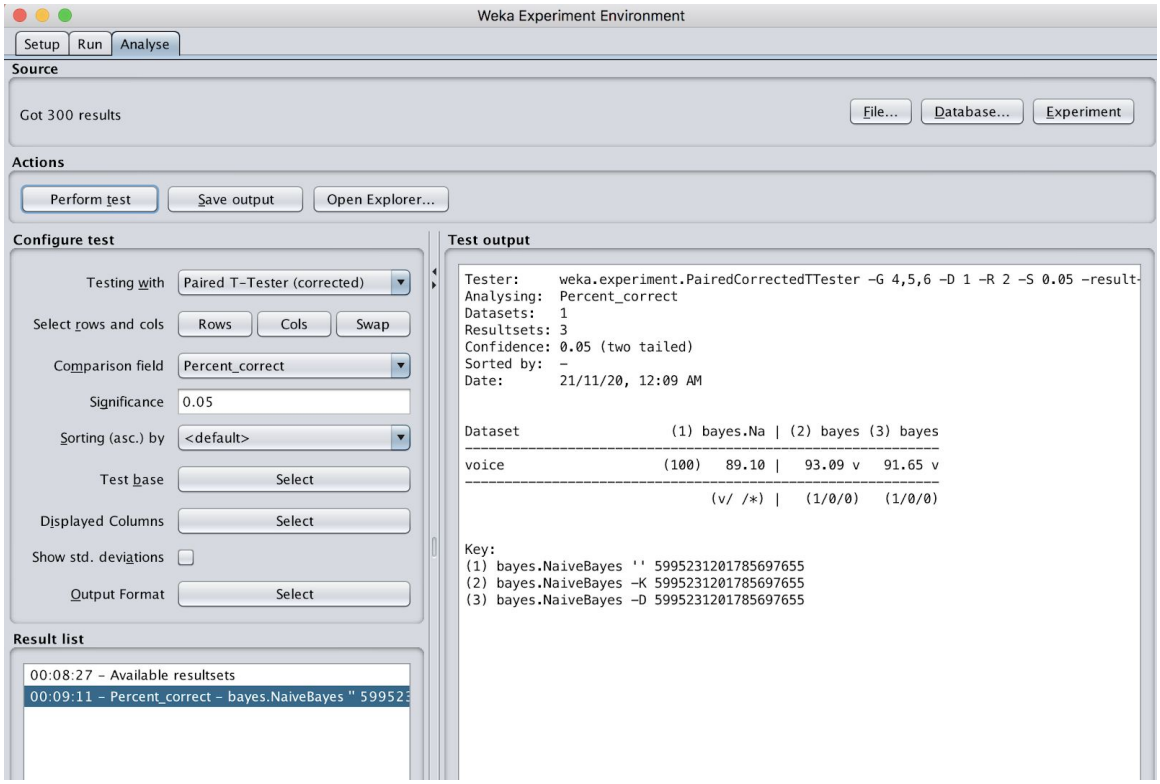
6. In the “Run” tab, click on “Start” button, and wait till the process is completed. If the algorithms ran successfully on the dataset, a message with 0 errors will be printed as shown below.



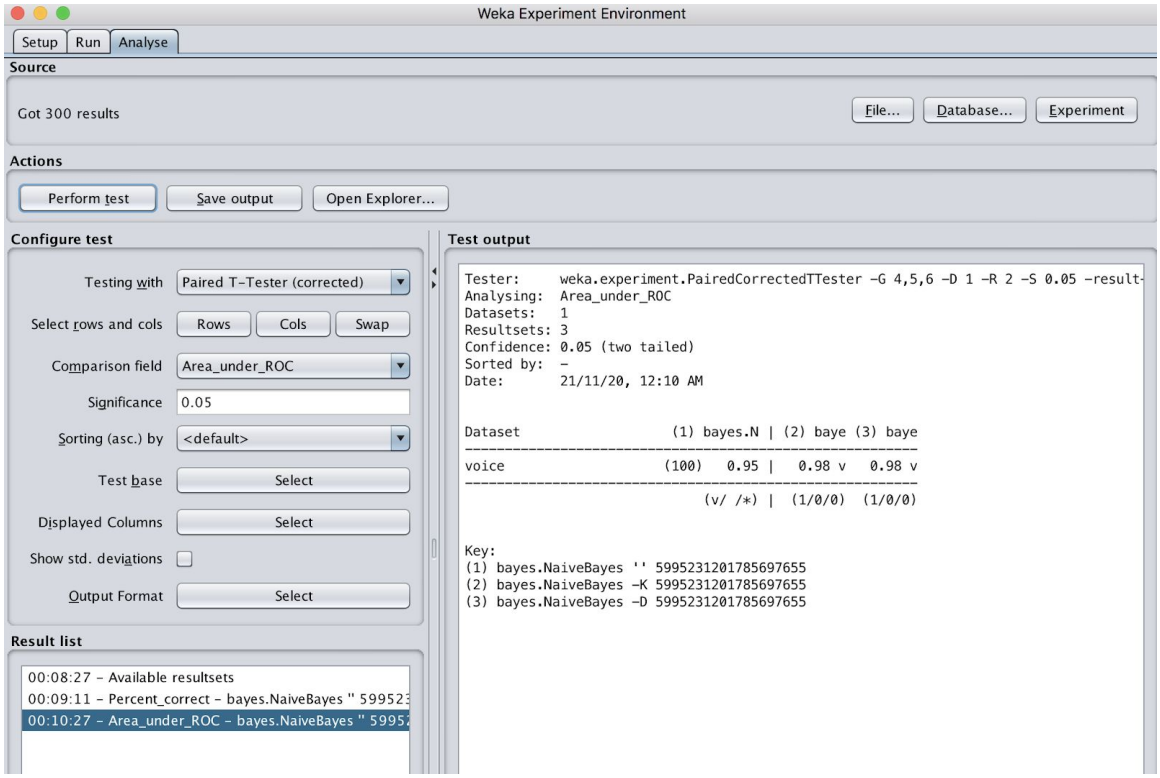
7. In the “Analyse” tab, in the “Source” section, click on “Experiment” button.



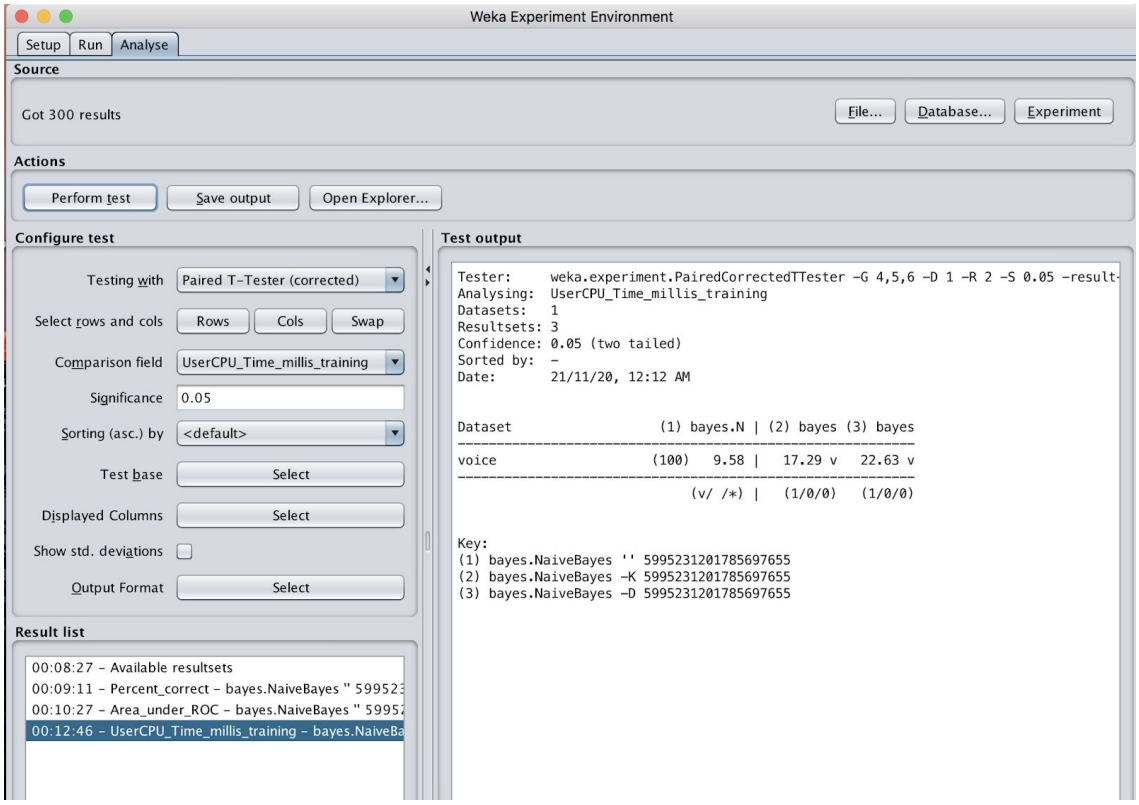
8. In the “Actions” section, click on “Perform Test” button to start the test. Once the test is successfully completed, a detailed comparison report will be available in “Test Output” window as shown below:
   The below report is for “Comparison filed” - Percent_correct, or in other words accuracy.



9. Now, from the drop down menu in “Comparison Field” in “Configure Test “ section, select “Area_under_ROC” and click on “Perform test” button to generate below comparison results between all the 3 models for Area under ROC curve.



10. Now, from the drop down menu in “Comparison Field” in “Configure Test “ section, select “UserCPU_Time_millis_training” and click on “Perform test” button to generate below comparison result between all the 3 models for Time required for training.



Summary of comparison for different parameters :-

Naive Bayes Model	useKernelEstimator	useSupervised Discretization	Accuracy (%)	Percentage Increase w.r.t base	Area under ROC	Training Time (ms)	Percentage Increase w.r.t base
1	False	False	89.10	-	0.95	9.58	-
2	True	False	93.09	4.47%	0.98	17.29	80.1%
3	False	True	91.65	2.86%	0.98	22.63	136.22 %

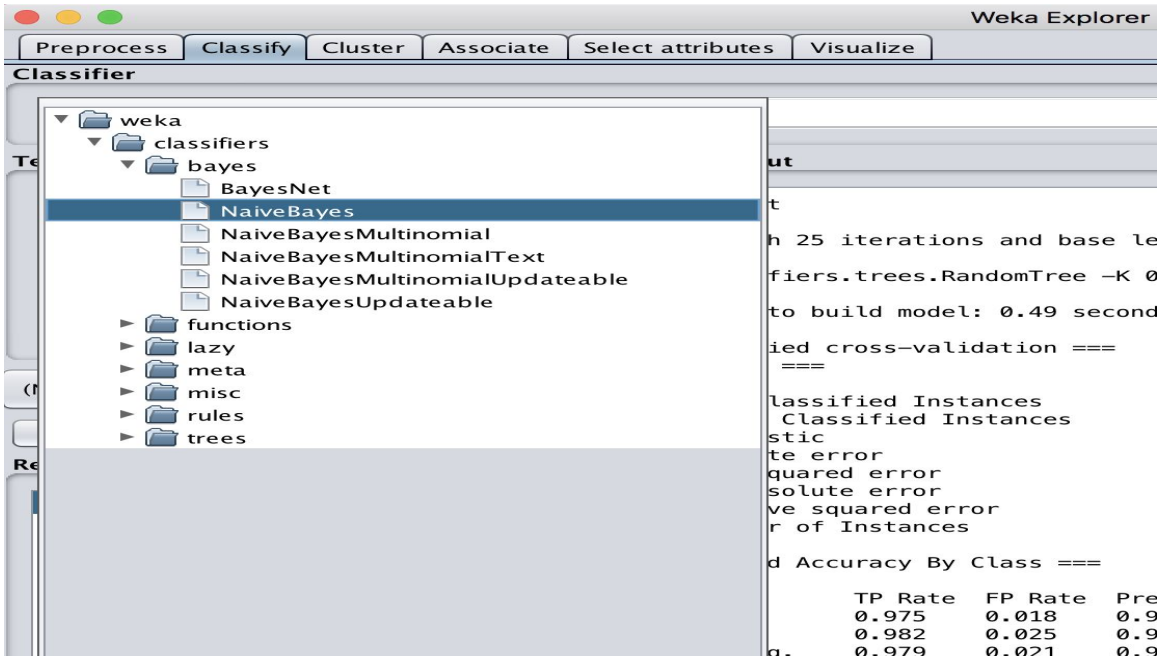
As shown above, the highest accuracy is achieved with Model 2, where kernel estimator is True and the training time is not increased by a great amount for that. So the most optimum model according to me which optimizes accuracy with fair time is Model 2 with Kernel Estimator as true and SuperVised Discretization False.

This model will be used in chapter 9 to compare against Random Forest algo.

