# Final Term Project

## Option 1 ( Supervised Data Mining : Classification )

## Data Mining 634-002

Supervised By :

Dr.Jason T.L. Wang

Professor,Department of Computer Science New Jersey Institute of Technology

Made By :

Neel Patel ( nap73@njit.edu )

# Index

# Introduction

This is a project report as a part of my final term project for CS 634-002 Data Mining under professor Dr. Jason Wang .

I choose option 1 - Supervised Data Mining : Classification from the list of options provided by the professor. I am going to implement and study algorithms from category 4 : Bayesian Networks (Weka BayeNet) and category 5 : Naïve Bayes(Weka Naïve Bayes) . The main aim of this project is to research about the algorithms selected and compare them by executing them against common datasets. The idea is to learn about the accuracy of this algorithms .

For the purpose of the project, I will be using WEKA – a open source software tool for machine learning which is written in java language and has it provides a good collections of algorithms for data analysis and visualization purpose.

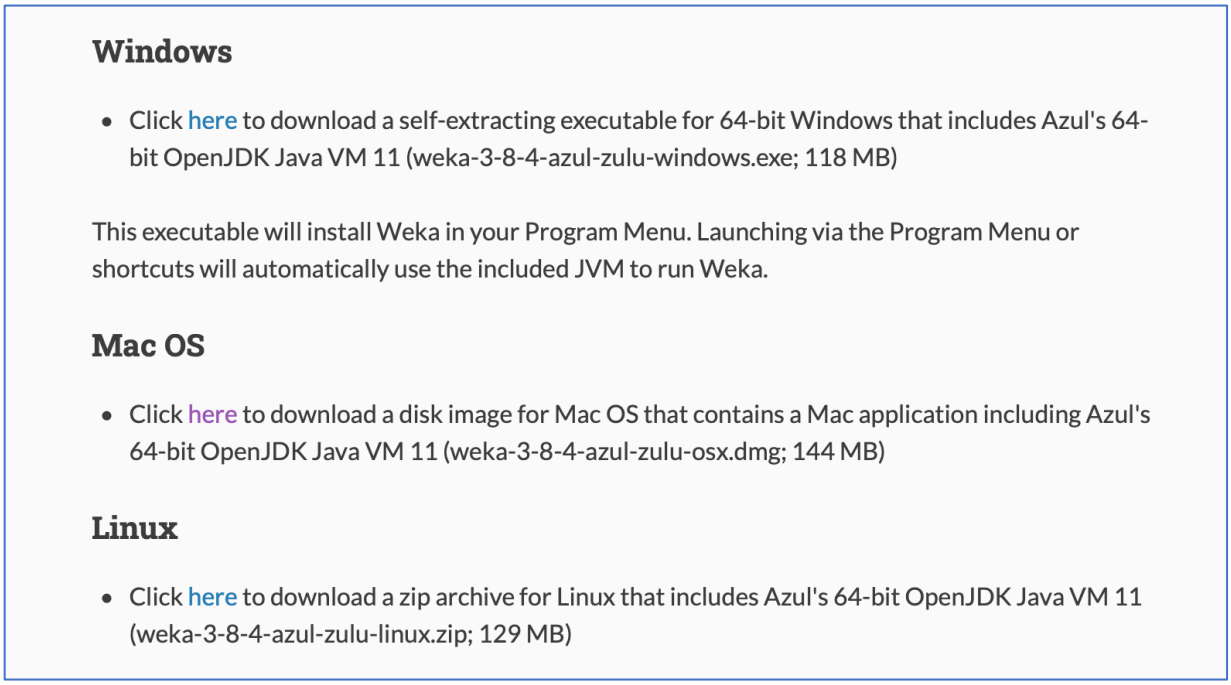
# System Overview

| | |
|---|---|
| Weka : | version 3.8.4 |
| Java : | java version "1.8.0_221" |
| OS : | macOS Cataline version 10.15.1 |
| Processor : | 1.6 GHz Dual-Core Intel Core i5 |
| Memory : | 16 GB 2133 MHz LPDDR3 |

# Pre-requisites & System Preparation

1.Download WEKA from link provided.
https://waikato.github.io/weka-wiki/downloading_weka/

Check for different Mac / Window package available. Choose according to your OS requirements.



**Windows**

- Click here to download a self-extracting executable for 64-bit Windows that includes Azul's 64-bit OpenJDK Java VM 11 (weka-3-8-4-azul-zulu-windows.exe; 118 MB)

This executable will install Weka in your Program Menu. Launching via the Program Menu or shortcuts will automatically use the included JVM to run Weka.

**Mac OS**

- Click here to download a disk image for Mac OS that contains a Mac application including Azul's 64-bit OpenJDK Java VM 11 (weka-3-8-4-azul-zulu-osx.dmg; 144 MB)

**Linux**

- Click here to download a zip archive for Linux that includes Azul's 64-bit OpenJDK Java VM 11 (weka-3-8-4-azul-zulu-linux.zip; 129 MB)

2.Download JAVA from link provided.
https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html

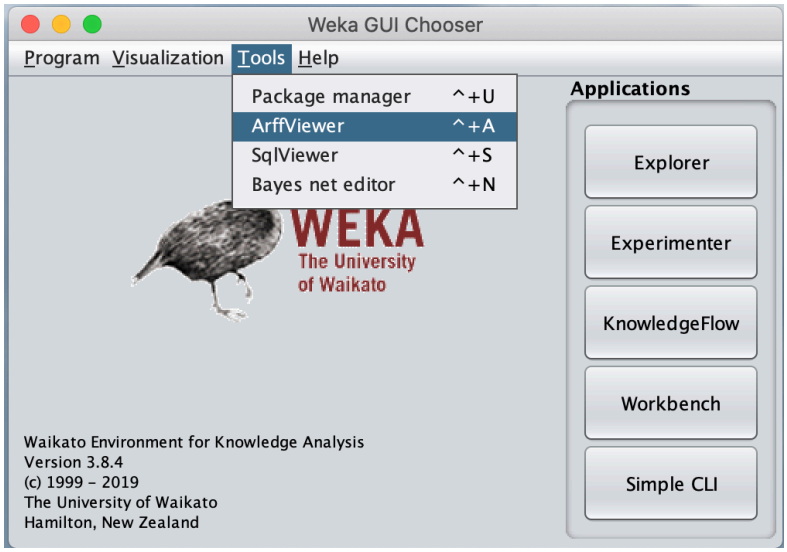
Helpful Link :
https://docs.oracle.com/javase/8/docs/technotes/guides/install/mac_jdk.html

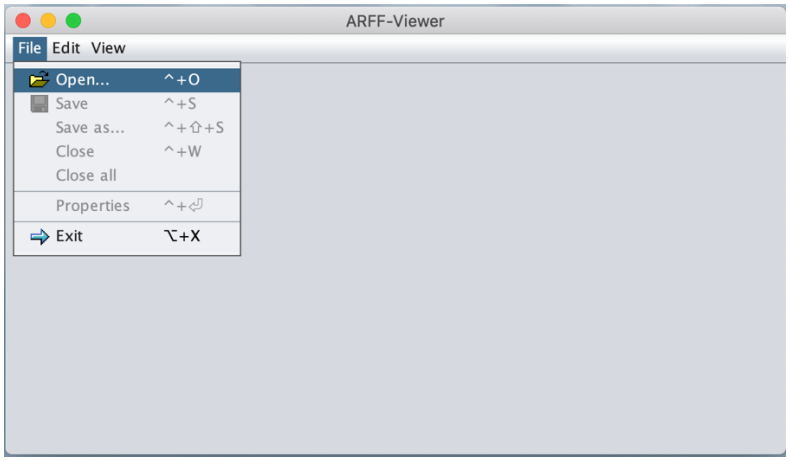
Check for different Mac / Window / Linux package available. Choose according to your OS requirements.

# Converting Dataset from CSV to ARFF

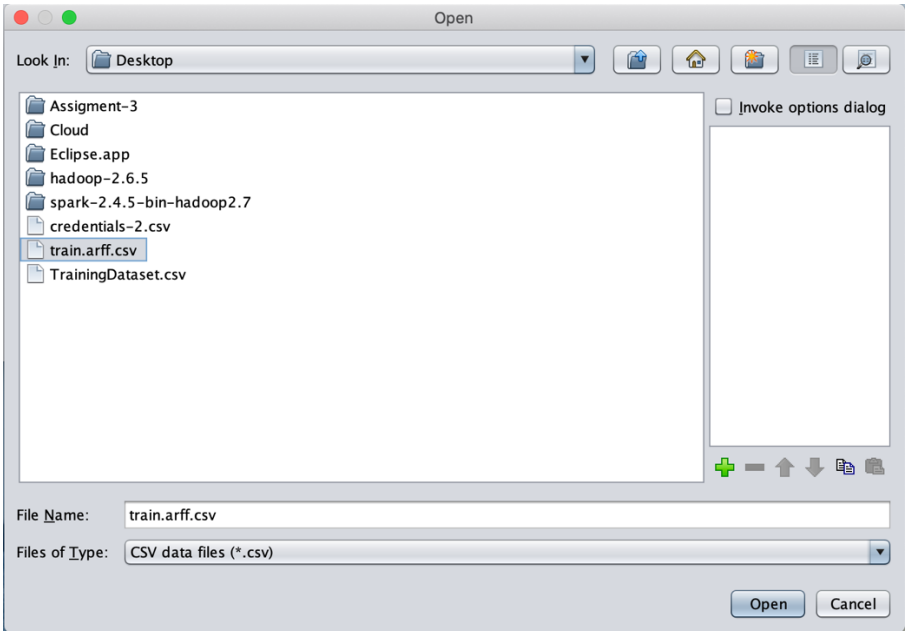
Weka usually supports the dataset of the format ARFF format. If you have the dataset in ARFF format than its good. Incase you have it in csv format, you can convert it to ARFF using WEKA functionality. Here I am mentioning the process to for the conversion.

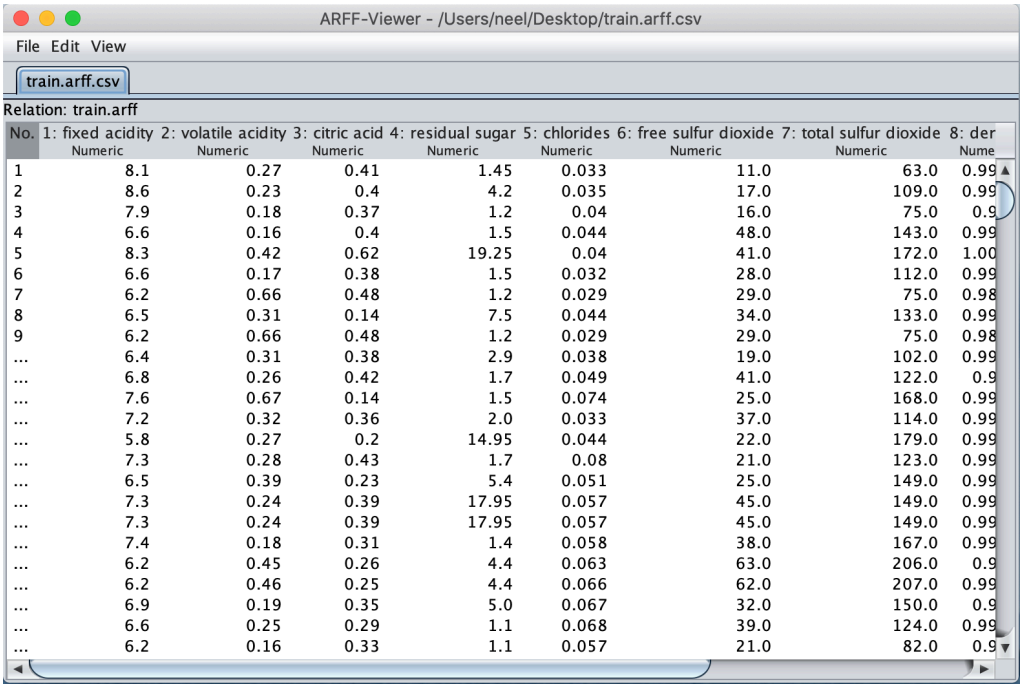
1.Open Weka
2.Goto Tools > ArffViewer > File > Open



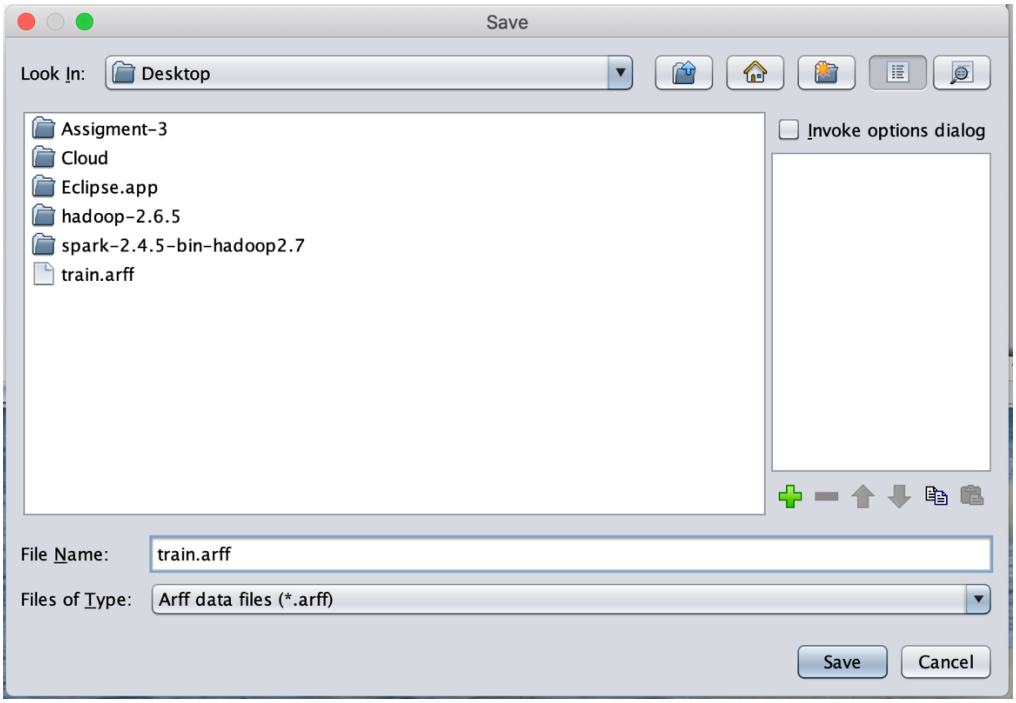
3.Select your .csv dataset from file explorer > Open
You can view the dataset in .csv format displayed on ARFF-Viewer tab.




| No. | 1: fixed acidity | 2: volatile acidity | 3: citric acid | 4: residual sugar | 5: chlorides | 6: free sulfur dioxide | 7: total sulfur dioxide | 8: der |
|---|---|---|---|---|---|---|---|---|
| | Numeric | Numeric | Numeric | Numeric | Numeric | Numeric | Numeric | Nume |
| 1 | 8.1 | 0.27 | 0.41 | 1.45 | 0.033 | 11.0 | 63.0 | 0.99 |
| 2 | 8.6 | 0.23 | 0.4 | 4.2 | 0.035 | 17.0 | 109.0 | 0.99 |
| 3 | 7.9 | 0.18 | 0.37 | 1.2 | 0.04 | 16.0 | 75.0 | 0.9 |
| 4 | 6.6 | 0.16 | 0.4 | 1.5 | 0.044 | 48.0 | 143.0 | 0.99 |
| 5 | 8.3 | 0.42 | 0.62 | 19.25 | 0.04 | 41.0 | 172.0 | 1.00 |
| 6 | 6.6 | 0.17 | 0.38 | 1.5 | 0.032 | 28.0 | 112.0 | 0.99 |
| 7 | 6.2 | 0.66 | 0.48 | 1.2 | 0.029 | 29.0 | 75.0 | 0.98 |
| 8 | 6.5 | 0.31 | 0.14 | 7.5 | 0.044 | 34.0 | 133.0 | 0.99 |
| 9 | 6.2 | 0.66 | 0.48 | 1.2 | 0.029 | 29.0 | 75.0 | 0.98 |
| ... | 6.4 | 0.31 | 0.38 | 2.9 | 0.038 | 19.0 | 102.0 | 0.99 |
| ... | 6.8 | 0.26 | 0.42 | 1.7 | 0.049 | 41.0 | 122.0 | 0.9 |
| ... | 7.6 | 0.67 | 0.14 | 1.5 | 0.074 | 25.0 | 168.0 | 0.99 |
| ... | 7.2 | 0.32 | 0.36 | 2.0 | 0.033 | 37.0 | 114.0 | 0.99 |
| ... | 5.8 | 0.27 | 0.2 | 14.95 | 0.044 | 22.0 | 179.0 | 0.99 |
| ... | 7.3 | 0.28 | 0.43 | 1.7 | 0.08 | 21.0 | 123.0 | 0.99 |
| ... | 6.5 | 0.39 | 0.23 | 5.4 | 0.051 | 25.0 | 149.0 | 0.99 |
| ... | 7.3 | 0.24 | 0.39 | 17.95 | 0.057 | 45.0 | 149.0 | 0.99 |
| ... | 7.3 | 0.24 | 0.39 | 17.95 | 0.057 | 45.0 | 149.0 | 0.99 |
| ... | 7.4 | 0.18 | 0.31 | 1.4 | 0.058 | 38.0 | 167.0 | 0.99 |
| ... | 6.2 | 0.45 | 0.26 | 4.4 | 0.063 | 63.0 | 206.0 | 0.9 |
| ... | 6.2 | 0.46 | 0.25 | 4.4 | 0.066 | 62.0 | 207.0 | 0.99 |
| ... | 6.9 | 0.19 | 0.35 | 5.0 | 0.067 | 32.0 | 150.0 | 0.9 |
| ... | 6.6 | 0.25 | 0.29 | 1.1 | 0.068 | 39.0 | 124.0 | 0.99 |
| ... | 6.2 | 0.16 | 0.33 | 1.1 | 0.057 | 21.0 | 82.0 | 0.9 |

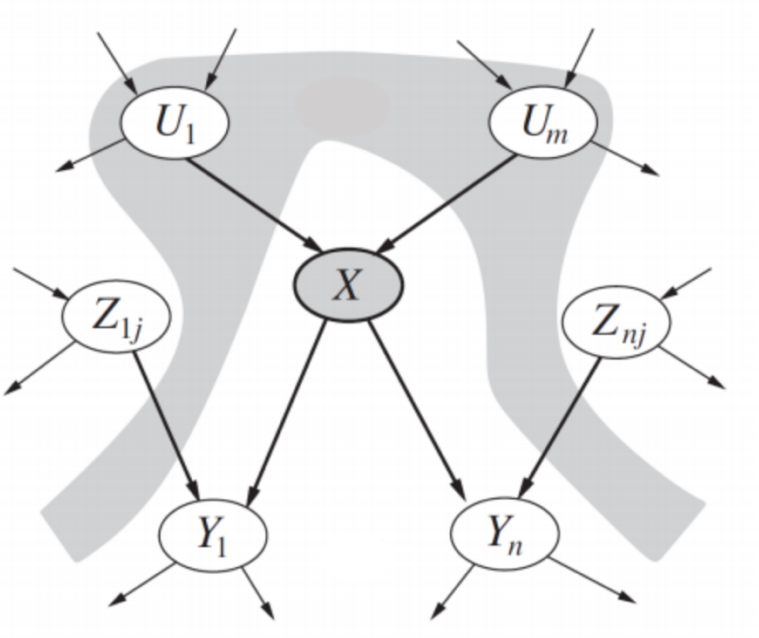
4.Go ahead and said the open .csv file into ARFF data type format.
You will see .arff formatted dataset file on your saved location.



# Algorithm Study

1.Bayesian Networks ( Category 4 )

**Bayesian networks** are a type of probabilistic graphical model that uses Bayesian inference for probability computations. Bayesian networks aim to model conditional dependence, and therefore causation, by representing conditional dependence by edges in a directed graph. Through these relationships, one can efficiently conduct inference on the random variables in the graph through the use of factors.



2.Naive Bayes ( Category 5 )

Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem. Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods. An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification.

**Bayes Theorem:**

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

Using Bayes theorem, we can find the probability of A happening, given that B has occurred. Here, B is the evidence and A is the hypothesis. The assumption made here is that the predictors/features are independent. That is presence of one particular feature does not affect the other. Hence it is called naïve.

# Dataset Used

## Wine Prediction Train Dataset

http://archive.ics.uci.edu/ml/datasets/Wine+Quality
Instances : 1890
Attributes : 12

The datasets are related to red and white variants of the Portuguese "Vinho Verde" wine.

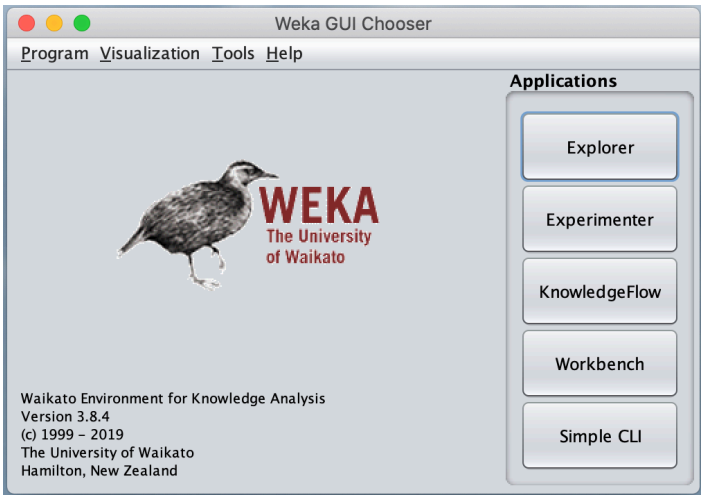
These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are many more normal wines than excellent or poor ones). Outlier detection algorithms could be used to detect the few excellent or poor wines. Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods.

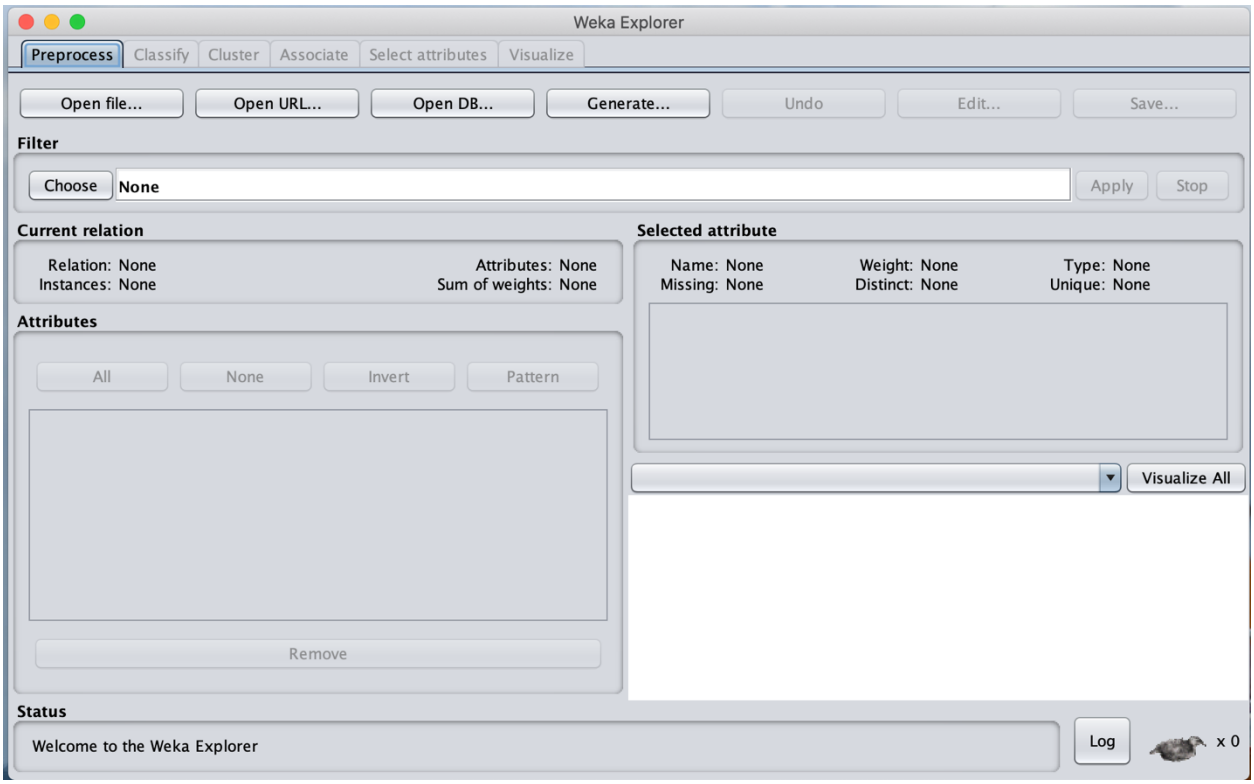
Input variables (based on physicochemical tests):
1 - fixed acidity
2 - volatile acidity
3 - citric acid
4 - residual sugar
5 - chlorides
6 - free sulfur dioxide
7 - total sulfur dioxide
8 - density
9 - pH
10 - sulphates
11 - alcohol

Output variable (based on sensory data):
12 - quality (score between 0 and 10)