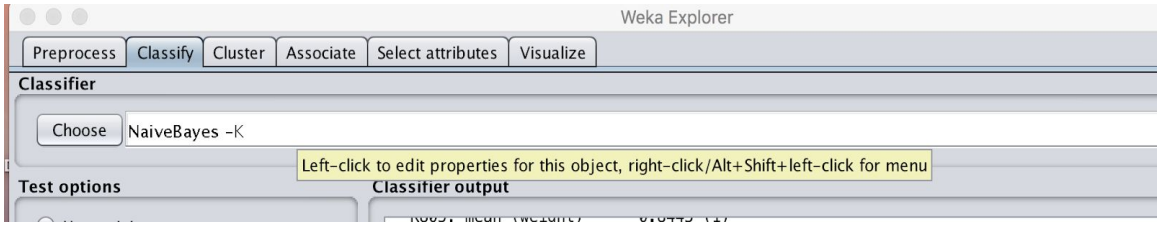
Generating results and ROC curves for this optimized Naive Bayes Model (useKerNelEstimator = True) :-

1. Add dataset file: voice.arff to Explorer in Weka, in Preprocess tab as shown by steps 1-3 in section 7.1.

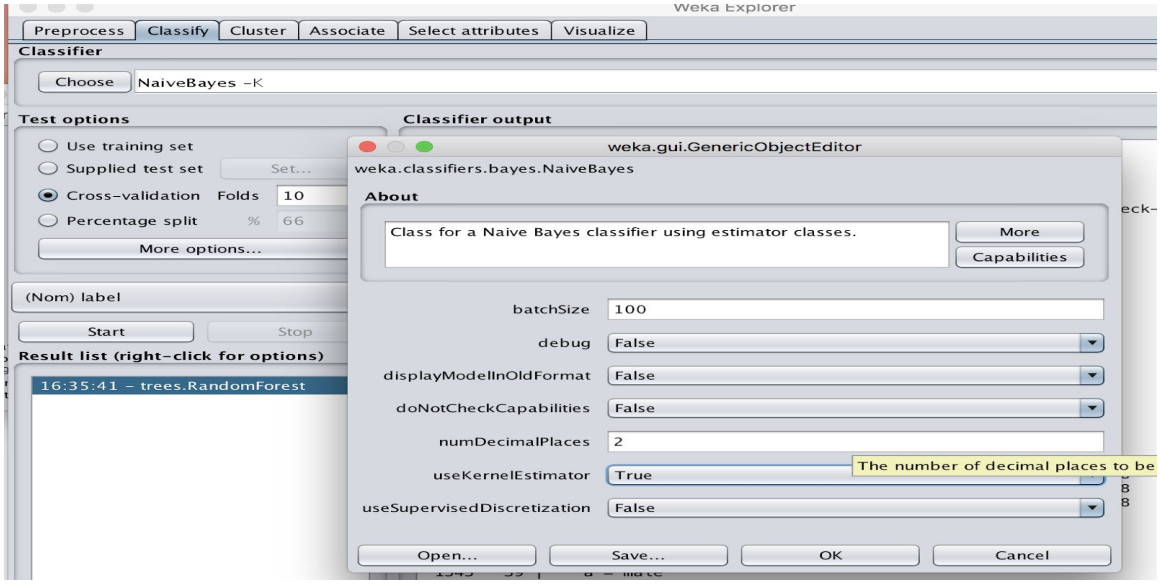
2. In “Classify” tab, click on the Choose and Select Naive Bayes algorithm.



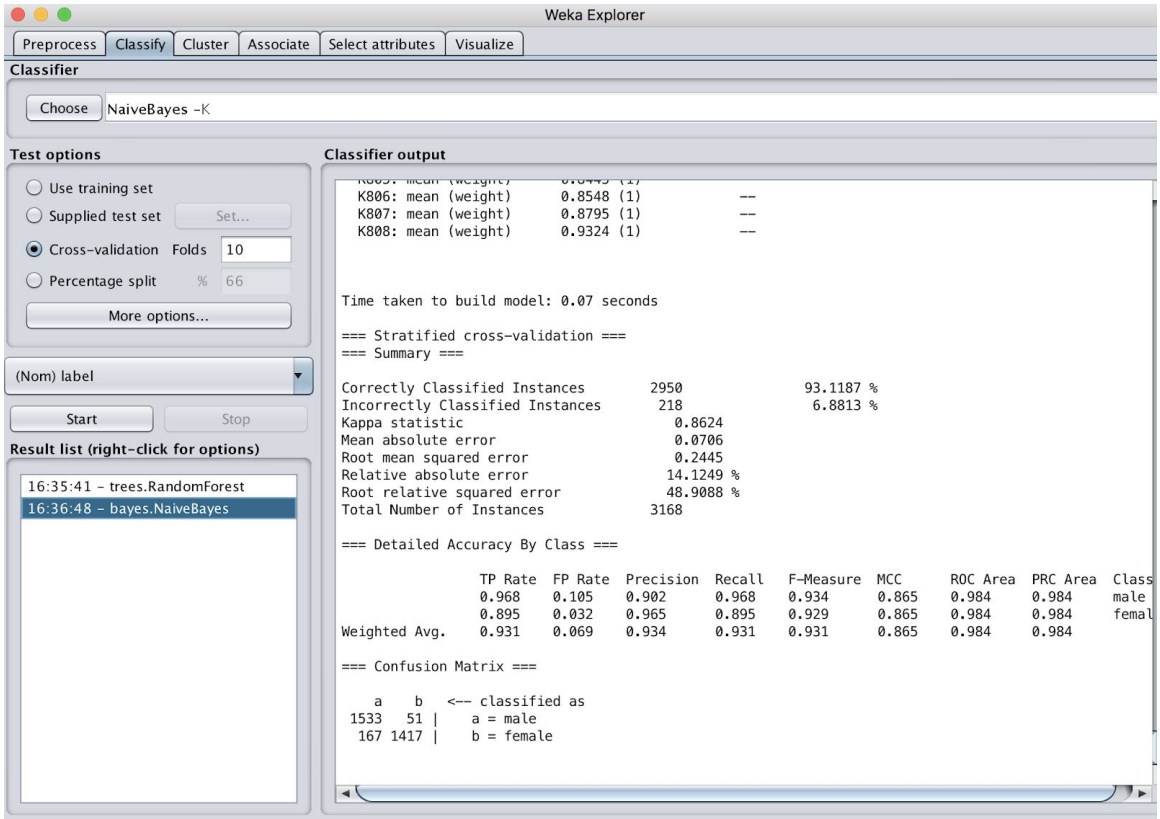
3. In the small window next to “Choose” , click on “Naive Bayes -K” line



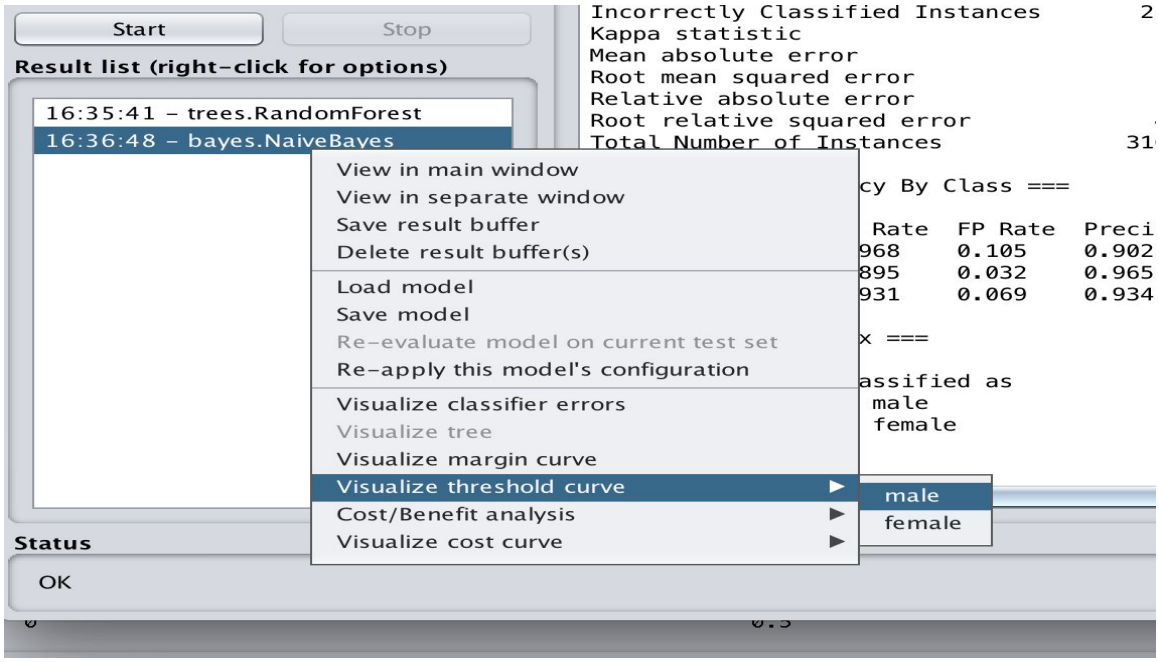
4. It will open parameter selection window for this algo. Change the value of useKernelEstimator to True and Click on OK.

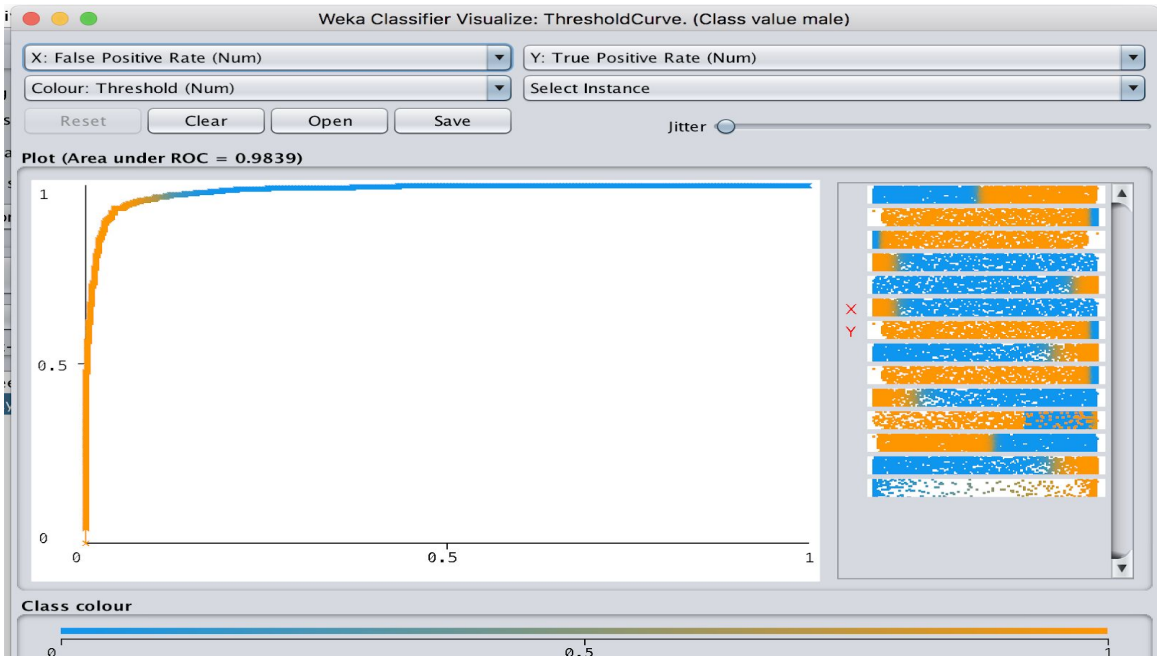


5. Click on Start button and result will be available in “Classifier output” window as below:



7. Right click on Naive Bayes result, in Results List, and in “Visualize threshold curve” click on male and ROC curve will be generated as below.





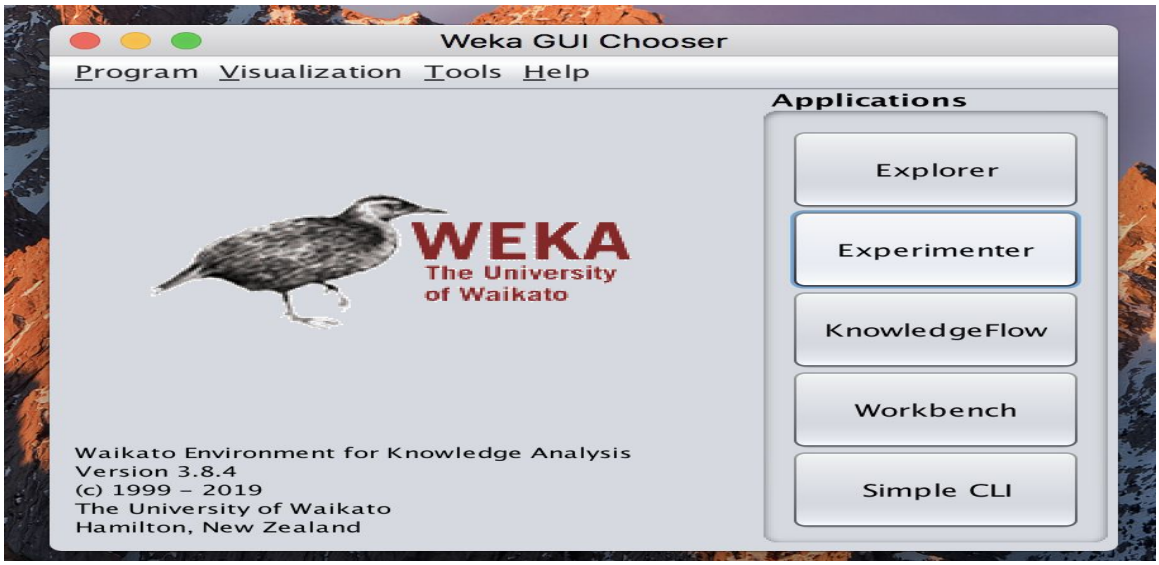
Result Summary for the optimized model where parameter useKernelEstimator = True:-