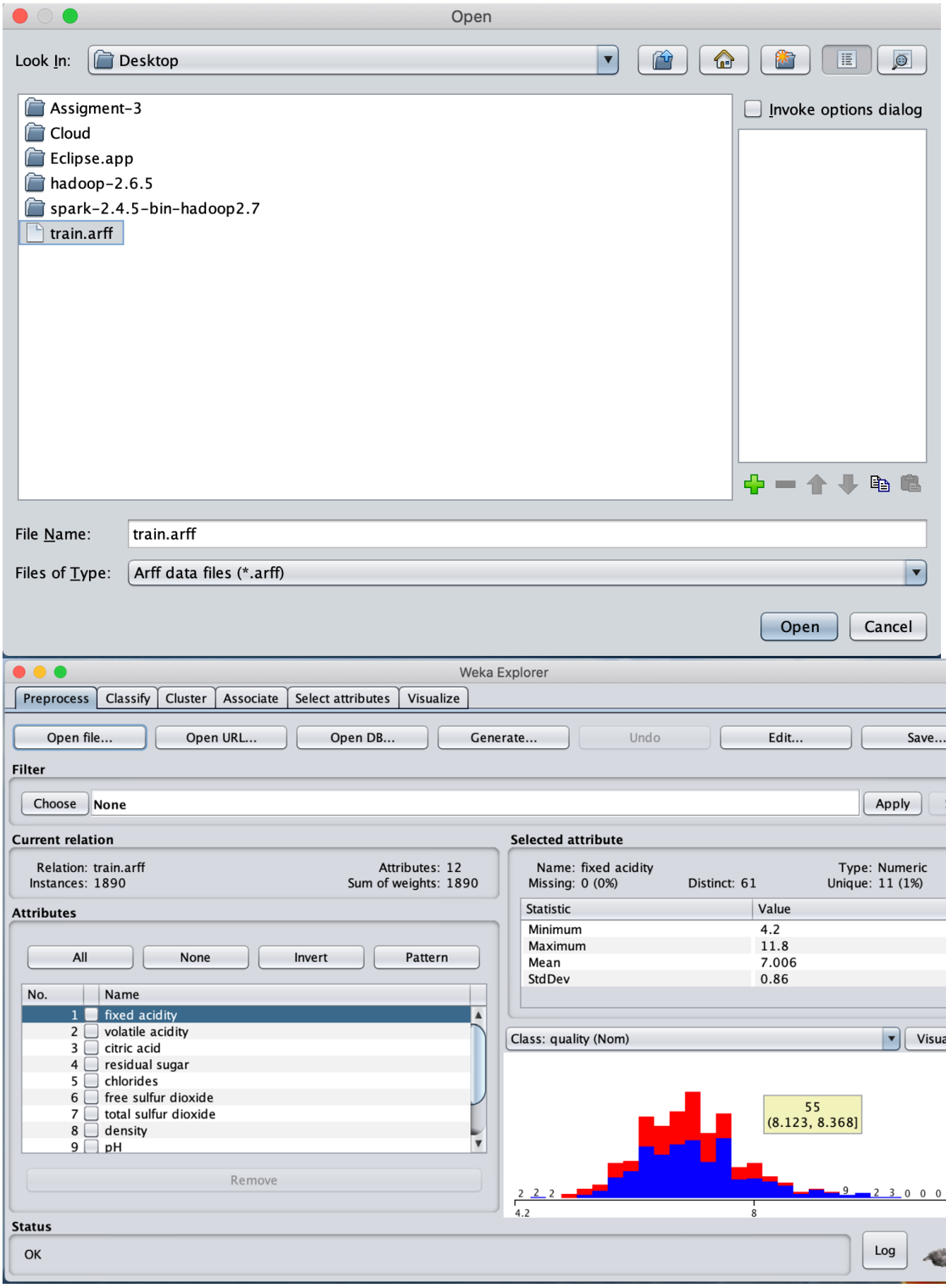
# Implementation of Bayesian Network

1.Open Weka
2.Weka > Weka Explorer

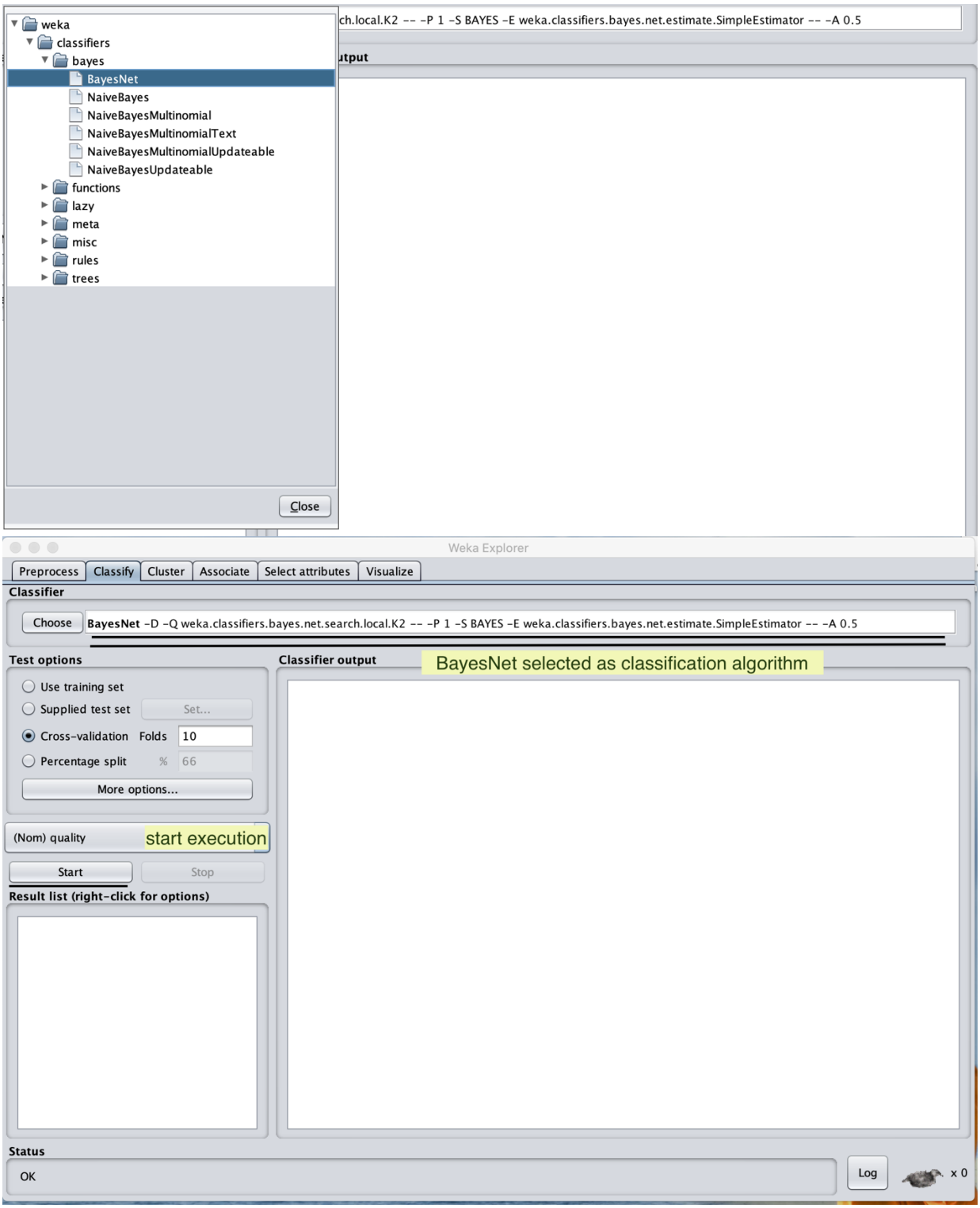


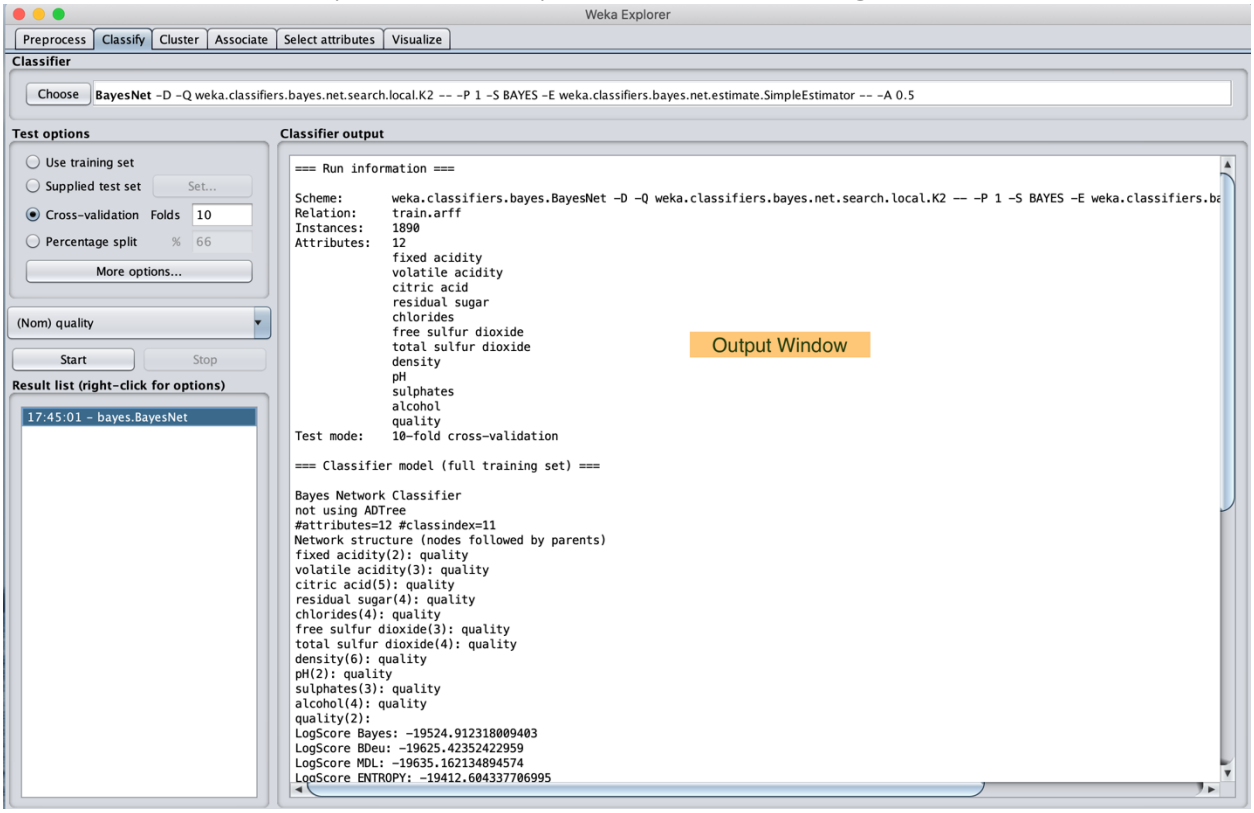
3.Open File > Select .arff dataset file > Open
4. You will see the dataset and its attributes opened in window.



5. select classify > classifiers > bayes > BayesNet
6. You will see the bayesNet selected in choose window. > Press start to execute.



7. You will see the output in the output window on the right side.




## Conclusion :

From the above screenshots, we clearly see that the accuracy of BayesNet for the given dataset is 81.16%.

# Output of Bayesian Network

```
=== Run information ===

Scheme:       weka.classifiers.bayes.BayesNet -D -Q weka.classifiers.bayes.net.search.local.K2
-- -P 1 -S BAYES -E weka.classifiers.bayes.net.estimate.SimpleEstimator -- -A 0.5
Relation:     train.arff
Instances:    1890
Attributes:   12
              fixed acidity
              volatile acidity
              citric acid
              residual sugar
              chlorides
              free sulfur dioxide
              total sulfur dioxide
              density
              pH
              sulphates
              alcohol
              quality
Test mode:    10-fold cross-validation

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

Bayes Network Classifier
not using ADTree
#attributes=12 #classindex=11
Network structure (nodes followed by parents)
fixed acidity(2): quality
volatile acidity(3): quality
citric acid(5): quality
residual sugar(4): quality
chlorides(4): quality
free sulfur dioxide(3): quality
total sulfur dioxide(4): quality
density(6): quality
pH(2): quality
sulphates(3): quality
alcohol(4): quality
quality(2):
LogScore Bayes: -19524.912318009403
LogScore BDeu: -19625.42352422959
LogScore MDL: -19635.162134894574
LogScore ENTROPY: -19412.604337706995
LogScore AIC: -19471.604337706995
```

Time taken to build model: 0.15 seconds

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

| | | |
|---|---|---|
| Correctly Classified Instances | 1534 | 81.164 % |
| Incorrectly Classified Instances | 356 | 18.836 % |
| Kappa statistic | 0.5987 | |
| Mean absolute error | 0.2082 | |
| Root mean squared error | 0.3868 | |
| Relative absolute error | 44.3543 % | |
| Root relative squared error | 79.8557 % | |
| Total Number of Instances | 1890 | |

=== Detailed Accuracy By Class ===

| | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area | Class |
|---|---|---|---|---|---|---|---|---|---|
| | 0.849 | 0.250 | 0.849 | 0.849 | 0.849 | 0.599 | 0.874 | 0.907 | bad |
| | 0.750 | 0.151 | 0.750 | 0.750 | 0.750 | 0.599 | 0.874 | 0.826 | good |
| Weighted Avg. | 0.812 | 0.213 | 0.812 | 0.812 | 0.812 | 0.599 | 0.874 | 0.877 | |

=== Confusion Matrix ===

| a | b | <-- classified as |
|---|---|---|
| 1001 | 178 | a = bad |
| 178 | 533 | b = good |

# Source Code of BayeNet

```
/*
 *   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/>.
 */

/*
 * BayesNet.java
 * Copyright (C) 2001-2012 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.classifiers.bayes.net.ADNode;
import weka.classifiers.bayes.net.BIFReader;
import weka.classifiers.bayes.net.ParentSet;
import weka.classifiers.bayes.net.estimate.BayesNetEstimator;
import weka.classifiers.bayes.net.estimate.DiscreteEstimatorBayes;
import weka.classifiers.bayes.net.estimate.SimpleEstimator;
import weka.classifiers.bayes.net.search.SearchAlgorithm;
import weka.classifiers.bayes.net.search.local.K2;
import weka.classifiers.bayes.net.search.local.LocalScoreSearchAlgorithm;
import weka.classifiers.bayes.net.search.local.Scoreable;
import weka.core.AdditionalMeasureProducer;
import weka.core.Attribute;
import weka.core.Capabilities;
import weka.core.Capabilities.Capability;
import weka.core.Drawable;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.Option;
import weka.core.OptionHandler;
import weka.core.RevisionUtils;
import weka.core.Utils;
import weka.core.WeightedInstancesHandler;
import weka.estimators.Estimator;
import weka.filters.Filter;
import weka.filters.supervised.attribute.Discretize;
import weka.filters.unsupervised.attribute.ReplaceMissingValues;
```

```
/**
 * <!-- globalinfo-start --> Bayes Network learning using various search
 * algorithms and quality measures.<br/>
 * Base class for a Bayes Network classifier. Provides datastructures (network
 * structure, conditional probability distributions, etc.) and facilities common
 * to Bayes Network learning algorithms like K2 and B.<br/>
 * <br/>
 * For more information see:<br/>
 * <br/>
 * http://sourceforge.net/projects/weka/files/documentation/WekaManual-3-7-0.pdf
 * /download
 * <p/>
 * <!-- globalinfo-end -->
 *
 * <!-- options-start --> Valid options are:
 * <p/>
 *
 * <pre>
 * -D
 *  Do not use ADTree data structure
 * </pre>
 *
 * <pre>
 * -B &lt;BIF file&gt;
 *  BIF file to compare with
 * </pre>
 *
 * <pre>
 * -Q weka.classifiers.bayes.net.search.SearchAlgorithm
 *  Search algorithm
 * </pre>
 *
 * <pre>
 * -E weka.classifiers.bayes.net.estimate.SimpleEstimator
 *  Estimator algorithm
 * </pre>
 *
 * <!-- options-end -->
 *
 * @author Remco Bouckaert (rrb@xm.co.nz)
 * @version $Revision$
 */
public class BayesNet extends AbstractClassifier implements OptionHandler,
  WeightedInstancesHandler, Drawable, AdditionalMeasureProducer {

  /** for serialization */
  static final long serialVersionUID = 746037443258775954L;

  /**
   * The parent sets.
   */
  protected ParentSet[] m_ParentSets;

  /**
   * The attribute estimators containing CPTs.
   */
  public Estimator[][] m_Distributions;
```

```
/** filter used to quantize continuous variables, if any **/
protected Discretize m_DiscretizeFilter = null;

/** attribute index of a non-nominal attribute */
int m_nNonDiscreteAttribute = -1;

/** filter used to fill in missing values, if any **/
protected ReplaceMissingValues m_MissingValuesFilter = null;

/**
 * The number of classes
 */
protected int m_NumClasses;

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

/**
 * The number of instances the model was built from
 */
private int m_NumInstances;

/**
 * Datastructure containing ADTree representation of the database. This may
 * result in more efficient access to the data.
 */
ADNode m_ADTree;

/**
 * Bayes network to compare the structure with.
 */
protected BIFReader m_otherBayesNet = null;

/**
 * Use the experimental ADTree datastructure for calculating contingency
 * tables
 */
boolean m_bUseADTree = false;

/**
 * Search algorithm used for learning the structure of a network.
 */
SearchAlgorithm m_SearchAlgorithm = new K2();

/**
 * Search algorithm used for learning the structure of a network.
 */
BayesNetEstimator m_BayesNetEstimator = new SimpleEstimator();

/**
 * 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
 * @throws 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();

 // ensure we have a data set with discrete variables only and with no
 // missing values
 instances = normalizeDataSet(instances);

 // Copy the instances
 m_Instances = new Instances(instances);
 m_NumInstances = m_Instances.numInstances();

 // sanity check: need more than 1 variable in datat set
 m_NumClasses = instances.numClasses();

 // initialize ADTree
 if (m_bUseADTree) {
  m_ADTree = ADNode.makeADTree(instances);
  // System.out.println("Oef, done!");
 }

 // build the network structure
 initStructure();

 // build the network structure
 buildStructure();
```

```
  // build the set of CPTs
  estimateCPTs();

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

/**
 * Returns the number of instances the model was built from.
 */
public int getNumInstances() {

  return m_NumInstances;
}

/**
 * ensure that all variables are nominal and that there are no missing values
 *
 * @param instances data set to check and quantize and/or fill in missing
 *          values
 * @return filtered instances
 * @throws Exception if a filter (Discretize, ReplaceMissingValues) fails
 */
protected Instances normalizeDataSet(Instances instances) throws Exception {

  m_nNonDiscreteAttribute = -1;
  Enumeration<Attribute> enu = instances.enumerateAttributes();
  while (enu.hasMoreElements()) {
    Attribute attribute = enu.nextElement();
    if (attribute.type() != Attribute.NOMINAL) {
      m_nNonDiscreteAttribute = attribute.index();
    }
  }

  if ((m_nNonDiscreteAttribute > -1)
    && (instances.attribute(m_nNonDiscreteAttribute).type() != Attribute.NOMINAL)) {
    m_DiscretizeFilter = new Discretize();
    m_DiscretizeFilter.setInputFormat(instances);
    instances = Filter.useFilter(instances, m_DiscretizeFilter);
  }

  m_MissingValuesFilter = new ReplaceMissingValues();
  m_MissingValuesFilter.setInputFormat(instances);
  instances = Filter.useFilter(instances, m_MissingValuesFilter);

  return instances;
} // normalizeDataSet

/**
 * ensure that all variables are nominal and that there are no missing values
 *
 * @param instance instance to check and quantize and/or fill in missing
 *          values
 * @return filtered instance
 * @throws Exception if a filter (Discretize, ReplaceMissingValues) fails
 */
```

```
protected Instance normalizeInstance(Instance instance) throws Exception {
  if ((m_nNonDiscreteAttribute > -1)
    && (instance.attribute(m_nNonDiscreteAttribute).type() != Attribute.NOMINAL)) {
    m_DiscretizeFilter.input(instance);
    instance = m_DiscretizeFilter.output();
  }

  m_MissingValuesFilter.input(instance);
  instance = m_MissingValuesFilter.output();

  return instance;
} // normalizeInstance

/**
 * Init structure initializes the structure to an empty graph or a Naive Bayes
 * graph (depending on the -N flag).
 *
 * @throws Exception in case of an error
 */
public void initStructure() throws Exception {

  // initialize topological ordering
  // m_nOrder = new int[m_Instances.numAttributes()];
  // m_nOrder[0] = m_Instances.classIndex();

  int nAttribute = 0;

  for (int iOrder = 1; iOrder < m_Instances.numAttributes(); iOrder++) {
    if (nAttribute == m_Instances.classIndex()) {
      nAttribute++;
    }

    // m_nOrder[iOrder] = nAttribute++;
  }

  // reserve memory
  m_ParentSets = new ParentSet[m_Instances.numAttributes()];

  for (int iAttribute = 0; iAttribute < m_Instances.numAttributes(); iAttribute++) {
    m_ParentSets[iAttribute] = new ParentSet(m_Instances.numAttributes());
  }
} // initStructure

/**
 * buildStructure determines the network structure/graph of the network. The
 * default behavior is creating a network where all nodes have the first node
 * as its parent (i.e., a BayesNet that behaves like a naive Bayes
 * classifier). This method can be overridden by derived classes to restrict
 * the class of network structures that are acceptable.
 *
 * @throws Exception in case of an error
 */
public void buildStructure() throws Exception {
  m_SearchAlgorithm.buildStructure(this, m_Instances);
} // buildStructure

/**
 * estimateCPTs estimates the conditional probability tables for the Bayes Net
```

```
 * using the network structure.
 *
 * @throws Exception in case of an error
 */
public void estimateCPTs() throws Exception {
  m_BayesNetEstimator.estimateCPTs(this);
} // estimateCPTs

/**
 * initializes the conditional probabilities
 *
 * @throws Exception in case of an error
 */
public void initCPTs() throws Exception {
  m_BayesNetEstimator.initCPTs(this);
} // estimateCPTs

/**
 * Updates the classifier with the given instance.
 *
 * @param instance the new training instance to include in the model
 * @throws Exception if the instance could not be incorporated in the model.
 */
public void updateClassifier(Instance instance) throws Exception {
  instance = normalizeInstance(instance);
  m_BayesNetEstimator.updateClassifier(this, instance);
} // updateClassifier

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

/**
 * Calculates the counts for Dirichlet distribution for the class membership
 * probabilities for the given test instance.
 *
 * @param instance the instance to be classified
 * @return counts for Dirichlet distribution for class probability
 * @throws Exception if there is a problem generating the prediction
 */
public double[] countsForInstance(Instance instance) throws Exception {
  double[] fCounts = new double[m_NumClasses];

  for (int iClass = 0; iClass < m_NumClasses; iClass++) {
    fCounts[iClass] = 0.0;
  }

  for (int iClass = 0; iClass < m_NumClasses; iClass++) {
    double fCount = 0;
```

```
    for (int iAttribute = 0; iAttribute < m_Instances.numAttributes(); iAttribute++) {
      double iCPT = 0;

      for (int iParent = 0; iParent < m_ParentSets[iAttribute]
        .getNrOfParents(); iParent++) {
        int nParent = m_ParentSets[iAttribute].getParent(iParent);

        if (nParent == m_Instances.classIndex()) {
          iCPT = iCPT * m_NumClasses + iClass;
        } else {
          iCPT = iCPT * m_Instances.attribute(nParent).numValues()
            + instance.value(nParent);
        }
      }

      if (iAttribute == m_Instances.classIndex()) {
        fCount += ((DiscreteEstimatorBayes) m_Distributions[iAttribute][(int) iCPT])
          .getCount(iClass);
      } else {
        fCount += ((DiscreteEstimatorBayes) m_Distributions[iAttribute][(int) iCPT])
          .getCount(instance.value(iAttribute));
      }
    }

    fCounts[iClass] += fCount;
  }
  return fCounts;
} // countsForInstance

/**
 * 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>(4);

  newVector.addElement(new Option("\tDo not use ADTree data structure\n",
    "D", 0, "-D"));
  newVector.addElement(new Option("\tBIF file to compare with\n", "B", 1,
    "-B <BIF file>"));
  newVector.addElement(new Option("\tSearch algorithm\n", "Q", 1,
    "-Q weka.classifiers.bayes.net.search.SearchAlgorithm"));
  newVector.addElement(new Option("\tEstimator algorithm\n", "E", 1,
    "-E weka.classifiers.bayes.net.estimate.SimpleEstimator"));
  newVector.addAll(Collections.list(super.listOptions()));

  newVector.addElement(new Option("", "", 0,
    "\nOptions specific to search method "
      + getSearchAlgorithm().getClass().getName() + ":"));
  newVector.addAll(Collections.list(getSearchAlgorithm().listOptions()));

  newVector.addElement(new Option("", "", 0,
    "\nOptions specific to estimator method "
      + getEstimator().getClass().getName() + ":"));
  newVector.addAll(Collections.list(getEstimator().listOptions()));
```

```
  return newVector.elements();
} // listOptions

/**
 * Parses a given list of options.
 * <p>
 *
 * <!-- options-start --> Valid options are:
 * <p/>
 *
 * <pre>
 * -D
 *  Do not use ADTree data structure
 * </pre>
 *
 * <pre>
 * -B &lt;BIF file&gt;
 *  BIF file to compare with
 * </pre>
 *
 * <pre>
 * -Q weka.classifiers.bayes.net.search.SearchAlgorithm
 *  Search algorithm
 * </pre>
 *
 * <pre>
 * -E weka.classifiers.bayes.net.estimate.SimpleEstimator
 *  Estimator algorithm
 * </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 {
  m_bUseADTree = !(Utils.getFlag('D', options));

  String sBIFFile = Utils.getOption('B', options);
  if (sBIFFile != null && !sBIFFile.equals("")) {
    setBIFFile(sBIFFile);
  }

  String searchAlgorithmName = Utils.getOption('Q', options);
  if (searchAlgorithmName.length() != 0) {
    setSearchAlgorithm((SearchAlgorithm) Utils.forName(SearchAlgorithm.class,
      searchAlgorithmName, partitionOptions(options)));
  } else {
    setSearchAlgorithm(new K2());
  }

  String estimatorName = Utils.getOption('E', options);
  if (estimatorName.length() != 0) {
    setEstimator((BayesNetEstimator) Utils.forName(BayesNetEstimator.class,
      estimatorName, Utils.partitionOptions(options)));
  } else {
```

```
    setEstimator(new SimpleEstimator());
   }

   super.setOptions(options);
  } // setOptions

  /**
   * Returns the secondary set of options (if any) contained in the supplied
   * options array. The secondary set is defined to be any options after the
   * first "--" but before the "-E". These options are removed from the original
   * options array.
   *
   * @param options the input array of options
   * @return the array of secondary options
   */
  public static String[] partitionOptions(String[] options) {

   for (int i = 0; i < options.length; i++) {
    if (options[i].equals("--")) {
     // ensure it follows by a -E option
     int j = i;
     while ((j < options.length) && !(options[j].equals("-E"))) {
      j++;
     }
     /*
      * if (j >= options.length) { return new String[0]; }
      */
     options[i++] = "";
     String[] result = new String[options.length - i];
     j = i;
     while ((j < options.length) && !(options[j].equals("-E"))) {
      result[j - i] = options[j];
      options[j] = "";
      j++;
     }
     while (j < options.length) {
      result[j - i] = "";
      j++;
     }
     return result;
    }
   }
   return new String[0];
  }

  /**
   * 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_bUseADTree) {
    options.add("-D");
```

```
  }

  if (m_otherBayesNet != null) {
    options.add("-B");
    options.add(m_otherBayesNet.getFileName());
  }

  options.add("-Q");
  options.add("" + getSearchAlgorithm().getClass().getName());
  options.add("--");
  Collections.addAll(options, getSearchAlgorithm().getOptions());

  options.add("-E");
  options.add("" + getEstimator().getClass().getName());
  options.add("--");
  Collections.addAll(options, getEstimator().getOptions());

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

/**
 * Set the SearchAlgorithm used in searching for network structures.
 *
 * @param newSearchAlgorithm the SearchAlgorithm to use.
 */
public void setSearchAlgorithm(SearchAlgorithm newSearchAlgorithm) {
  m_SearchAlgorithm = newSearchAlgorithm;
}

/**
 * Get the SearchAlgorithm used as the search algorithm
 *
 * @return the SearchAlgorithm used as the search algorithm
 */
public SearchAlgorithm getSearchAlgorithm() {
  return m_SearchAlgorithm;
}

/**
 * Set the Estimator Algorithm used in calculating the CPTs
 *
 * @param newBayesNetEstimator the Estimator to use.
 */
public void setEstimator(BayesNetEstimator newBayesNetEstimator) {
  m_BayesNetEstimator = newBayesNetEstimator;
}

/**
 * Get the BayesNetEstimator used for calculating the CPTs
 *
 * @return the BayesNetEstimator used.
 */
public BayesNetEstimator getEstimator() {
  return m_BayesNetEstimator;
}

/**
 * Set whether ADTree structure is used or not
```

```
 *
 * @param bUseADTree true if an ADTree structure is used
 */
public void setUseADTree(boolean bUseADTree) {
  m_bUseADTree = bUseADTree;
}

/**
 * Method declaration
 *
 * @return whether ADTree structure is used or not
 */
public boolean getUseADTree() {
  return m_bUseADTree;
}

/**
 * Set name of network in BIF file to compare with
 *
 * @param sBIFFile the name of the BIF file
 */
public void setBIFFile(String sBIFFile) {
  try {
    m_otherBayesNet = new BIFReader().processFile(sBIFFile);
  } catch (Throwable t) {
    m_otherBayesNet = null;
  }
}

/**
 * Get name of network in BIF file to compare with
 *
 * @return BIF file name
 */
public String getBIFFile() {
  if (m_otherBayesNet != null) {
    return m_otherBayesNet.getFileName();
  }
  return "";
}

/**
 * Returns a description of the classifier.
 *
 * @return a description of the classifier as a string.
 */
@Override
public String toString() {
  StringBuffer text = new StringBuffer();

  text.append("Bayes Network Classifier");
  text.append("\n" + (m_bUseADTree ? "Using " : "not using ") + "ADTree");

  if (m_Instances == null) {
    text.append(": No model built yet.");
  } else {

    // flatten BayesNet down to text
```

```
    text.append("\n#attributes=");
    text.append(m_Instances.numAttributes());
    text.append(" #classindex=");
    text.append(m_Instances.classIndex());
    text.append("\nNetwork structure (nodes followed by parents)\n");

    for (int iAttribute = 0; iAttribute < m_Instances.numAttributes(); iAttribute++) {
      text.append(m_Instances.attribute(iAttribute).name() + "("
        + m_Instances.attribute(iAttribute).numValues() + "): ");

      for (int iParent = 0; iParent < m_ParentSets[iAttribute]
        .getNrOfParents(); iParent++) {
        text.append(m_Instances.attribute(
          m_ParentSets[iAttribute].getParent(iParent)).name()
          + " ");
      }

      text.append("\n");

      // Description of distributions tends to be too much detail, so it is
      // commented out here
      // for (int iParent = 0; iParent <
      // m_ParentSets[iAttribute].GetCardinalityOfParents(); iParent++) {
      // text.append('(' + m_Distributions[iAttribute][iParent].toString() +
      // ')');
      // }
      // text.append("\n");
    }

    text.append("LogScore Bayes: " + measureBayesScore() + "\n");
    text.append("LogScore BDeu: " + measureBDeuScore() + "\n");
    text.append("LogScore MDL: " + measureMDLScore() + "\n");
    text.append("LogScore ENTROPY: " + measureEntropyScore() + "\n");
    text.append("LogScore AIC: " + measureAICScore() + "\n");

    if (m_otherBayesNet != null) {
      text.append("Missing: " + m_otherBayesNet.missingArcs(this)
        + " Extra: " + m_otherBayesNet.extraArcs(this) + " Reversed: "
        + m_otherBayesNet.reversedArcs(this) + "\n");
      text.append("Divergence: " + m_otherBayesNet.divergence(this) + "\n");
    }
   }

   return text.toString();
  } // toString

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

  /**
   * Returns a BayesNet graph in XMLBIF ver 0.3 format.
```

```
 *
 * @return String representing this BayesNet in XMLBIF ver 0.3
 * @throws Exception in case BIF generation fails
 */
@Override
public String graph() throws Exception {
  return toXMLBIF03();
}

public String getBIFHeader() {
  StringBuffer text = new StringBuffer();
  text.append("<?xml version=\"1.0\"?>\n");
  text.append("<!-- DTD for the XMLBIF 0.3 format -->\n");
  text.append("<!DOCTYPE BIF [\n");
  text.append("    <!ELEMENT BIF ( NETWORK )*>\n");
  text.append("        <!ATTLIST BIF VERSION CDATA #REQUIRED>\n");
  text
    .append("    <!ELEMENT NETWORK ( NAME, ( PROPERTY | VARIABLE | DEFINITION )* )>\n");
  text.append("    <!ELEMENT NAME (#PCDATA)>\n");
  text.append("    <!ELEMENT VARIABLE ( NAME, ( OUTCOME |  PROPERTY )* ) >\n");
  text
    .append("        <!ATTLIST VARIABLE TYPE (nature|decision|utility) \"nature\">\n");
  text.append("    <!ELEMENT OUTCOME (#PCDATA)>\n");
  text
    .append("    <!ELEMENT DEFINITION ( FOR | GIVEN | TABLE | PROPERTY )* >\n");
  text.append("    <!ELEMENT FOR (#PCDATA)>\n");
  text.append("    <!ELEMENT GIVEN (#PCDATA)>\n");
  text.append("    <!ELEMENT TABLE (#PCDATA)>\n");
  text.append("    <!ELEMENT PROPERTY (#PCDATA)>\n");
  text.append("]>\n");
  return text.toString();
} // getBIFHeader

/**
 * Returns a description of the classifier in XML BIF 0.3 format. See
 * http://www-2.cs.cmu.edu/~fgcozman/Research/InterchangeFormat/ for details
 * on XML BIF.
 *
 * @return an XML BIF 0.3 description of the classifier as a string.
 */
public String toXMLBIF03() {
  if (m_Instances == null) {
    return ("<!--No model built yet-->");
  }

  StringBuffer text = new StringBuffer();
  text.append(getBIFHeader());
  text.append("\n");
  text.append("\n");
  text.append("<BIF VERSION=\"0.3\">\n");
  text.append("<NETWORK>\n");
  text.append("<NAME>" + XMLNormalize(Utils.quote(m_Instances.relationName()))
    + "</NAME>\n");
  for (int iAttribute = 0; iAttribute < m_Instances.numAttributes(); iAttribute++) {
    text.append("<VARIABLE TYPE=\"nature\">\n");
    text.append("<NAME>"
      + XMLNormalize(Utils.quote(m_Instances.attribute(iAttribute).name())) + "</NAME>\n");
    for (int iValue = 0; iValue < m_Instances.attribute(iAttribute)
```

```
      .numValues(); iValue++) {
      text.append("<OUTCOME>"
        + XMLNormalize(Utils.quote(m_Instances.attribute(iAttribute).value(iValue)))
        + "</OUTCOME>\n");
    }
    text.append("</VARIABLE>\n");
   }

   for (int iAttribute = 0; iAttribute < m_Instances.numAttributes(); iAttribute++) {
    text.append("<DEFINITION>\n");
    text.append("<FOR>"
      + XMLNormalize(Utils.quote(m_Instances.attribute(iAttribute).name())) + "</FOR>\n");
    for (int iParent = 0; iParent < m_ParentSets[iAttribute].getNrOfParents(); iParent++) {
      text
        .append("<GIVEN>"
          + XMLNormalize(Utils.quote(m_Instances.attribute(
            m_ParentSets[iAttribute].getParent(iParent)).name()))
          + "</GIVEN>\n");
    }
    text.append("<TABLE>\n");
    for (int iParent = 0; iParent < m_ParentSets[iAttribute]
      .getCardinalityOfParents(); iParent++) {
      for (int iValue = 0; iValue < m_Instances.attribute(iAttribute)
        .numValues(); iValue++) {
        text.append(m_Distributions[iAttribute][iParent]
          .getProbability(iValue));
        text.append(' ');
      }
      text.append('\n');
    }
    text.append("</TABLE>\n");
    text.append("</DEFINITION>\n");
   }
   text.append("</NETWORK>\n");
   text.append("</BIF>\n");
   return text.toString();
  } // toXMLBIF03

  /**
   * XMLNormalize converts the five standard XML entities in a string g.e. the
   * string V&D's is returned as V&amp;D&apos;s
   *
   * @param sStr string to normalize
   * @return normalized string
   */
  protected String XMLNormalize(String sStr) {
   StringBuffer sStr2 = new StringBuffer();
   for (int iStr = 0; iStr < sStr.length(); iStr++) {
    char c = sStr.charAt(iStr);
    switch (c) {
    case '&':
      sStr2.append("&amp;");
      break;
    case '\'':
      sStr2.append("&apos;");
      break;
    case '\"':
      sStr2.append("&quot;");
```

```
      break;
    case '<':
      sStr2.append("&lt;");
      break;
    case '>':
      sStr2.append("&gt;");
      break;
    default:
      sStr2.append(c);
    }
  }
  return sStr2.toString();
} // XMLNormalize

/**
 * @return a string to describe the UseADTreeoption.
 */
public String useADTreeTipText() {
  return "When ADTree (the data structure for increasing speed on counts,"
    + " not to be confused with the classifier under the same name) is used"
    + " learning time goes down typically. However, because ADTrees are memory"
    + " intensive, memory problems may occur. Switching this option off makes"
    + " the structure learning algorithms slower, and run with less memory."
    + " By default, ADTrees are used.";
}

/**
 * @return a string to describe the SearchAlgorithm.
 */
public String searchAlgorithmTipText() {
  return "Select method used for searching network structures.";
}

/**
 * This will return a string describing the BayesNetEstimator.
 *
 * @return The string.
 */
public String estimatorTipText() {
  return "Select Estimator algorithm for finding the conditional probability tables"
    + " of the Bayes Network.";
}

/**
 * @return a string to describe the BIFFile.
 */
public String BIFFileTipText() {
  return "Set the name of a file in BIF XML format. A Bayes network learned"
    + " from data can be compared with the Bayes network represented by the BIF file."
    + " Statistics calculated are o.a. the number of missing and extra arcs.";
}

/**
 * This will return a string describing the classifier.
 *
 * @return The string.
 */
public String globalInfo() {
```

```
  return "Bayes Network learning using various search algorithms and "
    + "quality measures.\n"
    + "Base class for a Bayes Network classifier. Provides "
    + "datastructures (network structure, conditional probability "
    + "distributions, etc.) and facilities common to Bayes Network "
    + "learning algorithms like K2 and B.\n\n"
    + "For more information see:\n\n"
    + "http://www.cs.waikato.ac.nz/~remco/weka.pdf";
}

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

/**
 * get name of the Bayes network
 *
 * @return name of the Bayes net
 */
public String getName() {
  return m_Instances.relationName();
}

/**
 * get number of nodes in the Bayes network
 *
 * @return number of nodes
 */
public int getNrOfNodes() {
  return m_Instances.numAttributes();
}

/**
 * get name of a node in the Bayes network
 *
 * @param iNode index of the node
 * @return name of the specified node
 */
public String getNodeName(int iNode) {
  return m_Instances.attribute(iNode).name();
}

/**
 * get number of values a node can take
 *
 * @param iNode index of the node
 * @return cardinality of the specified node
 */
public int getCardinality(int iNode) {
  return m_Instances.attribute(iNode).numValues();
}

/**
```

```
 * get name of a particular value of a node
 *
 * @param iNode index of the node
 * @param iValue index of the value
 * @return cardinality of the specified node
 */
public String getNodeValue(int iNode, int iValue) {
  return m_Instances.attribute(iNode).value(iValue);
}

/**
 * get number of parents of a node in the network structure
 *
 * @param iNode index of the node
 * @return number of parents of the specified node
 */
public int getNrOfParents(int iNode) {
  return m_ParentSets[iNode].getNrOfParents();
}

/**
 * get node index of a parent of a node in the network structure
 *
 * @param iNode index of the node
 * @param iParent index of the parents, e.g., 0 is the first parent, 1 the
 *          second parent, etc.
 * @return node index of the iParent's parent of the specified node
 */
public int getParent(int iNode, int iParent) {
  return m_ParentSets[iNode].getParent(iParent);
}

/**
 * Get full set of parent sets.
 *
 * @return parent sets;
 */
public ParentSet[] getParentSets() {
  return m_ParentSets;
}

/**
 * Get full set of estimators.
 *
 * @return estimators;
 */
public Estimator[][] getDistributions() {
  return m_Distributions;
}

/**
 * get number of values the collection of parents of a node can take
 *
 * @param iNode index of the node
 * @return cardinality of the parent set of the specified node
 */
public int getParentCardinality(int iNode) {
  return m_ParentSets[iNode].getCardinalityOfParents();
```

```
}

/**
 * get particular probability of the conditional probability distribtion of a
 * node given its parents.
 *
 * @param iNode index of the node
 * @param iParent index of the parent set, 0 <= iParent <=
 *        getParentCardinality(iNode)
 * @param iValue index of the value, 0 <= iValue <= getCardinality(iNode)
 * @return probability
 */
public double getProbability(int iNode, int iParent, int iValue) {
  return m_Distributions[iNode][iParent].getProbability(iValue);
}

/**
 * get the parent set of a node
 *
 * @param iNode index of the node
 * @return Parent set of the specified node.
 */
public ParentSet getParentSet(int iNode) {
  return m_ParentSets[iNode];
}

/**
 * get ADTree strucrture containing efficient representation of counts.
 *
 * @return ADTree strucrture
 */
public ADNode getADTree() {
  return m_ADTree;
}

// implementation of AdditionalMeasureProducer interface
/**
 * Returns an enumeration of the measure names. Additional measures must
 * follow the naming convention of starting with "measure", eg. double
 * measureBlah()
 *
 * @return an enumeration of the measure names
 */
@Override
public Enumeration<String> enumerateMeasures() {
  Vector<String> newVector = new Vector<String>(4);
  newVector.addElement("measureExtraArcs");
  newVector.addElement("measureMissingArcs");
  newVector.addElement("measureReversedArcs");
  newVector.addElement("measureDivergence");
  newVector.addElement("measureBayesScore");
  newVector.addElement("measureBDeuScore");
  newVector.addElement("measureMDLScore");
  newVector.addElement("measureAICScore");
  newVector.addElement("measureEntropyScore");
  return newVector.elements();
} // enumerateMeasures
```