Accuracy:-	93.11 %
Area under ROC:-	0.984
Precision :-	0.902
Recall:-	0.968
F1:-	0.934
Time Taken to build the Model:-	0.07 sec

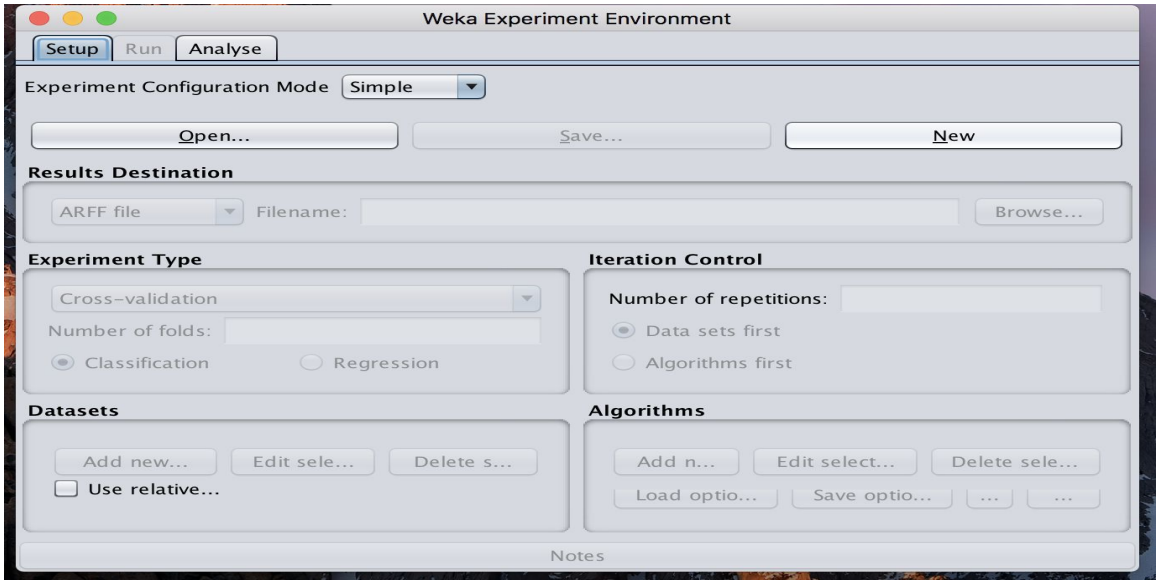
Chapter 9 : Comparison of Random Forest vs Naive Bayes

This Chapter gives step by step guidance on how to run two algorithms : Random Forest and Naive Bayes on the same dataset and generate results for the comparison and analysis of the two algorithms.

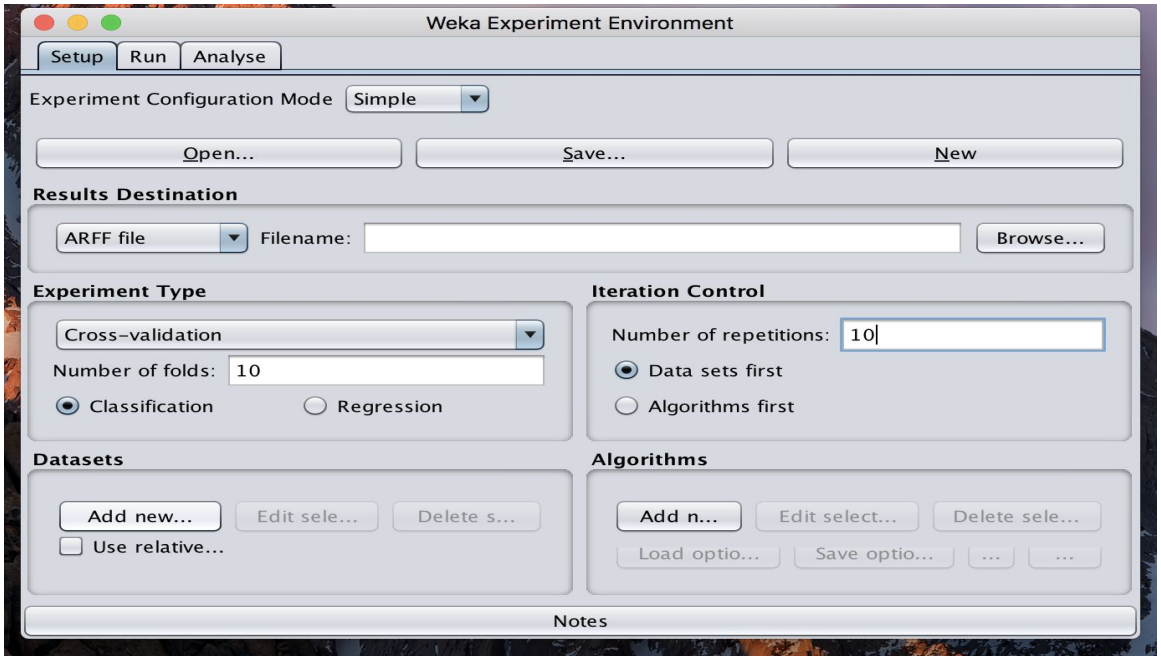
1. Click on “Experimenter” in the Weka Opening Page.



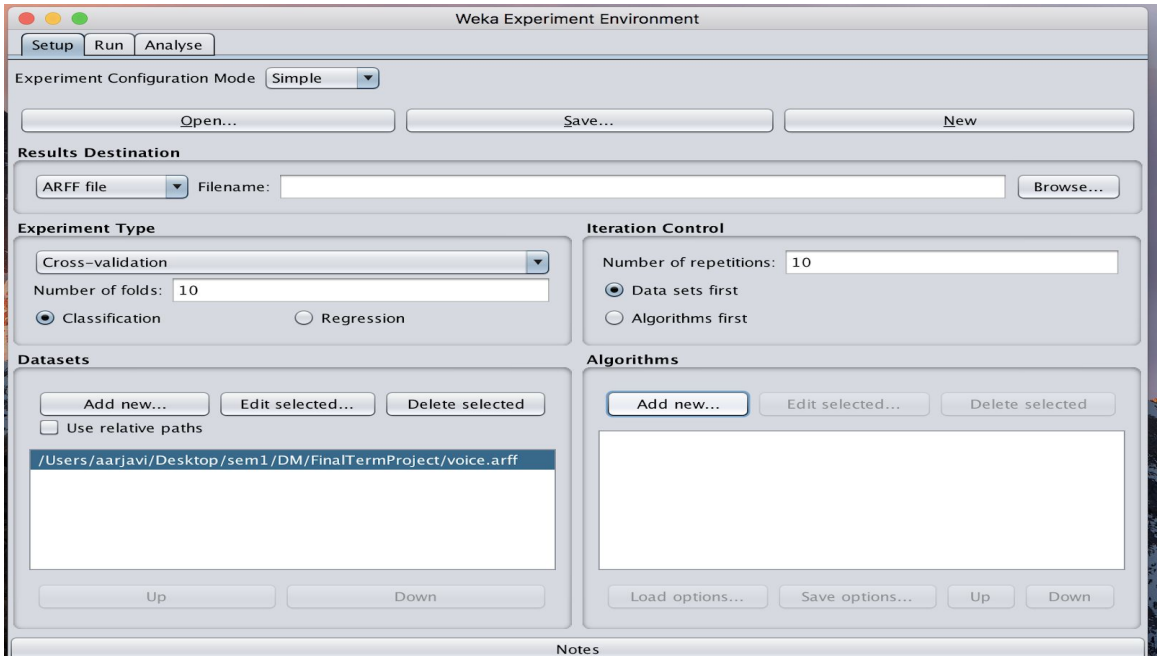
2. In the “Setup” Tab, click on “New” Button on the right most side of the screen.



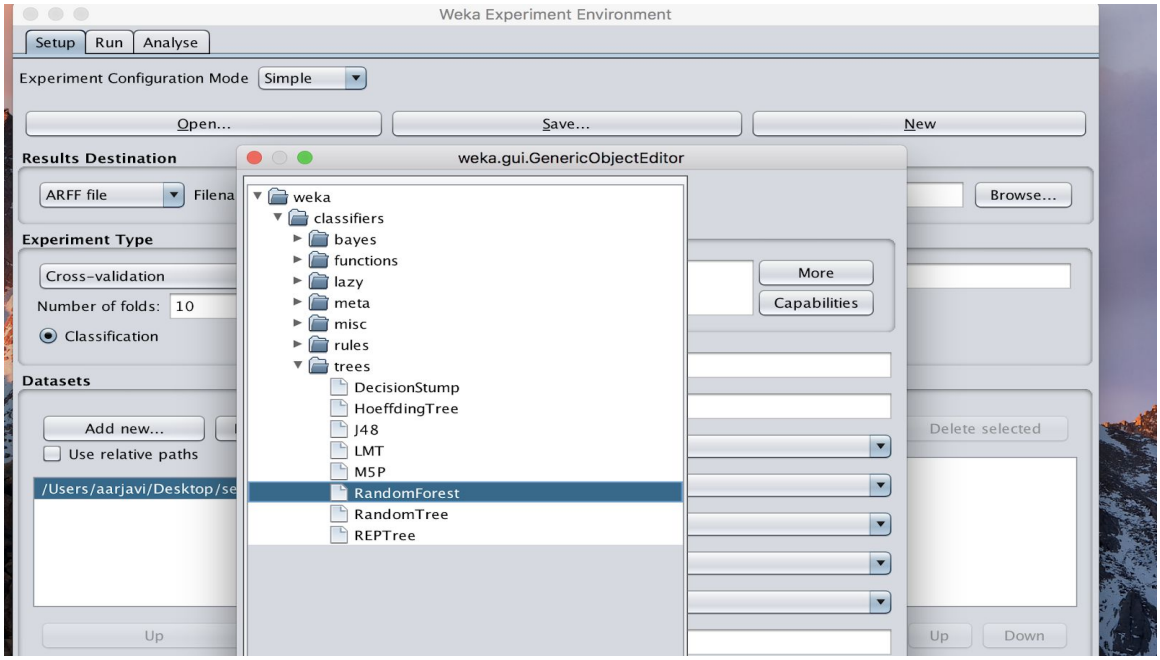
3. Set “Number of folds” to 10 in “Experiment Type” section and set “Number of repetitions” to 10 in “Iteration Control” window.

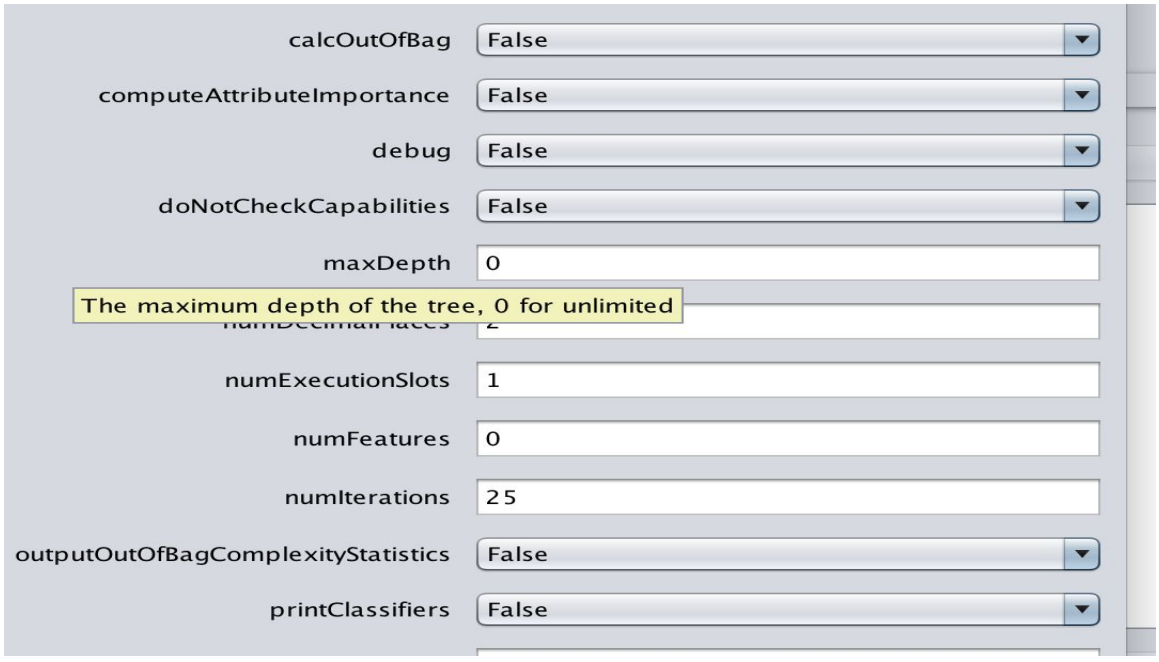


4. In the “Datasets” section, click on “Add new..” button and browse through the dataset file :- voice.arff as shown below.