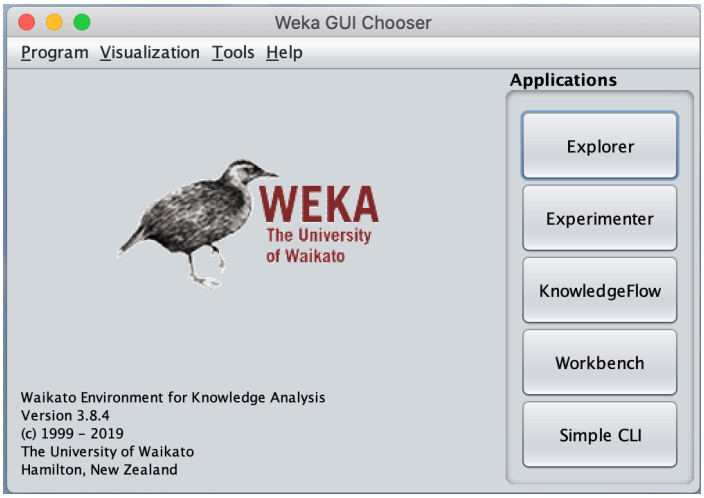
```
public double measureExtraArcs() {
  if (m_otherBayesNet != null) {
    return m_otherBayesNet.extraArcs(this);
  }
  return 0;
} // measureExtraArcs

public double measureMissingArcs() {
  if (m_otherBayesNet != null) {
    return m_otherBayesNet.missingArcs(this);
  }
  return 0;
} // measureMissingArcs

public double measureReversedArcs() {
  if (m_otherBayesNet != null) {
    return m_otherBayesNet.reversedArcs(this);
  }
  return 0;
} // measureReversedArcs

public double measureDivergence() {
  if (m_otherBayesNet != null) {
    return m_otherBayesNet.divergence(this);
  }
  return 0;
} // measureDivergence

public double measureBayesScore() {
  try {
    LocalScoreSearchAlgorithm s = new LocalScoreSearchAlgorithm(this,
                                          m_Instances);
    return s.logScore(Scoreable.BAYES);
  } catch (ArithmeticException ex) {
    return Double.NaN;
  }
} // measureBayesScore

public double measureBDeuScore() {

  try {
    LocalScoreSearchAlgorithm s = new LocalScoreSearchAlgorithm(this,
                                          m_Instances);
    return s.logScore(Scoreable.BDeu);
  } catch (ArithmeticException ex) {
    return Double.NaN;
  }
} // measureBDeuScore

public double measureMDLScore() {

  try {
    LocalScoreSearchAlgorithm s = new LocalScoreSearchAlgorithm(this,
                                          m_Instances);
    return s.logScore(Scoreable.MDL);
  } catch (ArithmeticException ex) {
    return Double.NaN;
  }
```