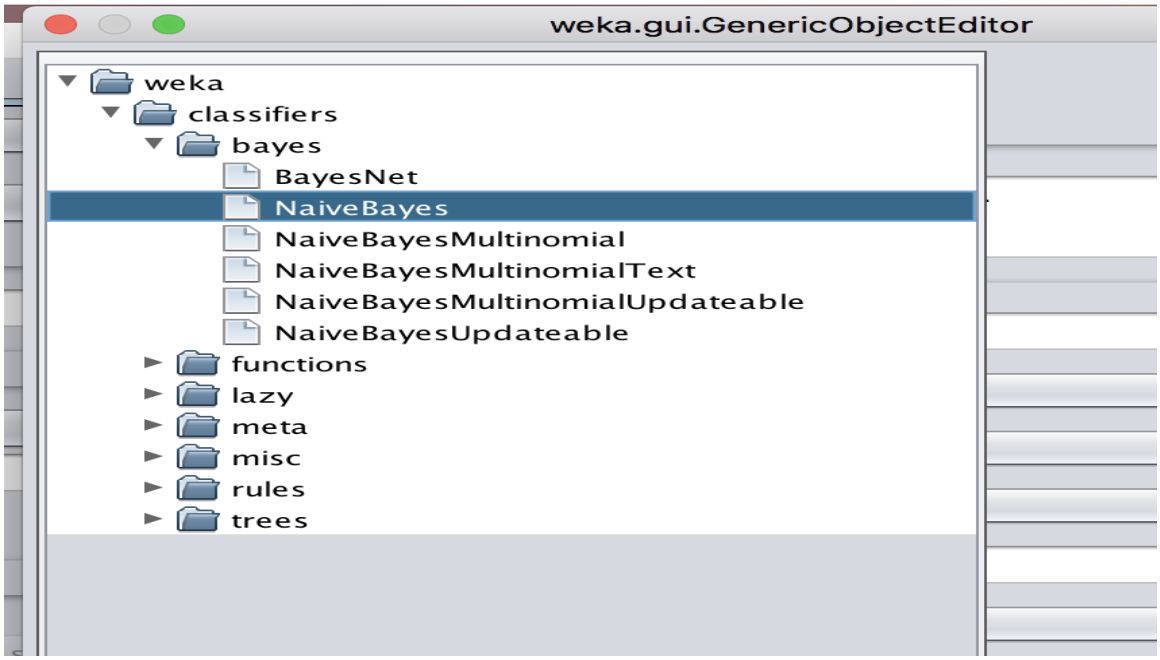


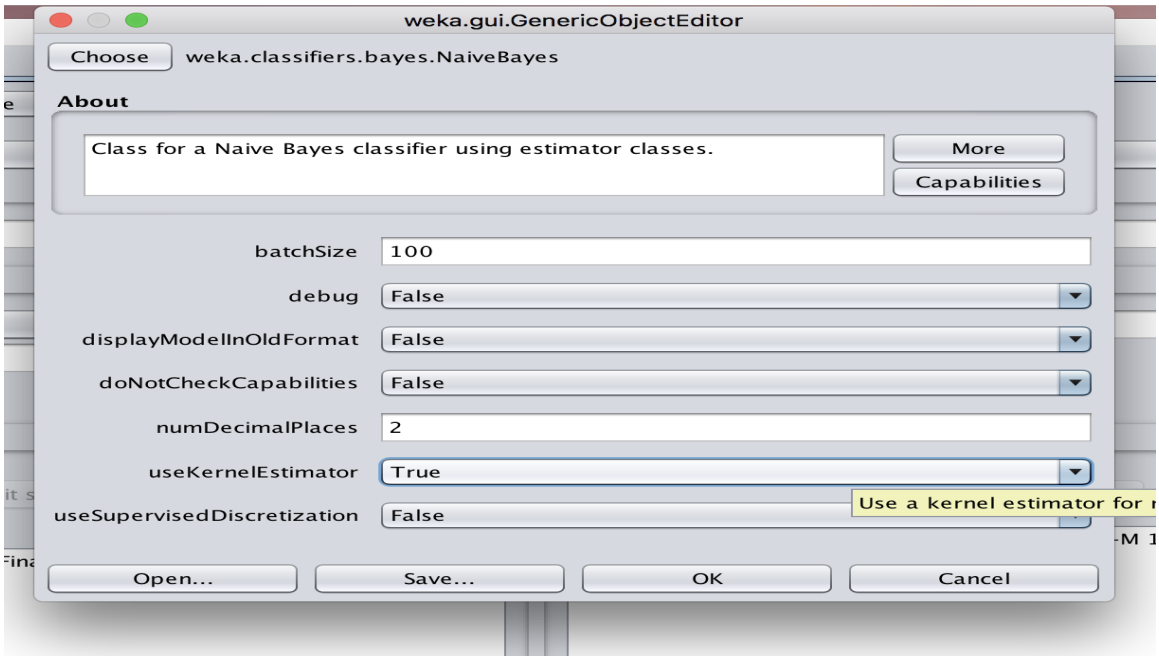
5. In “Algorithm” section, click on “Add new..” button and select “RandomForest” algorithm and select “numIterations” = 25 and click on OK.



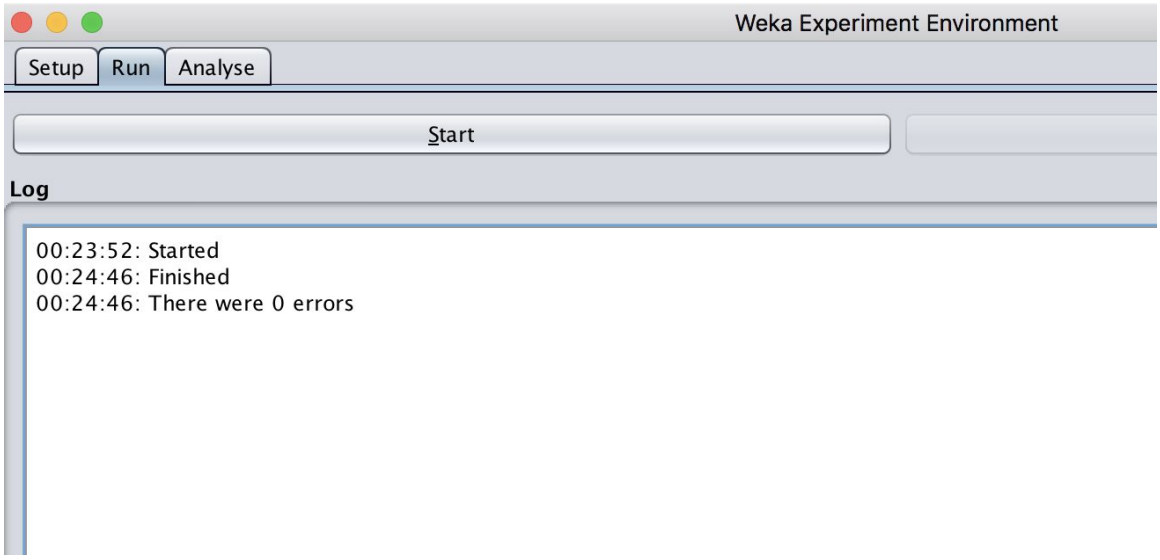


6. Again, click on “Add new” button and now select “Naive Bayes” Algorithm. Set “useKernelEstimator” as True and click on OK.





7. In the “Run” tab, click on “Start” button, and wait till the process is completed. If the algorithms ran successfully on the dataset, a message with 0 errors will be printed as shown below.



8. In the “Analyse” tab, in the “Source” section, click on “Experiment” button.

9. In the “Actions” section, click on “Perform Test” button to start the test. Once the test is successfully completed, a detailed comparison report will be available in “Test Output” window as shown below:

   The below report is for “Comparison filed” - Percent_correct, or in other words accuracy.

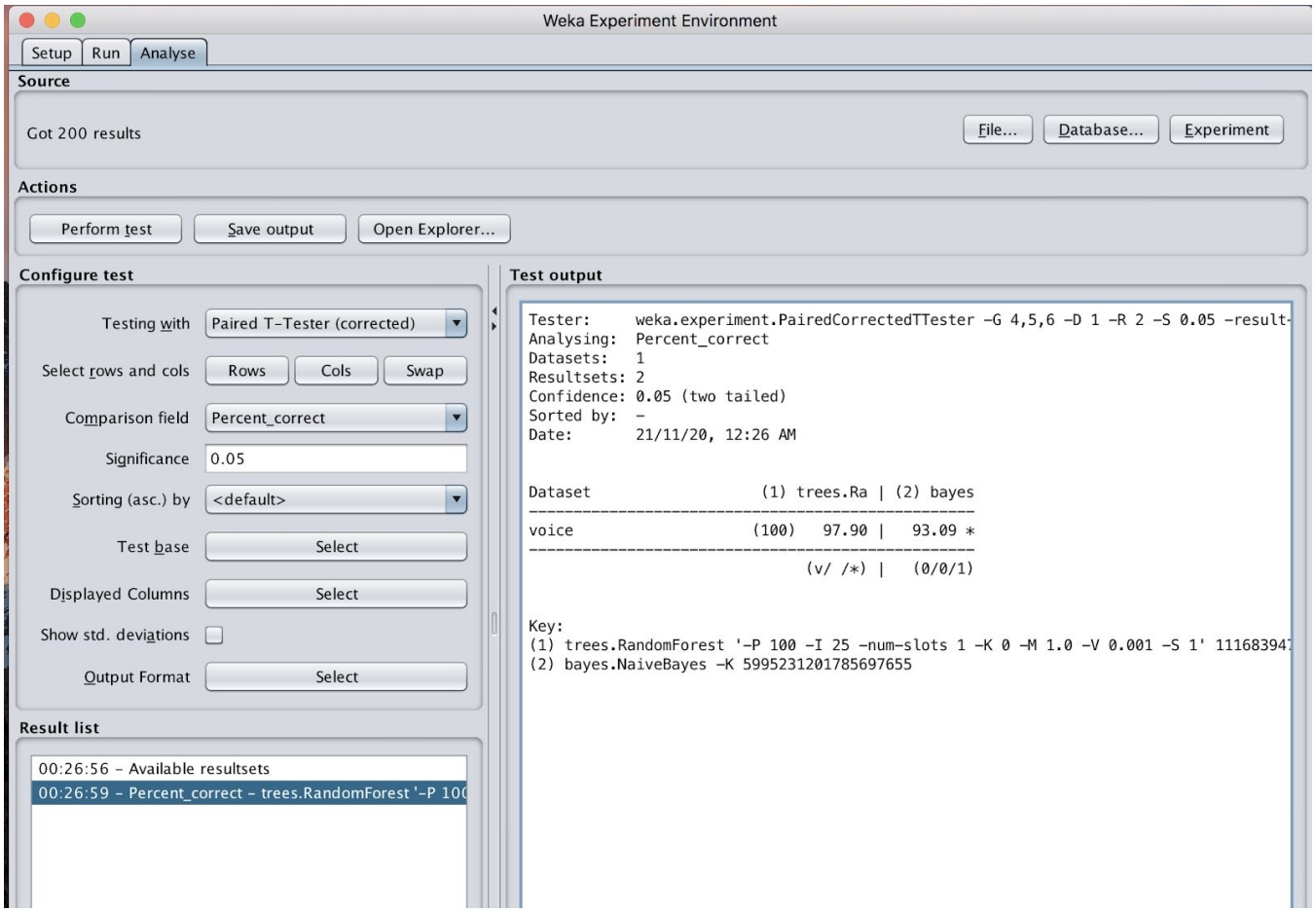
Test Output:-

Tester: weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -result-matrix "weka.experiment.ResultMatrixPlainText -mean-prec 2 -stddev-prec 2 -col-name-width 0 -row-name-width 25 -mean-width 0 -stddev-width 0 -sig-width 0 -count-width 5 -print-col-names -print-row-names -enum-col-names"
Analysing: Percent_correct
Datasets: 1
Resultsets: 2
Confidence: 0.05 (two tailed)
Sorted by: -
Date: 21/11/20, 12:26 AM