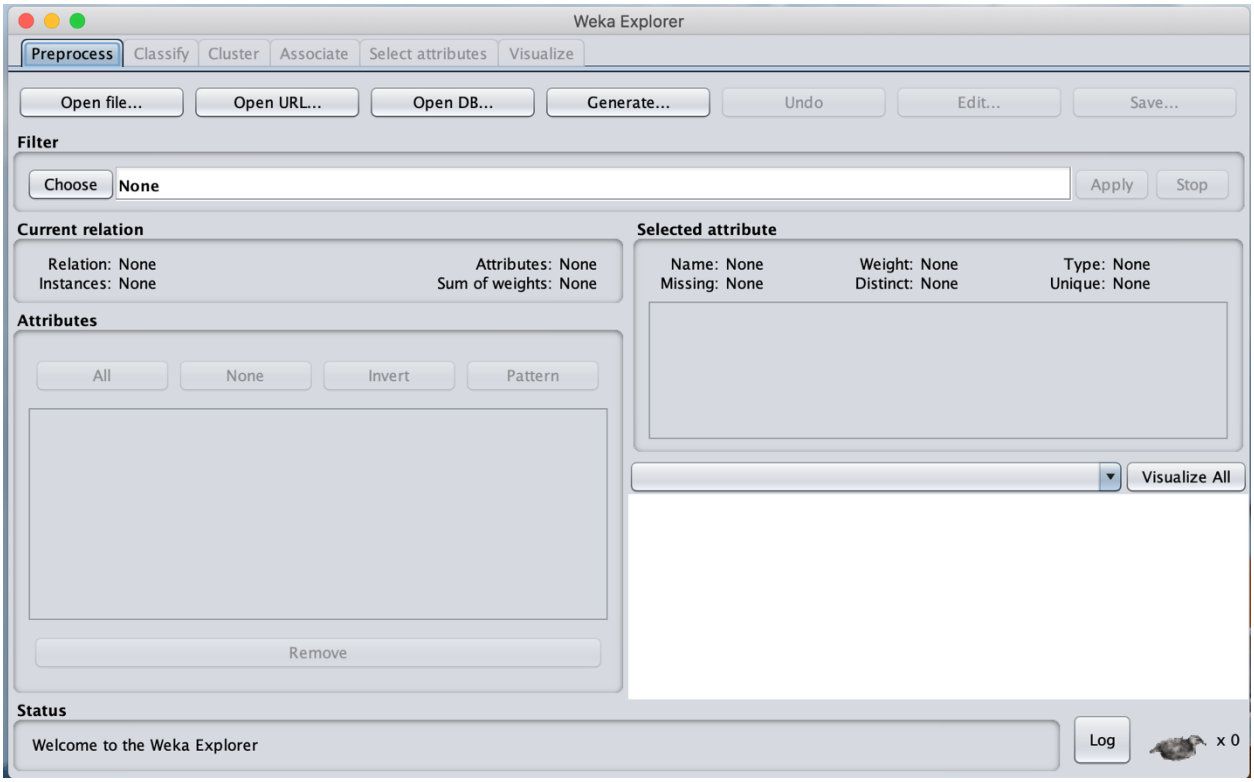
```
} // measureMDLScore

public double measureAICScore() {

 try {
  LocalScoreSearchAlgorithm s = new LocalScoreSearchAlgorithm(this,
                                        m_Instances);
  return s.logScore(Scoreable.AIC);
 } catch (ArithmeticException ex) {
  return Double.NaN;
 }
} // measureAICScore

public double measureEntropyScore() {

 try {
  LocalScoreSearchAlgorithm s = new LocalScoreSearchAlgorithm(this,
                                        m_Instances);
  return s.logScore(Scoreable.ENTROPY);
 } catch (ArithmeticException ex) {
  return Double.NaN;
 }
} // measureEntropyScore

/**
 * Returns the value of the named measure
 *
 * @param measureName the name of the measure to query for its value
 * @return the value of the named measure
 * @throws IllegalArgumentException if the named measure is not supported
 */
@Override
public double getMeasure(String measureName) {
 if (measureName.equals("measureExtraArcs")) {
  return measureExtraArcs();
 }
 if (measureName.equals("measureMissingArcs")) {
  return measureMissingArcs();
 }
 if (measureName.equals("measureReversedArcs")) {
  return measureReversedArcs();
 }
 if (measureName.equals("measureDivergence")) {
  return measureDivergence();
 }
 if (measureName.equals("measureBayesScore")) {
  return measureBayesScore();
 }
 if (measureName.equals("measureBDeuScore")) {
  return measureBDeuScore();
 }
 if (measureName.equals("measureMDLScore")) {
  return measureMDLScore();
 }
 if (measureName.equals("measureAICScore")) {
  return measureAICScore();
 }
 if (measureName.equals("measureEntropyScore")) {
```

```
    return measureEntropyScore();
   }
   return 0;
  } // getMeasure

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

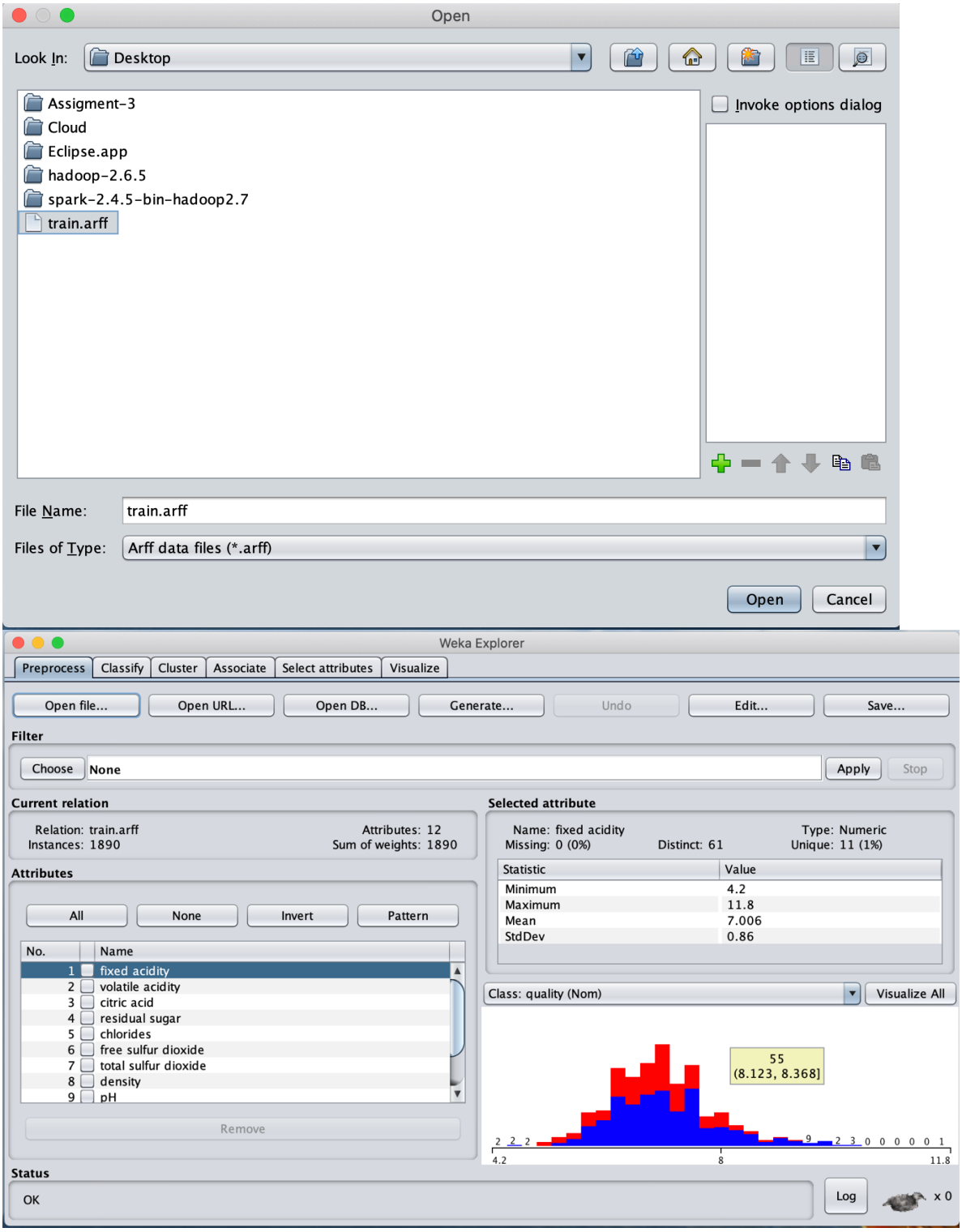
# Implementation of Naïve Bayes

1.Open Weka
2.Weka > Weka Explorer

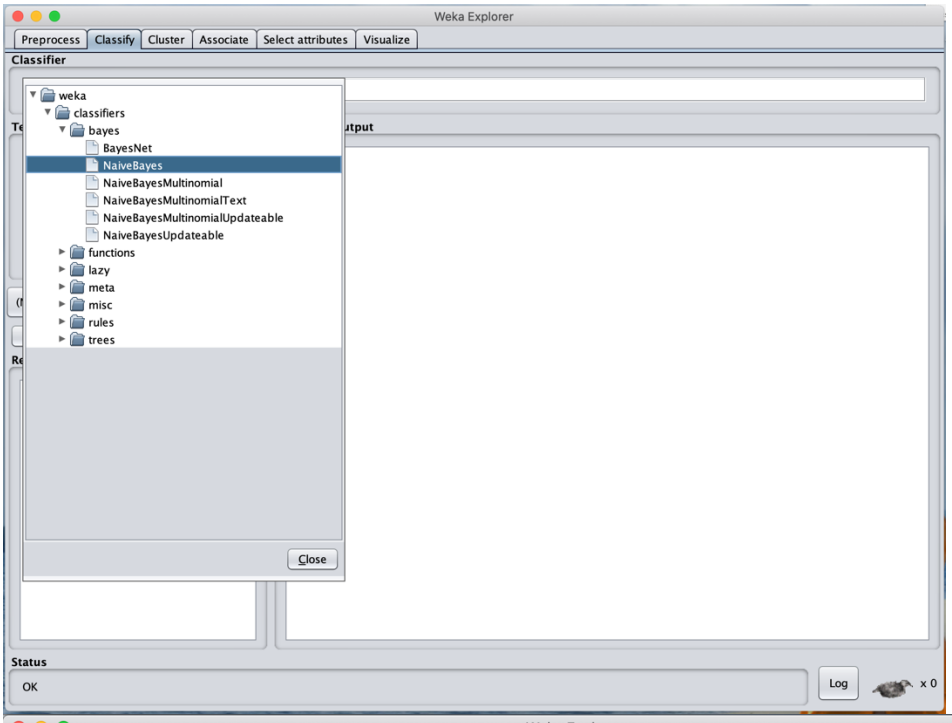


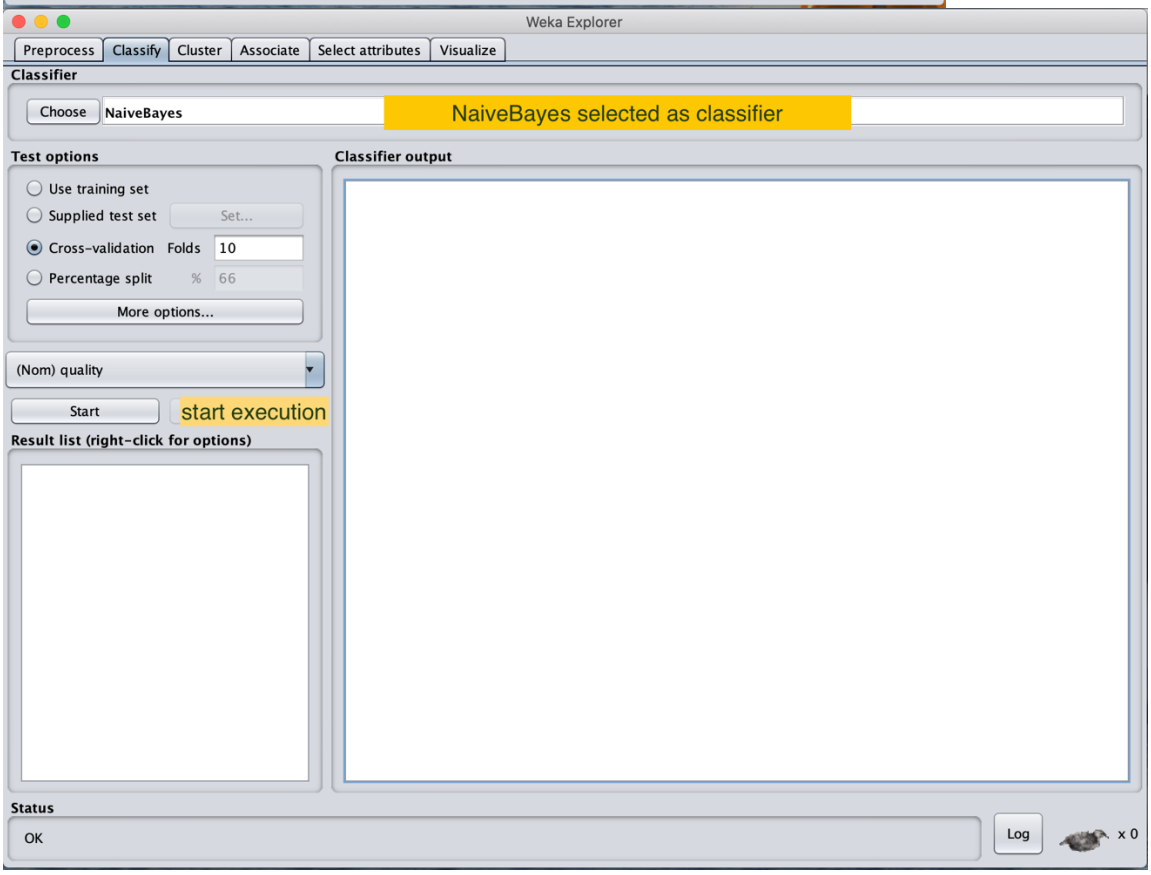



3.Open File > Select .arff dataset file > Open
4. You will see the dataset and its attributes opened in window.

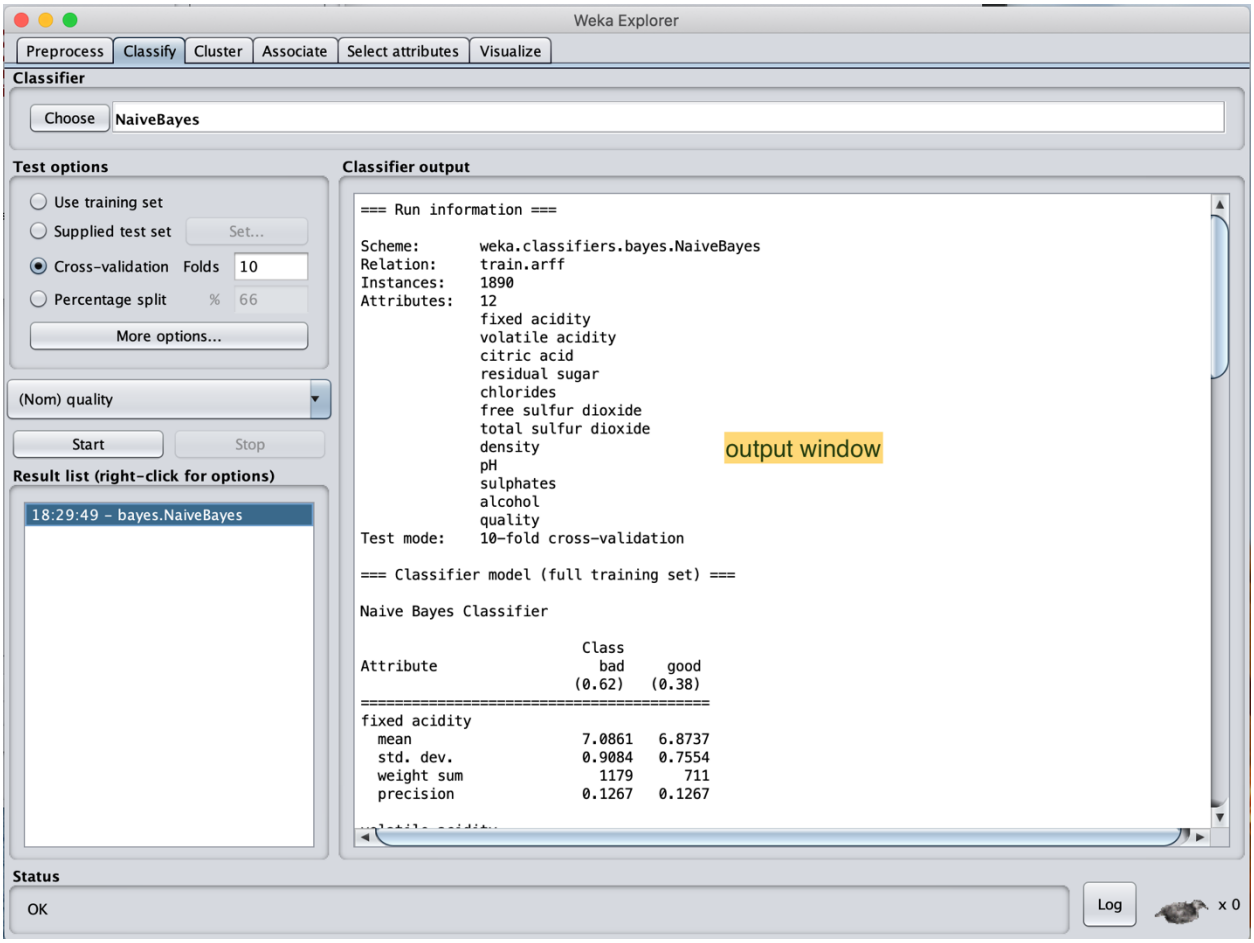


5. select classify > classifiers > bayes > NaiveBayes
6. You will see the NaiveBayes selected in choose window. > Press start to execute.



7. You will see the output in the output window on the right side.




## Conclusion :

From the above screenshots, we clearly see that the accuracy of NaiveBayes for the given dataset is 78.88%.

# Output of NaïveBayes