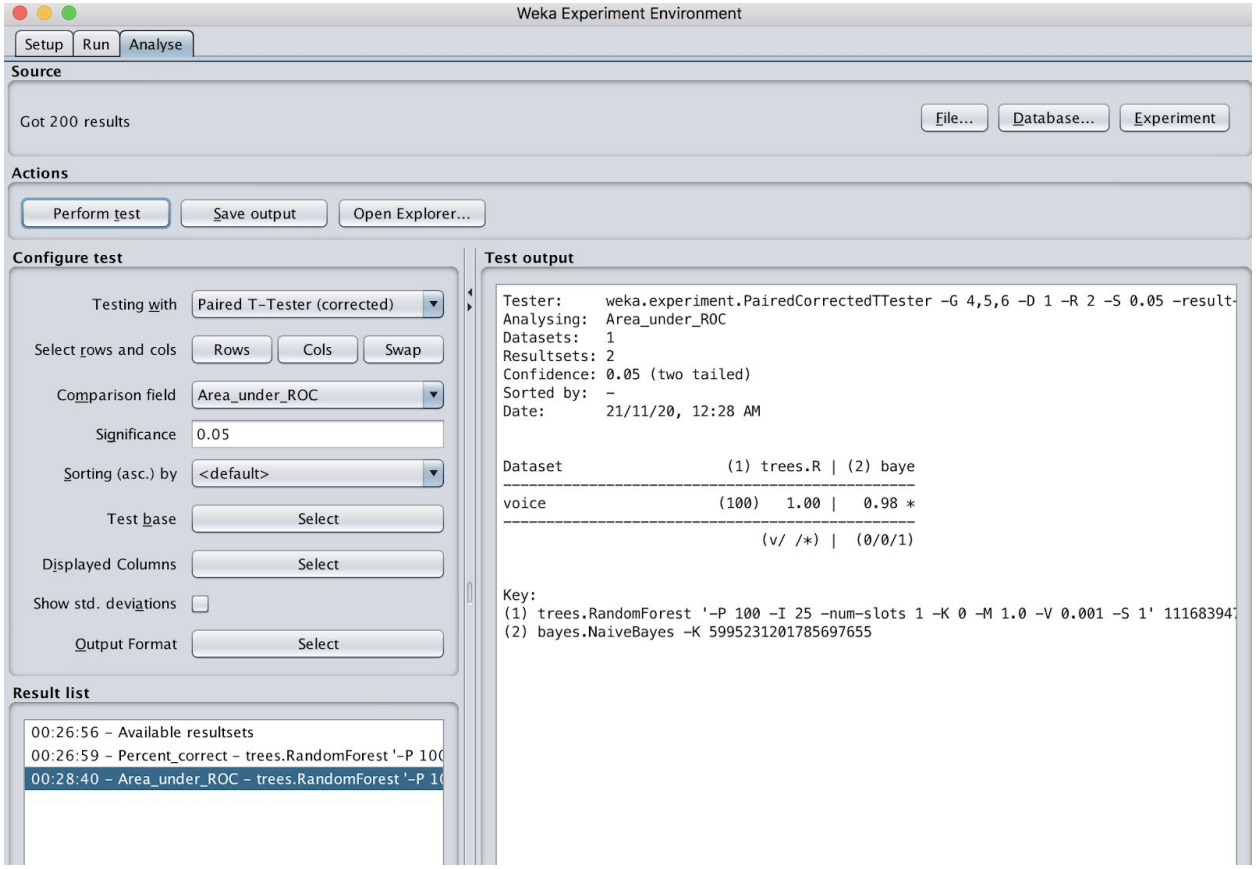
Dataset (1) trees.Ra | (2) bayes
---
voice (100) 97.90 | 93.09 *

(v/ /*) | (0/0/1)

Key:
(1) trees.RandomForest '-P 100 -I 25 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1' 1116839470751428698
(2) bayes.NaiveBayes -K 5995231201785697655



10. Now, from the drop down menu in “Comparison Field” in “Configure Test “ section, select “Area_under_ROC” and click on “Perform test” button to generate below comparison results between the 2 algorithms for Area under ROC curve.



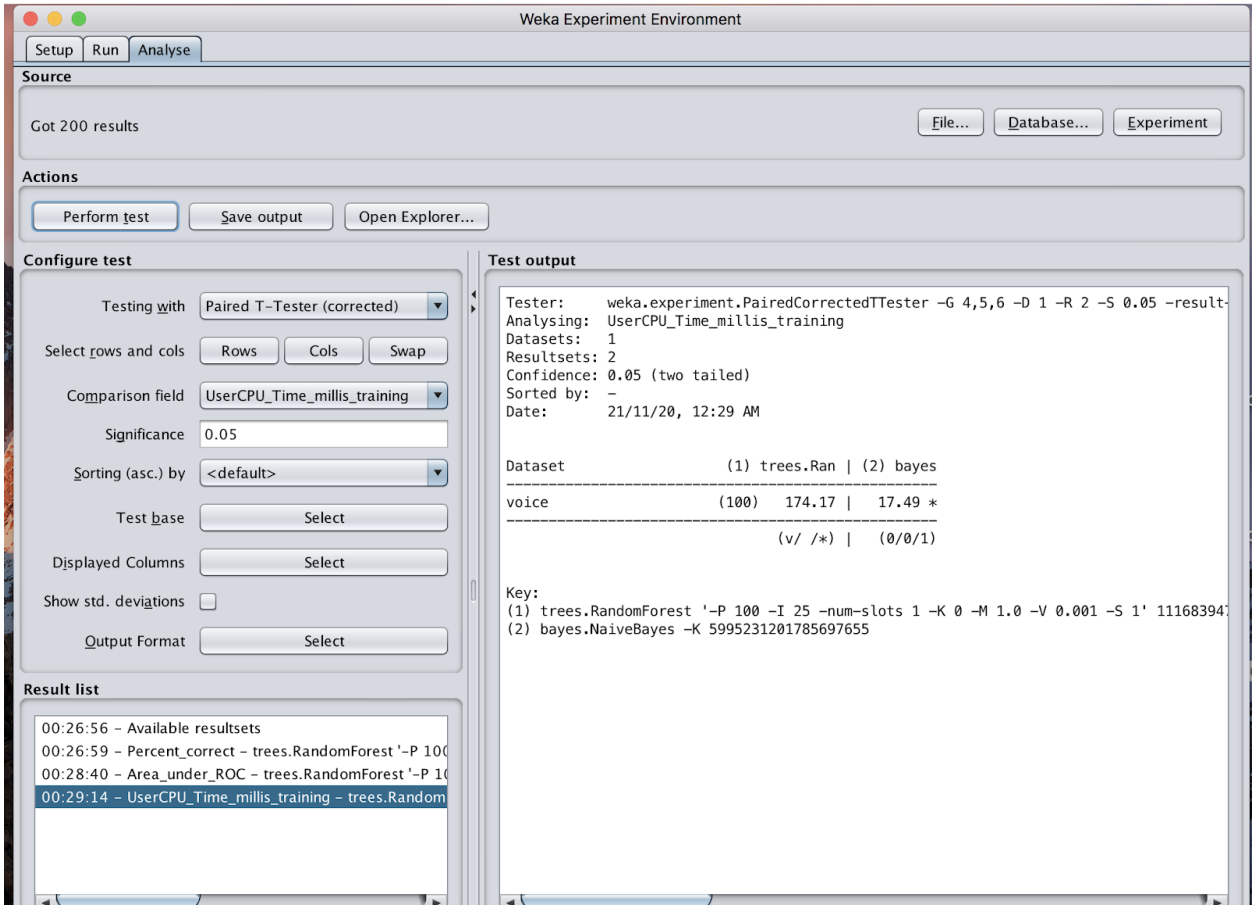
Tester Output:-

Tester: weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -result-matrix "weka.experiment.ResultMatrixPlainText -mean-prec 2 -stddev-prec 2 -col-name-width 0 -row-name-width 25 -mean-width 2 -stddev-width 2 -sig-width 1 -count-width 5 -print-col-names -print-row-names -enum-col-names"
Analysing: Area_under_ROC
Datasets: 1
Resultsets: 2
Confidence: 0.05 (two tailed)
Sorted by: -
Date: 21/11/20, 12:28 AM

Dataset	(1) trees.R	(2) baye
voice	(100) 1.00	0.98 *
	(v/ /*)	(0/0/1)

Key:
(1) trees.RandomForest '-P 100 -I 25 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1' 1116839470751428698
(2) bayes.NaiveBayes -K 5995231201785697655

11. Now, from the drop down menu in “Comparison Field” in “Configure Test “ section, select “UserCPU_Time_millis_training” and click on “Perform test” button to generate below comparison results between the 2 algorithms for training time.



Tester Output:-

Tester: weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -result-matrix "weka.experiment.ResultMatrixPlainText -mean-prec 2 -stddev-prec 2 -col-name-width 0 -row-name-width 25 -mean-width 2 -stddev-width 2 -sig-width 1 -count-width 5 -print-col-names -print-row-names -enum-col-names"
Analysing: UserCPU_Time_millis_training
Datasets: 1
Resultsets: 2
Confidence: 0.05 (two tailed)
Sorted by: -
Date: 21/11/20, 12:29 AM

Dataset	(1) trees.Ran | (2) bayes
voice	(100) 174.17 | 17.49 *
	(v/ /*) | (0/0/1)

Key:
(1) trees.RandomForest '-P 100 -I 25 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1' 1116839470751428698
(2) bayes.NaiveBayes -K 5995231201785697655

Chapter 10 : Conclusion

Below table gives the complete comparison of results between the two algorithms:-

Parameter	Random Forest	Naive Bayes
Accuracy	97.9 %	93.09 %
Area Under ROC Curve	1	.98
UserCPU_Time_milles_training	174.17 ms	17.49 ms
Precision	0.982	0.902
Recall	0.975	0.968
F1	0.979	0.934
Time taken to build the model	0.49 sec	0.07 sec

As shown by the above comparison table,

- The accuracy achieved by the Random Forest Algorithm is much better than Naive Bayes Algorithm, while
- The time taken to build the Random Forest model and its training/testing time is very much higher than that of Naive Bayes algorithm.

Chapter 11 : Appendix

11.1 Source Code for RandomForests

Link for the source code :
https://github.com/Waikato/weka-3.8/blob/master/weka/src/main/java/weka/classifiers/trees/RandomForest.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 3 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, see <http://www.gnu.org/licenses/>.
*/

/*
*    RandomForest.java
*    Copyright (C) 2001-2012 University of Waikato, Hamilton, New Zealand
*
*/

package weka.classifiers.trees;

import weka.classifiers.AbstractClassifier;
import weka.classifiers.Classifier;
import weka.classifiers.meta.Bagging;
import weka.core.Capabilities;
import weka.core.Option;
import weka.core.OptionHandler;
import weka.core.RevisionUtils;
import weka.core.TechnicalInformation;
import weka.core.TechnicalInformation.Field;
import weka.core.TechnicalInformation.Type;
import weka.core.Utils;
import weka.core.WekaException;
import weka.gui.ProgrammaticProperty;

import java.util.Collections;
```

```
import java.util.Enumeration;
import java.util.List;
import java.util.Vector;

/**
* <!-- globalinfo-start --> Class for constructing a forest of random
trees.<br>
* <br>
* For more information see: <br>
* <br>
* Leo Breiman (2001). Random Forests. Machine Learning. 45(1):5-32. <br>
* <br>
* <!-- globalinfo-end -->
*
* <!-- technical-bibtex-start --> BibTeX:
*
* <pre>
* &#64;article{Breiman2001,
*    author = {Leo Breiman},
*    journal = {Machine Learning},
*    number = {1},
*    pages = {5-32},
*    title = {Random Forests},
*    volume = {45},
*    year = {2001}
* }
* </pre>
*
* <br>
* <br>
* <!-- technical-bibtex-end -->
*
* <!-- options-start --> Valid options are:
* <p>
*
* <pre>
* -P
*  Size of each bag, as a percentage of the
*  training set size. (default 100)
* </pre>
*
* <pre>
* -O
*  Calculate the out of bag error.
* </pre>
*
* <pre>
* -store-out-of-bag-predictions
*  Whether to store out of bag predictions in internal evaluation object.
```

```
* </pre>
*
* <pre>
* -output-out-of-bag-complexity-statistics
*  Whether to output complexity-based statistics when out-of-bag evaluation is
performed.
* </pre>
*
* <pre>
* -print
*  Print the individual classifiers in the output
* </pre>
*
* <pre>
* -attribute-importance
*  Compute and output attribute importance (mean impurity decrease method)
* </pre>
*
* <pre>
* -I &lt;num&gt;
*  Number of iterations.
*  (current value 100)
* </pre>
*
* <pre>
* -num-slots &lt;num&gt;
*  Number of execution slots.
*  (default 1 - i.e. no parallelism)
*  (use 0 to auto-detect number of cores)
* </pre>
*
* <pre>
* -K &lt;number of attributes&gt;
*  Number of attributes to randomly investigate. (default 0)
*  (&lt;1 = int(log_2(#predictors)+1)).
* </pre>
*
* <pre>
* -M &lt;minimum number of instances&gt;
*  Set minimum number of instances per leaf.
*  (default 1)
* </pre>
*
* <pre>
* -V &lt;minimum variance for split&gt;
*  Set minimum numeric class variance proportion
*  of train variance for split (default 1e-3).
* </pre>
*
```

```
* <pre>
* -S &lt;num&gt;
*  Seed for random number generator.
*  (default 1)
* </pre>
*
* <pre>
* -depth &lt;num&gt;
*  The maximum depth of the tree, 0 for unlimited.
*  (default 0)
* </pre>
*
* <pre>
* -N &lt;num&gt;
*  Number of folds for backfitting (default 0, no backfitting).
* </pre>
*
* <pre>
* -U
*  Allow unclassified instances.
* </pre>
*
* <pre>
* -B
*  Break ties randomly when several attributes look equally good.
* </pre>
*
* <pre>
* -output-debug-info
*  If set, classifier is run in debug mode and
*  may output additional info to the console
* </pre>
*
* <pre>
* -do-not-check-capabilities
*  If set, classifier capabilities are not checked before classifier is built
*  (use with caution).
* </pre>
*
* <pre>
* -num-decimal-places
*  The number of decimal places for the output of numbers in the model
(default 2).
* </pre>
*
* <pre>
* -batch-size
*  The desired batch size for batch prediction  (default 100).
* </pre>
```

```
 *
 * <!-- options-end -->
 *
 * @author Richard Kirkby (rkirkby@cs.waikato.ac.nz)
 * @version $Revision$
 */
public class RandomForest extends Bagging {

 /** for serialization */
 static final long serialVersionUID = 1116839470751428698L;

 /** True to compute attribute importance */
 protected boolean m_computeAttributeImportance;

 /**
  * The default number of iterations to perform.
  */
 @Override
 protected int defaultNumberOfIterations() {
   return 100;
 }

 /**
  * Constructor that sets base classifier for bagging to RandomTre and default
  * number of iterations to 100.
  */
 public RandomForest() {

   RandomTree rTree = new RandomTree();
   rTree.setDoNotCheckCapabilities(true);
   super.setClassifier(rTree);
   super.setRepresentCopiesUsingWeights(true);
   setNumIterations(defaultNumberOfIterations());
 }

 /**
  * Returns default capabilities of the base classifier.
  *
  * @return the capabilities of the base classifier
  */
 public Capabilities getCapabilities() {

   // Cannot use the main RandomTree object because capabilities checking has
   // been turned off
   // for that object.
   return (new RandomTree()).getCapabilities();
 }

 /**
```

```
 * String describing default classifier.
 *
 * @return the default classifier classname
 */
@Override
protected String defaultClassifierString() {

  return "weka.classifiers.trees.RandomTree";
}

/**
 * String describing default classifier options.
 *
 * @return the default classifier options
 */
@Override
protected String[] defaultClassifierOptions() {

  String[] args = { "-do-not-check-capabilities" };
  return args;
}

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

  return "Class for constructing a forest of random trees.\n\n"
    + "For more information see: \n\n" + getTechnicalInformation().toString();
}

/**
 * Returns an instance of a TechnicalInformation object, containing detailed
 * information about the technical background of this class, e.g., paper
 * reference or book this class is based on.
 *
 * @return the technical information about this class
 */
@Override
public TechnicalInformation getTechnicalInformation() {
  TechnicalInformation result;

  result = new TechnicalInformation(Type.ARTICLE);
  result.setValue(Field.AUTHOR, "Leo Breiman");
  result.setValue(Field.YEAR, "2001");
  result.setValue(Field.TITLE, "Random Forests");
```

```
    result.setValue(Field.JOURNAL, "Machine Learning");
    result.setValue(Field.VOLUME, "45");
    result.setValue(Field.NUMBER, "1");
    result.setValue(Field.PAGES, "5-32");

    return result;
  }

  /**
   * This method only accepts RandomTree arguments.
   *
   * @param newClassifier the RandomTree to use.
   * @exception if argument is not a RandomTree
   */
  @Override
  @ProgrammaticProperty
  public void setClassifier(Classifier newClassifier) {
    if (!(newClassifier instanceof RandomTree)) {
      throw new IllegalArgumentException(
        "RandomForest: Argument of setClassifier() must be a RandomTree.");
    }
    super.setClassifier(newClassifier);
  }

  /**
   * This method only accepts true as its argument
   *
   * @param representUsingWeights must be set to true.
   * @exception if argument is not true
   */
  @Override
  @ProgrammaticProperty
  public void setRepresentCopiesUsingWeights(boolean representUsingWeights) {
    if (!representUsingWeights) {
      throw new IllegalArgumentException(
        "RandomForest: Argument of setRepresentCopiesUsingWeights() must be
true.");
    }
    super.setRepresentCopiesUsingWeights(representUsingWeights);
  }

  /**
   * Returns the tip text for this property
   *
   * @return tip text for this property suitable for displaying in the
   *         explorer/experimenter gui
   */
  public String numFeaturesTipText() {
    return ((RandomTree) getClassifier()).KValueTipText();
```

```
 }

 /**
  * Get the number of features used in random selection.
  *
  * @return Value of numFeatures.
  */
 public int getNumFeatures() {

   return ((RandomTree) getClassifier()).getKValue();
 }

 /**
  * Set the number of features to use in random selection.
  *
  * @param newNumFeatures Value to assign to numFeatures.
  */
 public void setNumFeatures(int newNumFeatures) {

   ((RandomTree) getClassifier()).setKValue(newNumFeatures);
 }

 /**
  * Returns the tip text for this property
  *
  * @return tip text for this property suitable for displaying in the
  *         explorer/experimenter gui
  */
 public String computeAttributeImportanceTipText() {
   return "Compute attribute importance via mean impurity decrease";
 }

 /**
  * Set whether to compute and output attribute importance scores
  *
  * @param computeAttributeImportance true to compute attribute importance
  *          scores
  */
 public void setComputeAttributeImportance(boolean computeAttributeImportance)
{
   m_computeAttributeImportance = computeAttributeImportance;

((RandomTree)m_Classifier).setComputeImpurityDecreases(computeAttributeImportan
ce);
 }

 /**
  * Get whether to compute and output attribute importance scores
  *
```

```
   * @return true if computing attribute importance scores
   */
  public boolean getComputeAttributeImportance() {
    return m_computeAttributeImportance;
  }

  /**
   * Returns the tip text for this property
   *
   * @return tip text for this property suitable for displaying in the
   *         explorer/experimenter gui
   */
  public String maxDepthTipText() {
    return ((RandomTree) getClassifier()).maxDepthTipText();
  }

  /**
   * Get the maximum depth of trh tree, 0 for unlimited.
   *
   * @return the maximum depth.
   */
  public int getMaxDepth() {
    return ((RandomTree) getClassifier()).getMaxDepth();
  }

  /**
   * Set the maximum depth of the tree, 0 for unlimited.
   *
   * @param value the maximum depth.
   */
  public void setMaxDepth(int value) {
    ((RandomTree) getClassifier()).setMaxDepth(value);
  }

  /**
   * Returns the tip text for this property
   *
   * @return tip text for this property suitable for displaying in the
   *         explorer/experimenter gui
   */
  public String breakTiesRandomlyTipText() {
    return ((RandomTree) getClassifier()).breakTiesRandomlyTipText();
  }

  /**
   * Get whether to break ties randomly.
   *
   * @return true if ties are to be broken randomly.
   */
```

```
public boolean getBreakTiesRandomly() {

  return ((RandomTree) getClassifier()).getBreakTiesRandomly();
}

/**
 * Set whether to break ties randomly.
 *
 * @param newBreakTiesRandomly true if ties are to be broken randomly
 */
public void setBreakTiesRandomly(boolean newBreakTiesRandomly) {

  ((RandomTree) getClassifier()).setBreakTiesRandomly(newBreakTiesRandomly);
}

/**
 * Set debugging mode.
 *
 * @param debug true if debug output should be printed
 */
public void setDebug(boolean debug) {

  super.setDebug(debug);
  ((RandomTree) getClassifier()).setDebug(debug);
}

/**
 * Set the number of decimal places.
 */
public void setNumDecimalPlaces(int num) {

  super.setNumDecimalPlaces(num);
  ((RandomTree) getClassifier()).setNumDecimalPlaces(num);
}

/**
 * Set the preferred batch size for batch prediction.
 *
 * @param size the batch size to use
 */
@Override
public void setBatchSize(String size) {

  super.setBatchSize(size);
  ((RandomTree) getClassifier()).setBatchSize(size);
}

/**
 * Sets the seed for the random number generator.
```

```
   *
   * @param s the seed to be used
   */
  public void setSeed(int s) {

    super.setSeed(s);
    ((RandomTree) getClassifier()).setSeed(s);
  }

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

    if (m_Classifiers == null) {
      return "RandomForest: No model built yet.";
    }
    StringBuilder buffer = new StringBuilder("RandomForest\n\n");
    buffer.append(super.toString());

    if (getComputeAttributeImportance()) {
      try {
        double[] nodeCounts = new double[m_data.numAttributes()];
        double[] impurityScores =
          computeAverageImpurityDecreasePerAttribute(nodeCounts);
        int[] sortedIndices = Utils.sort(impurityScores);
        buffer
          .append("\n\nAttribute importance based on average impurity decrease "
            + "(and number of nodes using that attribute)\n\n");
        for (int i = sortedIndices.length - 1; i >= 0; i--) {
          int index = sortedIndices[i];
          if (index != m_data.classIndex()) {
            buffer
              .append(
                Utils.doubleToString(impurityScores[index], 10,
                  getNumDecimalPlaces())).append(" (")
              .append(Utils.doubleToString(nodeCounts[index], 6, 0))
              .append(")  ").append(m_data.attribute(index).name())
              .append("\n");
          }
        }
      } catch (WekaException ex) {
        // ignore
      }
    }
```

```
    return buffer.toString();
  }

  /**
   * Computes the average impurity decrease per attribute over the trees
   *
   * @param nodeCounts an optional array that, if non-null, will hold the count
   *          of the number of nodes at which each attribute was used for
   *          splitting
   * @return the average impurity decrease per attribute over the trees
   */
  public double[] computeAverageImpurityDecreasePerAttribute(
    double[] nodeCounts) throws WekaException {

    if (m_Classifiers == null) {
      throw new WekaException("Classifier has not been built yet!");
    }

    if (!getComputeAttributeImportance()) {
      throw new WekaException("Stats for attribute importance have not "
        + "been collected!");
    }

    double[] impurityDecreases = new double[m_data.numAttributes()];
    if (nodeCounts == null) {
      nodeCounts = new double[m_data.numAttributes()];
    }
    for (Classifier c : m_Classifiers) {
      double[][] forClassifier = ((RandomTree) c).getImpurityDecreases();
      for (int i = 0; i < m_data.numAttributes(); i++) {
        impurityDecreases[i] += forClassifier[i][0];
        nodeCounts[i] += forClassifier[i][1];
      }
    }
    for (int i = 0; i < m_data.numAttributes(); i++) {
      if (nodeCounts[i] > 0) {
        impurityDecreases[i] /= nodeCounts[i];
      }
    }

    return impurityDecreases;
  }

  /**
   * Returns an enumeration describing the available options.
   *
   * @return an enumeration of all the available options
   */
  @Override
```

```
 public Enumeration<Option> listOptions() {

   Vector<Option> newVector = new Vector<Option>();

   newVector.addElement(new Option(
     "\tSize of each bag, as a percentage of the\n"
       + "\ttraining set size. (default 100)", "P", 1, "-P"));

   newVector.addElement(new Option("\tCalculate the out of bag error.", "O",
     0, "-O"));

   newVector
     .addElement(new Option(
       "\tWhether to store out of bag predictions in internal evaluation
object.",
       "store-out-of-bag-predictions", 0, "-store-out-of-bag-predictions"));

   newVector
     .addElement(new Option(
       "\tWhether to output complexity-based statistics when out-of-bag
evaluation is performed.",
       "output-out-of-bag-complexity-statistics", 0,
       "-output-out-of-bag-complexity-statistics"));

   newVector
     .addElement(new Option(
       "\tPrint the individual classifiers in the output", "print", 0,
       "-print"));

   newVector.addElement(new Option(
     "\tCompute and output attribute importance (mean impurity decrease "
       + "method)", "attribute-importance", 0, "-attribute-importance"));

   newVector.addElement(new Option("\tNumber of iterations.\n"
     + "\t(current value " + getNumIterations() + ")", "I", 1, "-I <num>"));

   newVector.addElement(new Option("\tNumber of execution slots.\n"
     + "\t(default 1 - i.e. no parallelism)\n"
     + "\t(use 0 to auto-detect number of cores)", "num-slots", 1,
     "-num-slots <num>"));

   // Add base classifier options
   List<Option> list =
     Collections.list(((OptionHandler) getClassifier()).listOptions());
   newVector.addAll(list);

   return newVector.elements();
 }
```

```
/**
 * Gets the current settings of the forest.
 *
 * @return an array of strings suitable for passing to setOptions()
 */
@Override
public String[] getOptions() {
  Vector<String> result = new Vector<String>();

  result.add("-P");
  result.add("" + getBagSizePercent());

  if (getCalcOutOfBag()) {
    result.add("-O");
  }

  if (getStoreOutOfBagPredictions()) {
    result.add("-store-out-of-bag-predictions");
  }

  if (getOutputOutOfBagComplexityStatistics()) {
    result.add("-output-out-of-bag-complexity-statistics");
  }

  if (getPrintClassifiers()) {
    result.add("-print");
  }

  if (getComputeAttributeImportance()) {
    result.add("-attribute-importance");
  }

  result.add("-I");
  result.add("" + getNumIterations());

  result.add("-num-slots");
  result.add("" + getNumExecutionSlots());

  if (getDoNotCheckCapabilities()) {
    result.add("-do-not-check-capabilities");
  }

  // Add base classifier options
  Vector<String> classifierOptions = new Vector<String>();
  Collections.addAll(classifierOptions,
    ((OptionHandler) getClassifier()).getOptions());
  Option.deleteFlagString(classifierOptions, "-do-not-check-capabilities");
  result.addAll(classifierOptions);
```

```
    return result.toArray(new String[result.size()]);
  }

  /**
   * Parses a given list of options.
   * <p/>
   *
   * <!-- options-start --> Valid options are:
   * <p>
   *
   * <pre>
   * -P
   *  Size of each bag, as a percentage of the
   *  training set size. (default 100)
   * </pre>
   *
   * <pre>
   * -O
   *  Calculate the out of bag error.
   * </pre>
   *
   * <pre>
   * -store-out-of-bag-predictions
   *  Whether to store out of bag predictions in internal evaluation object.
   * </pre>
   *
   * <pre>
   * -output-out-of-bag-complexity-statistics
   *  Whether to output complexity-based statistics when out-of-bag evaluation
is performed.
   * </pre>
   *
   * <pre>
   * -print
   *  Print the individual classifiers in the output
   * </pre>
   *
   * <pre>
   * -attribute-importance
   *  Compute and output attribute importance (mean impurity decrease method)
   * </pre>
   *
   * <pre>
   * -I &lt;num&gt;
   *  Number of iterations.
   *  (current value 100)
   * </pre>
   *
   * <pre>
```