```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     train.arff
Instances:    1890
Attributes:   12
              fixed acidity
              volatile acidity
              citric acid
              residual sugar
              chlorides
              free sulfur dioxide
              total sulfur dioxide
              density
              pH
              sulphates
              alcohol
              quality
Test mode:    10-fold cross-validation

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

Naive Bayes Classifier

                        Class
Attribute                 bad     good
                        (0.62)   (0.38)
=========================================
fixed acidity
  mean                  7.0861   6.8737
  std. dev.             0.9084   0.7554
  weight sum              1179      711
  precision             0.1267   0.1267

volatile acidity
  mean                  0.3081   0.2641
  std. dev.             0.1145   0.0925
  weight sum              1179      711
  precision             0.0096   0.0096

citric acid
  mean                  0.3461   0.3396
  std. dev.             0.1467   0.0837
  weight sum              1179      711
  precision             0.0125   0.0125
```

residual sugar

| | | |
|---|---|---|
| mean | 7.1042 | 5.1344 |
| std. dev. | 5.3507 | 4.2357 |
| weight sum | 1179 | 711 |
| precision | 0.0966 | 0.0966 |

chlorides

| | | |
|---|---|---|
| mean | 0.0509 | 0.0381 |
| std. dev. | 0.0274 | 0.0121 |
| weight sum | 1179 | 711 |
| precision | 0.0029 | 0.0029 |

free sulfur dioxide

| | | |
|---|---|---|
| mean | 34.9411 | 34.4641 |
| std. dev. | 19.3755 | 13.6684 |
| weight sum | 1179 | 711 |
| precision | 1.3798 | 1.3798 |

total sulfur dioxide

| | | |
|---|---|---|
| mean | 150.0791 | 127.9552 |
| std. dev. | 47.4217 | 34.8526 |
| weight sum | 1179 | 711 |
| precision | 1.5653 | 1.5653 |

density

| | | |
|---|---|---|
| mean | 0.9954 | 0.9927 |
| std. dev. | 0.0025 | 0.0027 |
| weight sum | 1179 | 711 |
| precision | 0 | 0 |

pH

| | | |
|---|---|---|
| mean | 3.1763 | 3.2344 |
| std. dev. | 0.1494 | 0.1563 |
| weight sum | 1179 | 711 |
| precision | 0.0114 | 0.0114 |

sulphates

| | | |
|---|---|---|
| mean | 0.4783 | 0.5026 |
| std. dev. | 0.1044 | 0.1344 |
| weight sum | 1179 | 711 |
| precision | 0.011 | 0.011 |

alcohol

| | | |
|---|---|---|
| mean | 9.79 | 11.2754 |
| std. dev. | 0.8431 | 1.1893 |
| weight sum | 1179 | 711 |
| precision | 0.1111 | 0.1111 |

Time taken to build model: 0.05 seconds

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

| | | |
|---|---|---|
| Correctly Classified Instances | 1491 | 78.8889 % |
| Incorrectly Classified Instances | 399 | 21.1111 % |
| Kappa statistic | 0.5668 | |
| Mean absolute error | 0.2354 | |
| Root mean squared error | 0.4145 | |
| Relative absolute error | 50.158 % | |
| Root relative squared error | 85.5744 % | |
| Total Number of Instances | 1890 | |

=== Detailed Accuracy By Class ===

| | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area | Class |
|---|---|---|---|---|---|---|---|---|---|
| | 0.773 | 0.184 | 0.874 | 0.773 | 0.820 | 0.573 | 0.855 | 0.882 | bad |
| | 0.816 | 0.227 | 0.684 | 0.816 | 0.744 | 0.573 | 0.855 | 0.813 | good |
| Weighted Avg. | 0.789 | 0.200 | 0.803 | 0.789 | 0.792 | 0.573 | 0.855 | 0.856 | |

=== Confusion Matrix ===

| a | b | <-- classified as |
|---|---|---|
| 911 | 268 | a = bad |
| 131 | 580 | b = good |

# Source Code of Naïve Bayes

```
/*
 *   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() {
```

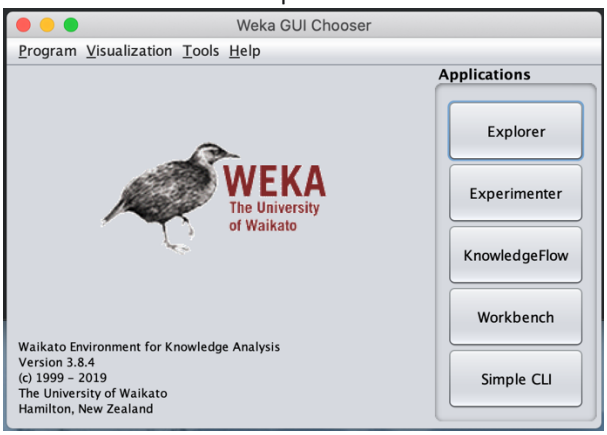
```
 return m_Instances;
}