```
* -num-slots &lt;num&gt;
*  Number of execution slots.
*  (default 1 - i.e. no parallelism)
*  (use 0 to auto-detect number of cores)
* </pre>
*
* <pre>
* -K &lt;number of attributes&gt;
*  Number of attributes to randomly investigate. (default 0)
*  (&lt;1 = int(log_2(#predictors)+1)).
* </pre>
*
* <pre>
* -M &lt;minimum number of instances&gt;
*  Set minimum number of instances per leaf.
*  (default 1)
* </pre>
*
* <pre>
* -V &lt;minimum variance for split&gt;
*  Set minimum numeric class variance proportion
*  of train variance for split (default 1e-3).
* </pre>
*
* <pre>
* -S &lt;num&gt;
*  Seed for random number generator.
*  (default 1)
* </pre>
*
* <pre>
* -depth &lt;num&gt;
*  The maximum depth of the tree, 0 for unlimited.
*  (default 0)
* </pre>
*
* <pre>
* -N &lt;num&gt;
*  Number of folds for backfitting (default 0, no backfitting).
* </pre>
*
* <pre>
* -U
*  Allow unclassified instances.
* </pre>
*
* <pre>
* -B
*  Break ties randomly when several attributes look equally good.
```

```
   * </pre>
   *
   * <pre>
   * -output-debug-info
   *  If set, classifier is run in debug mode and
   *  may output additional info to the console
   * </pre>
   *
   * <pre>
   * -do-not-check-capabilities
   *  If set, classifier capabilities are not checked before classifier is
built
   *  (use with caution).
   * </pre>
   *
   * <pre>
   * -num-decimal-places
   *  The number of decimal places for the output of numbers in the model
(default 2).
   * </pre>
   *
   * <pre>
   * -batch-size
   *  The desired batch size for batch prediction  (default 100).
   * </pre>
   *
   * <!-- options-end -->
   *
   * @param options the list of options as an array of strings
   * @throws Exception if an option is not supported
   */
  @Override
  public void setOptions(String[] options) throws Exception {

    String bagSize = Utils.getOption('P', options);
    if (bagSize.length() != 0) {
      setBagSizePercent(Integer.parseInt(bagSize));
    } else {
      setBagSizePercent(100);
    }

    setCalcOutOfBag(Utils.getFlag('O', options));

    setStoreOutOfBagPredictions(Utils.getFlag("store-out-of-bag-predictions",
      options));

    setOutputOutOfBagComplexityStatistics(Utils.getFlag(
      "output-out-of-bag-complexity-statistics", options));
```

```
    setPrintClassifiers(Utils.getFlag("print", options));

    setComputeAttributeImportance(Utils
      .getFlag("attribute-importance", options));

    String iterations = Utils.getOption('I', options);
    if (iterations.length() != 0) {
      setNumIterations(Integer.parseInt(iterations));
    } else {
      setNumIterations(defaultNumberOfIterations());
    }

    String numSlots = Utils.getOption("num-slots", options);
    if (numSlots.length() != 0) {
      setNumExecutionSlots(Integer.parseInt(numSlots));
    } else {
      setNumExecutionSlots(1);
    }

    RandomTree classifier =
      ((RandomTree) AbstractClassifier.forName(defaultClassifierString(),
        options));
    classifier.setComputeImpurityDecreases(m_computeAttributeImportance);
    setDoNotCheckCapabilities(classifier.getDoNotCheckCapabilities());
    setSeed(classifier.getSeed());
    setDebug(classifier.getDebug());
    setNumDecimalPlaces(classifier.getNumDecimalPlaces());
    setBatchSize(classifier.getBatchSize());
    classifier.setDoNotCheckCapabilities(true);

    // Set base classifier and options
    setClassifier(classifier);

    Utils.checkForRemainingOptions(options);
  }

  /**
   * Returns the revision string.
   *
   * @return the revision
   */
  @Override
  public String getRevision() {
    return RevisionUtils.extract("$Revision$");
  }

  /**
   * Main method for this class.
   *
```

```
  * @param argv the options
  */
 public static void main(String[] argv) {
   runClassifier(new RandomForest(), argv);
 }
}
```

11.2 Source Code for Naive Bayes

Link for the source code:
https://github.com/Waikato/weka-3.8/blob/master/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 3 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, see <http://www.gnu.org/licenses/>.
*/

/*
*    NaiveBayes.java
*    Copyright (C) 1999-2012 University of Waikato, Hamilton, New Zealand
*
*/

package weka.classifiers.bayes;

import java.util.Collections;
import java.util.Enumeration;
import java.util.Vector;
```

```
import weka.classifiers.AbstractClassifier;
import weka.core.*;
import weka.core.Capabilities.Capability;
import weka.core.TechnicalInformation.Field;
import weka.core.TechnicalInformation.Type;
import weka.estimators.DiscreteEstimator;
import weka.estimators.Estimator;
import weka.estimators.KernelEstimator;
import weka.estimators.NormalEstimator;

/**
* <!-- globalinfo-start --> 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.<br/>
* <br/>
* For more information on Naive Bayes classifiers, see<br/>
* <br/>
* George H. John, Pat Langley: Estimating Continuous Distributions in Bayesian
* Classifiers. In: Eleventh Conference on Uncertainty in Artificial
* Intelligence, San Mateo, 338-345, 1995.
* <p/>
* <!-- globalinfo-end -->
*
* <!-- technical-bibtex-start --> BibTeX:
*
* <pre>
* &#64;inproceedings{John1995,
*    address = {San Mateo},
*    author = {George H. John and Pat Langley},
*    booktitle = {Eleventh Conference on Uncertainty in Artificial
Intelligence},
*    pages = {338-345},
*    publisher = {Morgan Kaufmann},
*    title = {Estimating Continuous Distributions in Bayesian Classifiers},
*    year = {1995}
* }
* </pre>
* <p/>
* <!-- technical-bibtex-end -->
*
* <!-- options-start --> Valid options are:
* <p/>
*
```

```
* <pre>
* -K
*  Use kernel density estimator rather than normal
*  distribution for numeric attributes
* </pre>
*
* <pre>
* -D
*  Use supervised discretization to process numeric attributes
* </pre>
*
* <pre>
* -O
*  Display model in old format (good when there are many classes)
* </pre>
*
* <!-- options-end -->
*
* @author Len Trigg (trigg@cs.waikato.ac.nz)
* @author Eibe Frank (eibe@cs.waikato.ac.nz)
* @version $Revision$
*/
public class NaiveBayes extends AbstractClassifier implements OptionHandler,
 WeightedInstancesHandler, WeightedAttributesHandler,
TechnicalInformationHandler,
 Aggregateable<NaiveBayes> {

 /** for serialization */
 static final long serialVersionUID = 5995231201785697655L;

 /** 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;

 protected boolean m_displayModelInOldFormat = false;

 /**
  * 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"
     + getTechnicalInformation().toString();
 }

 /**
  * Returns an instance of a TechnicalInformation object, containing detailed
  * information about the technical background of this class, e.g., paper
  * reference or book this class is based on.
  *
  * @return the technical information about this class
  */
 @Override
```

```
public TechnicalInformation getTechnicalInformation() {
  TechnicalInformation result;

  result = new TechnicalInformation(Type.INPROCEEDINGS);
  result.setValue(Field.AUTHOR, "George H. John and Pat Langley");
  result.setValue(Field.TITLE,
    "Estimating Continuous Distributions in Bayesian Classifiers");
  result.setValue(Field.BOOKTITLE,
    "Eleventh Conference on Uncertainty in Artificial Intelligence");
  result.setValue(Field.YEAR, "1995");
  result.setValue(Field.PAGES, "338-345");
  result.setValue(Field.PUBLISHER, "Morgan Kaufmann");
  result.setValue(Field.ADDRESS, "San Mateo");

  return result;
}

/**
 * Returns default capabilities of the classifier.
 *
 * @return the capabilities of this classifier
 */
@Override
public Capabilities getCapabilities() {
  Capabilities result = super.getCapabilities();
  result.disableAll();

  // attributes
  result.enable(Capability.NOMINAL_ATTRIBUTES);
  result.enable(Capability.NUMERIC_ATTRIBUTES);
  result.enable( Capability.MISSING_VALUES );

  // class
  result.enable(Capability.NOMINAL_CLASS);
  result.enable(Capability.MISSING_CLASS_VALUES);

  // instances
  result.setMinimumNumberInstances(0);

  return result;
}

/**
 * Generates the classifier.
 *
 * @param instances set of instances serving as training data
 * @exception Exception if the classifier has not been generated successfully
 */
@Override
```

```
public void buildClassifier(Instances instances) throws Exception {

  // can classifier handle the data?
  getCapabilities().testWithFail(instances);

  // remove instances with missing class
  instances = new Instances(instances);
  instances.deleteWithMissingClass();

  m_NumClasses = instances.numClasses();

  // 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<Attribute> enu = m_Instances.enumerateAttributes();
  while (enu.hasMoreElements()) {
    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<Instance> enumInsts = m_Instances.enumerateInstances();
  while (enumInsts.hasMoreElements()) {
    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<Attribute> enumAtts = m_Instances.enumerateAttributes();
    int attIndex = 0;
    while (enumAtts.hasMoreElements()) {
      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
 */
@Override
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<Attribute> enumAtts = instance.enumerateAttributes();
  int attIndex = 0;
  while (enumAtts.hasMoreElements()) {
    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, Math.pow(m_Distributions[attIndex][j]
          .getProbability(instance.value(attribute)),
          m_Instances.attribute(attIndex).weight()));
        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.
 */
@Override
public Enumeration<Option> listOptions() {

  Vector<Option> newVector = new Vector<Option>(3);

  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"));

  newVector
    .addElement(new Option(
      "\tDisplay model in old format (good when there are "
        + "many classes)\n", "O", 0, "-O"));

  newVector.addAll(Collections.list(super.listOptions()));

  return newVector.elements();
}

/**
 * Parses a given list of options.
```

```
   * <p/>
   *
   * <!-- options-start --> Valid options are:
   * <p/>
   *
   * <pre>
   * -K
   *  Use kernel density estimator rather than normal
   *  distribution for numeric attributes
   * </pre>
   *
   * <pre>
   * -D
   *  Use supervised discretization to process numeric attributes
   * </pre>
   *
   * <pre>
   * -O
   *  Display model in old format (good when there are many classes)
   * </pre>
   *
   * <!-- options-end -->
   *
   * @param options the list of options as an array of strings
   * @exception Exception if an option is not supported
   */
  @Override
  public void setOptions(String[] options) throws Exception {

    super.setOptions(options);
    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);
    setDisplayModelInOldFormat(Utils.getFlag('O', options));
    Utils.checkForRemainingOptions(options);
  }

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

```
  Vector<String> options = new Vector<String>();

  Collections.addAll(options, super.getOptions());

  if (m_UseKernelEstimator) {
    options.add("-K");
  }

  if (m_UseDiscretization) {
    options.add("-D");
  }

  if (m_displayModelInOldFormat) {
    options.add("-O");
  }

  return options.toArray(new String[0]);
}

/**
 * Returns a description of the classifier.
 *
 * @return a description of the classifier as a string.
 */
@Override
public String toString() {
  if (m_displayModelInOldFormat) {
    return toStringOriginal();
  }

  StringBuffer temp = new StringBuffer();
  temp.append("Naive Bayes Classifier");
  if (m_Instances == null) {
    temp.append(": No model built yet.");
  } else {

    int maxWidth = 0;
    int maxAttWidth = 0;
    boolean containsKernel = false;

    // set up max widths
    // class values
    for (int i = 0; i < m_Instances.numClasses(); i++) {
      if (m_Instances.classAttribute().value(i).length() > maxWidth) {
        maxWidth = m_Instances.classAttribute().value(i).length();
      }
    }
    // attributes
```

```
for (int i = 0; i < m_Instances.numAttributes(); i++) {
  if (i != m_Instances.classIndex()) {
    Attribute a = m_Instances.attribute(i);
    if (a.name().length() > maxAttWidth) {
      maxAttWidth = m_Instances.attribute(i).name().length();
    }
    if (a.isNominal()) {
      // check values
      for (int j = 0; j < a.numValues(); j++) {
        String val = a.value(j) + "  ";
        if (val.length() > maxAttWidth) {
          maxAttWidth = val.length();
        }
      }
    }
  }
}

for (Estimator[] m_Distribution : m_Distributions) {
  for (int j = 0; j < m_Instances.numClasses(); j++) {
    if (m_Distribution[0] instanceof NormalEstimator) {
      // check mean/precision dev against maxWidth
      NormalEstimator n = (NormalEstimator) m_Distribution[j];
      double mean = Math.log(Math.abs(n.getMean())) / Math.log(10.0);
      double precision = Math.log(Math.abs(n.getPrecision()))
        / Math.log(10.0);
      double width = (mean > precision) ? mean : precision;
      if (width < 0) {
        width = 1;
      }
      // decimal + # decimal places + 1
      width += 6.0;
      if ((int) width > maxWidth) {
        maxWidth = (int) width;
      }
    } else if (m_Distribution[0] instanceof KernelEstimator) {
      containsKernel = true;
      KernelEstimator ke = (KernelEstimator) m_Distribution[j];
      int numK = ke.getNumKernels();
      String temps = "K" + numK + ": mean (weight)";
      if (maxAttWidth < temps.length()) {
        maxAttWidth = temps.length();
      }
      // check means + weights against maxWidth
      if (ke.getNumKernels() > 0) {
        double[] means = ke.getMeans();
        double[] weights = ke.getWeights();
        for (int k = 0; k < ke.getNumKernels(); k++) {
          String m = Utils.doubleToString(means[k], maxWidth, 4).trim();
```

```
            m += " ("
              + Utils.doubleToString(weights[k], maxWidth, 1).trim() + ")";
            if (maxWidth < m.length()) {
              maxWidth = m.length();
            }
          }
        }
      } else if (m_Distribution[0] instanceof DiscreteEstimator) {
        DiscreteEstimator d = (DiscreteEstimator) m_Distribution[j];
        for (int k = 0; k < d.getNumSymbols(); k++) {
          String size = "" + d.getCount(k);
          if (size.length() > maxWidth) {
            maxWidth = size.length();
          }
        }
        int sum = ("" + d.getSumOfCounts()).length();
        if (sum > maxWidth) {
          maxWidth = sum;
        }
      }
    }
  }

  // Check width of class labels
  for (int i = 0; i < m_Instances.numClasses(); i++) {
    String cSize = m_Instances.classAttribute().value(i);
    if (cSize.length() > maxWidth) {
      maxWidth = cSize.length();
    }
  }

  // Check width of class priors
  for (int i = 0; i < m_Instances.numClasses(); i++) {
    String priorP = Utils.doubleToString(
      ((DiscreteEstimator) m_ClassDistribution).getProbability(i),
      maxWidth, 2).trim();
    priorP = "(" + priorP + ")";
    if (priorP.length() > maxWidth) {
      maxWidth = priorP.length();
    }
  }

  if (maxAttWidth < "Attribute".length()) {
    maxAttWidth = "Attribute".length();
  }

  if (maxAttWidth < "  weight sum".length()) {
    maxAttWidth = "  weight sum".length();
  }
```

```
if (containsKernel) {
  if (maxAttWidth < "  [precision]".length()) {
    maxAttWidth = "  [precision]".length();
  }
}

maxAttWidth += 2;

temp.append("\n\n");
temp.append(pad("Class", " ",
  (maxAttWidth + maxWidth + 1) - "Class".length(), true));

temp.append("\n");
temp.append(pad("Attribute", " ", maxAttWidth - "Attribute".length(),
  false));
// class labels
for (int i = 0; i < m_Instances.numClasses(); i++) {
  String classL = m_Instances.classAttribute().value(i);
  temp.append(pad(classL, " ", maxWidth + 1 - classL.length(), true));
}
temp.append("\n");
// class priors
temp.append(pad("", " ", maxAttWidth, true));
for (int i = 0; i < m_Instances.numClasses(); i++) {
  String priorP = Utils.doubleToString(
    ((DiscreteEstimator) m_ClassDistribution).getProbability(i),
    maxWidth, 2).trim();
  priorP = "(" + priorP + ")";
  temp.append(pad(priorP, " ", maxWidth + 1 - priorP.length(), true));
}
temp.append("\n");
temp.append(pad(
  "",
  "=",
  maxAttWidth + (maxWidth * m_Instances.numClasses())
    + m_Instances.numClasses() + 1, true));
temp.append("\n");

// loop over the attributes
int counter = 0;
for (int i = 0; i < m_Instances.numAttributes(); i++) {
  if (i == m_Instances.classIndex()) {
    continue;
  }
  String attName = m_Instances.attribute(i).name();
  temp.append(attName + "\n");

  if (m_Distributions[counter][0] instanceof NormalEstimator) {
```

```
      String meanL = "  mean";
      temp.append(pad(meanL, " ", maxAttWidth + 1 - meanL.length(), false));
      for (int j = 0; j < m_Instances.numClasses(); j++) {
        // means
        NormalEstimator n = (NormalEstimator) m_Distributions[counter][j];
        String mean = Utils.doubleToString(n.getMean(), maxWidth, 4).trim();
        temp.append(pad(mean, " ", maxWidth + 1 - mean.length(), true));
      }
      temp.append("\n");
      // now do std deviations
      String stdDevL = "  std. dev.";
      temp.append(pad(stdDevL, " ", maxAttWidth + 1 - stdDevL.length(),
        false));
      for (int j = 0; j < m_Instances.numClasses(); j++) {
        NormalEstimator n = (NormalEstimator) m_Distributions[counter][j];
        String stdDev = Utils.doubleToString(n.getStdDev(), maxWidth, 4)
          .trim();
        temp.append(pad(stdDev, " ", maxWidth + 1 - stdDev.length(), true));
      }
      temp.append("\n");
      // now the weight sums
      String weightL = "  weight sum";
      temp.append(pad(weightL, " ", maxAttWidth + 1 - weightL.length(),
        false));
      for (int j = 0; j < m_Instances.numClasses(); j++) {
        NormalEstimator n = (NormalEstimator) m_Distributions[counter][j];
        String weight = Utils.doubleToString(n.getSumOfWeights(), maxWidth,
          4).trim();
        temp.append(pad(weight, " ", maxWidth + 1 - weight.length(), true));
      }
      temp.append("\n");
      // now the precisions
      String precisionL = "  precision";
      temp.append(pad(precisionL, " ",
        maxAttWidth + 1 - precisionL.length(), false));
      for (int j = 0; j < m_Instances.numClasses(); j++) {
        NormalEstimator n = (NormalEstimator) m_Distributions[counter][j];
        String precision = Utils.doubleToString(n.getPrecision(), maxWidth,
          4).trim();
        temp.append(pad(precision, " ", maxWidth + 1 - precision.length(),
          true));
      }
      temp.append("\n\n");

    } else if (m_Distributions[counter][0] instanceof DiscreteEstimator) {
      Attribute a = m_Instances.attribute(i);
      for (int j = 0; j < a.numValues(); j++) {
        String val = "  " + a.value(j);
        temp.append(pad(val, " ", maxAttWidth + 1 - val.length(), false));
```

```
            for (int k = 0; k < m_Instances.numClasses(); k++) {
              DiscreteEstimator d = (DiscreteEstimator)
m_Distributions[counter][k];
              String count = "" + d.getCount(j);
              temp.append(pad(count, " ", maxWidth + 1 - count.length(), true));
            }
            temp.append("\n");
          }
          // do the totals
          String total = "  [total]";
          temp.append(pad(total, " ", maxAttWidth + 1 - total.length(), false));
          for (int k = 0; k < m_Instances.numClasses(); k++) {
            DiscreteEstimator d = (DiscreteEstimator)
m_Distributions[counter][k];
            String count = "" + d.getSumOfCounts();
            temp.append(pad(count, " ", maxWidth + 1 - count.length(), true));
          }
          temp.append("\n\n");
        } else if (m_Distributions[counter][0] instanceof KernelEstimator) {
          String kL = "  [# kernels]";
          temp.append(pad(kL, " ", maxAttWidth + 1 - kL.length(), false));
          for (int k = 0; k < m_Instances.numClasses(); k++) {
            KernelEstimator ke = (KernelEstimator) m_Distributions[counter][k];
            String nk = "" + ke.getNumKernels();
            temp.append(pad(nk, " ", maxWidth + 1 - nk.length(), true));
          }
          temp.append("\n");
          // do num kernels, std. devs and precisions
          String stdDevL = "  [std. dev]";
          temp.append(pad(stdDevL, " ", maxAttWidth + 1 - stdDevL.length(),
            false));
          for (int k = 0; k < m_Instances.numClasses(); k++) {
            KernelEstimator ke = (KernelEstimator) m_Distributions[counter][k];
            String stdD = Utils.doubleToString(ke.getStdDev(), maxWidth, 4)
              .trim();
            temp.append(pad(stdD, " ", maxWidth + 1 - stdD.length(), true));
          }
          temp.append("\n");
          String precL = "  [precision]";
          temp.append(pad(precL, " ", maxAttWidth + 1 - precL.length(), false));
          for (int k = 0; k < m_Instances.numClasses(); k++) {
            KernelEstimator ke = (KernelEstimator) m_Distributions[counter][k];
            String prec = Utils.doubleToString(ke.getPrecision(), maxWidth, 4)
              .trim();
            temp.append(pad(prec, " ", maxWidth + 1 - prec.length(), true));
          }
          temp.append("\n");
          // first determine max number of kernels accross the classes
          int maxK = 0;
```

```
          for (int k = 0; k < m_Instances.numClasses(); k++) {
            KernelEstimator ke = (KernelEstimator) m_Distributions[counter][k];
            if (ke.getNumKernels() > maxK) {
              maxK = ke.getNumKernels();
            }
          }
          for (int j = 0; j < maxK; j++) {
            // means first
            String meanL = "  K" + (j + 1) + ": mean (weight)";
            temp
              .append(pad(meanL, " ", maxAttWidth + 1 - meanL.length(), false));
            for (int k = 0; k < m_Instances.numClasses(); k++) {
              KernelEstimator ke = (KernelEstimator)
m_Distributions[counter][k];
              double[] means = ke.getMeans();
              double[] weights = ke.getWeights();
              String m = "--";
              if (ke.getNumKernels() == 0) {
                m = "" + 0;
              } else if (j < ke.getNumKernels()) {
                m = Utils.doubleToString(means[j], maxWidth, 4).trim();
                m += " ("
                  + Utils.doubleToString(weights[j], maxWidth, 1).trim() + ")";
              }
              temp.append(pad(m, " ", maxWidth + 1 - m.length(), true));
            }
            temp.append("\n");
          }
          temp.append("\n");
        }

        counter++;
      }
    }

    return temp.toString();
  }

  /**
   * Returns a description of the classifier in the old format.
   *
   * @return a description of the classifier as a string.
   */
  protected String toStringOriginal() {

    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<Attribute> enumAtts = m_Instances.enumerateAttributes();
          int attIndex = 0;
          while (enumAtts.hasMoreElements()) {
            Attribute attribute = enumAtts.nextElement();
            if (attribute.weight() > 0) {
              text.append(attribute.name() + ":  "
                + m_Distributions[attIndex][i]);
            }
            attIndex++;
          }
        }
      } catch (Exception ex) {
        text.append(ex.getMessage());
      }
    }

    return text.toString();
  }

  private String pad(String source, String padChar, int length, boolean leftPad)
  {
    StringBuffer temp = new StringBuffer();

    if (leftPad) {
      for (int i = 0; i < length; i++) {
        temp.append(padChar);
      }
      temp.append(source);
    } else {
      temp.append(source);
      for (int i = 0; i < length; i++) {
        temp.append(padChar);
      }
    }
    return temp.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);
    }
  }

  /**
   * Returns the tip text for this property
   *
   * @return tip text for this property suitable for displaying in the
   *         explorer/experimenter gui
   */
  public String displayModelInOldFormatTipText() {
    return "Use old format for model output. The old format is "
      + "better when there are many class values. The new format "
      + "is better when there are fewer classes and many attributes.";
  }

  /**
   * Set whether to display model output in the old, original format.
   *
   * @param d true if model ouput is to be shown in the old format
   */
  public void setDisplayModelInOldFormat(boolean d) {
    m_displayModelInOldFormat = d;
  }

  /**
   * Get whether to display model output in the old, original format.
   *
   * @return true if model ouput is to be shown in the old format
   */
  public boolean getDisplayModelInOldFormat() {
    return m_displayModelInOldFormat;
  }

  /**
   * Return the header that this classifier was trained with
   *
```

```
   * @return the header that this classifier was trained with
   */
  public Instances getHeader() {
    return m_Instances;
  }

  /**
   * Get all the conditional estimators.
   *
   * @return all the conditional estimators.
   */
  public Estimator[][] getConditionalEstimators() {
    return m_Distributions;
  }

  /**
   * Get the class estimator.
   *
   * @return the class estimator
   */
  public Estimator getClassEstimator() {
    return m_ClassDistribution;
  }

  /**
   * Returns the revision string.
   *
   * @return the revision
   */
  @Override
  public String getRevision() {
    return RevisionUtils.extract("$Revision$");
  }

  @SuppressWarnings({ "rawtypes", "unchecked" })
  @Override
  public NaiveBayes aggregate(NaiveBayes toAggregate) throws Exception {

    // Highly unlikely that discretization intervals will match between the
    // two classifiers
    if (m_UseDiscretization || toAggregate.getUseSupervisedDiscretization()) {
      throw new Exception("Unable to aggregate when supervised discretization "
        + "has been turned on");
    }

    if (!m_Instances.equalHeaders(toAggregate.m_Instances)) {
      throw new Exception("Can't aggregate - data headers don't match: "
        + m_Instances.equalHeadersMsg(toAggregate.m_Instances));
    }
```

```
    ((Aggregateable) m_ClassDistribution)
      .aggregate(toAggregate.m_ClassDistribution);

    // aggregate all conditional estimators
    for (int i = 0; i < m_Distributions.length; i++) {
      for (int j = 0; j < m_Distributions[i].length; j++) {
        ((Aggregateable) m_Distributions[i][j])
          .aggregate(toAggregate.m_Distributions[i][j]);
      }
    }

    return this;
  }

  @Override
  public void finalizeAggregation() throws Exception {
    // nothing to do
  }

  /**
   * Main method for testing this class.
   *
   * @param argv the options
   */
  public static void main(String[] argv) {
    runClassifier(new NaiveBayes(), argv);
  }
}
```