/**
 * 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++) {
```

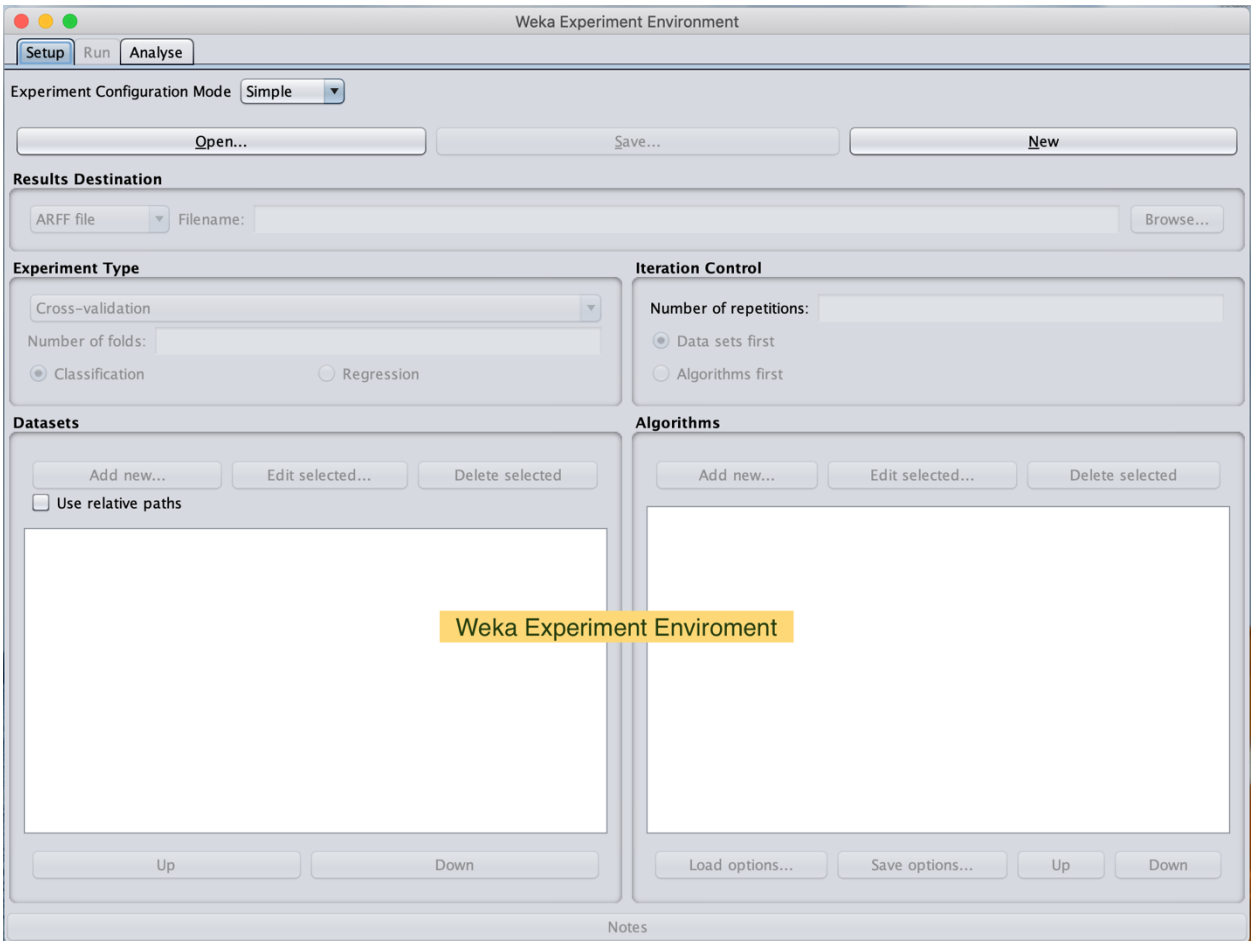
```
      ((Aggregateable) m_Distributions[i][j])
        .aggregate(toAggregate.m_Distributions[i][j]);
    }
   }

   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);
  }
}
```

# Comparative Implementation of Weka BayeNet and Naïve Bayes

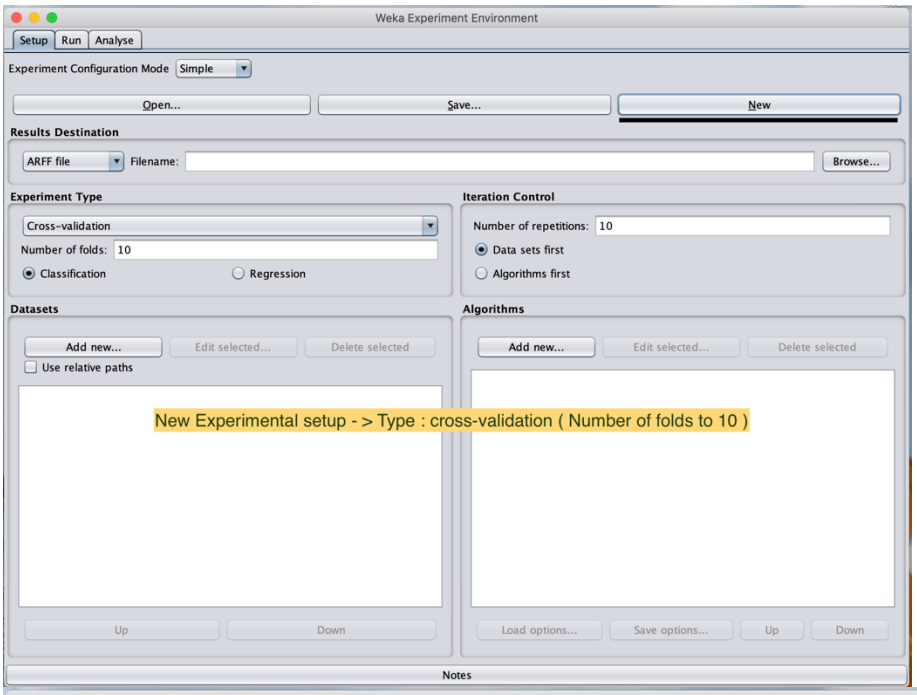
1.Open Weka

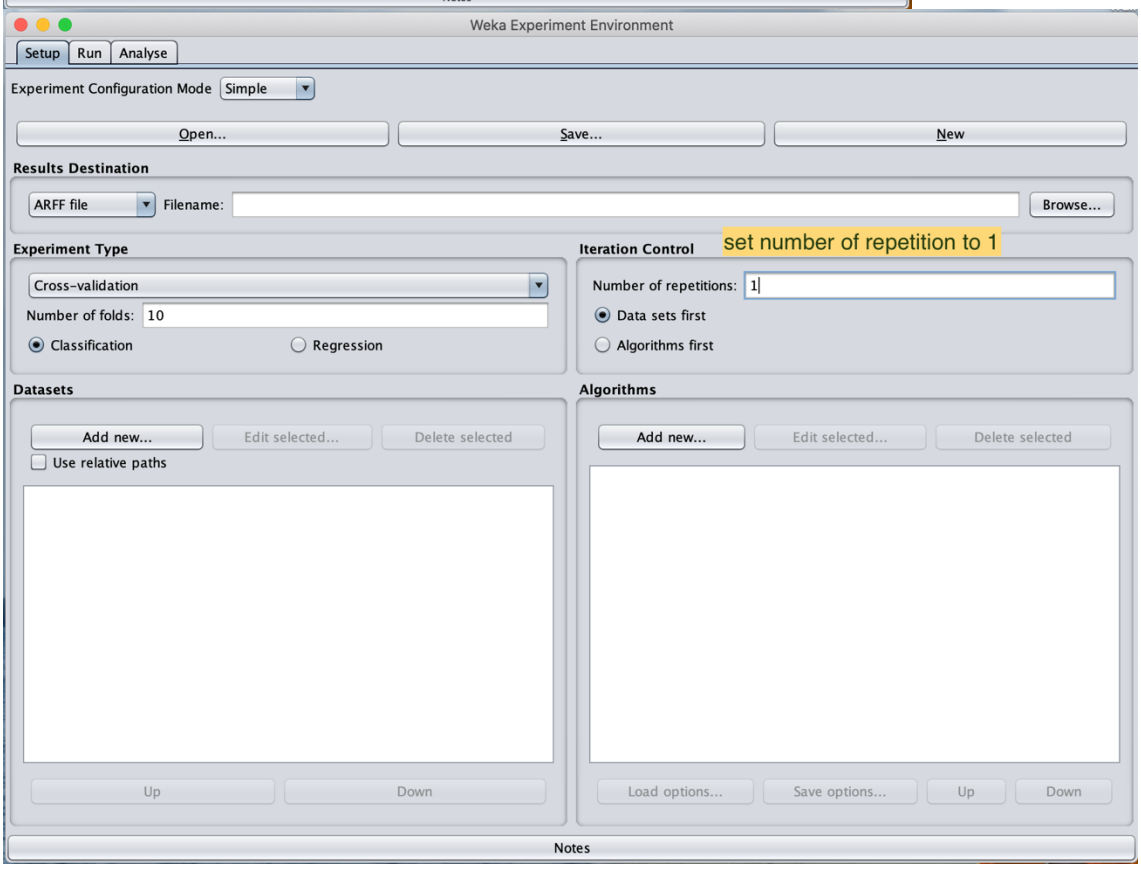
2.Weka > Weka Experimenter



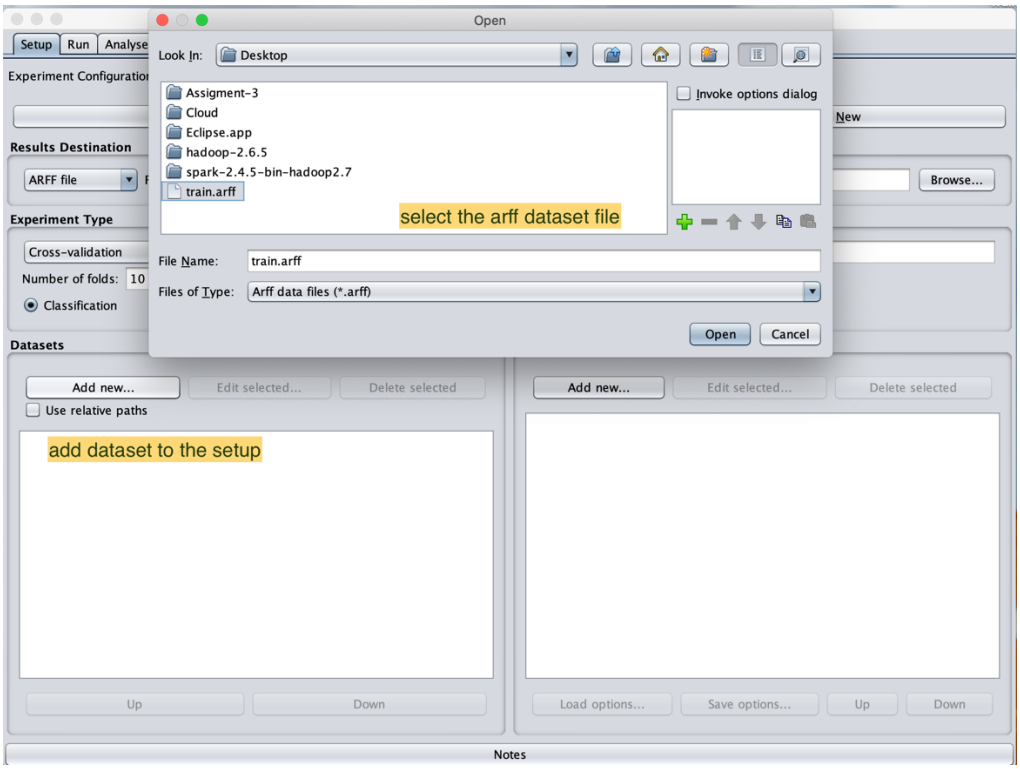



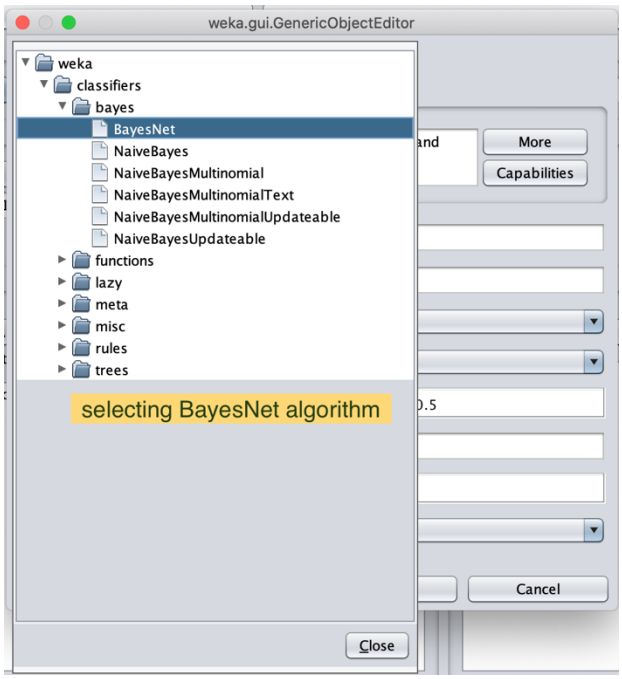
3.Click New from the Options in top of the window.
4.Experimental Tyes > Cross validation > Number of folds : 10
5.Number of repetations : 1

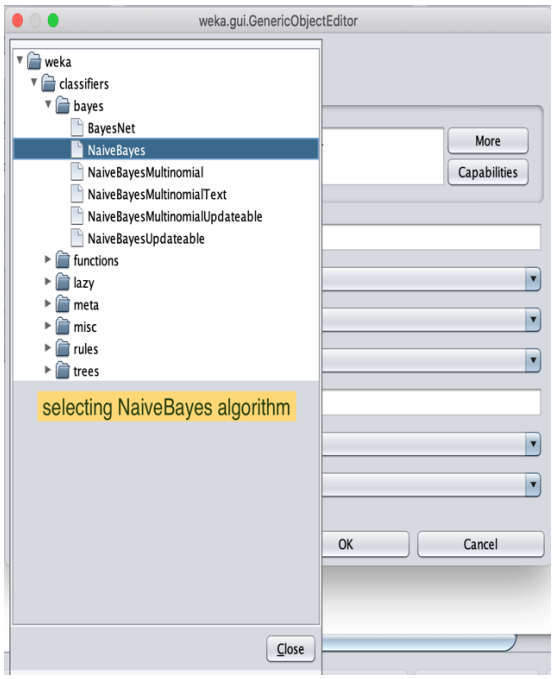


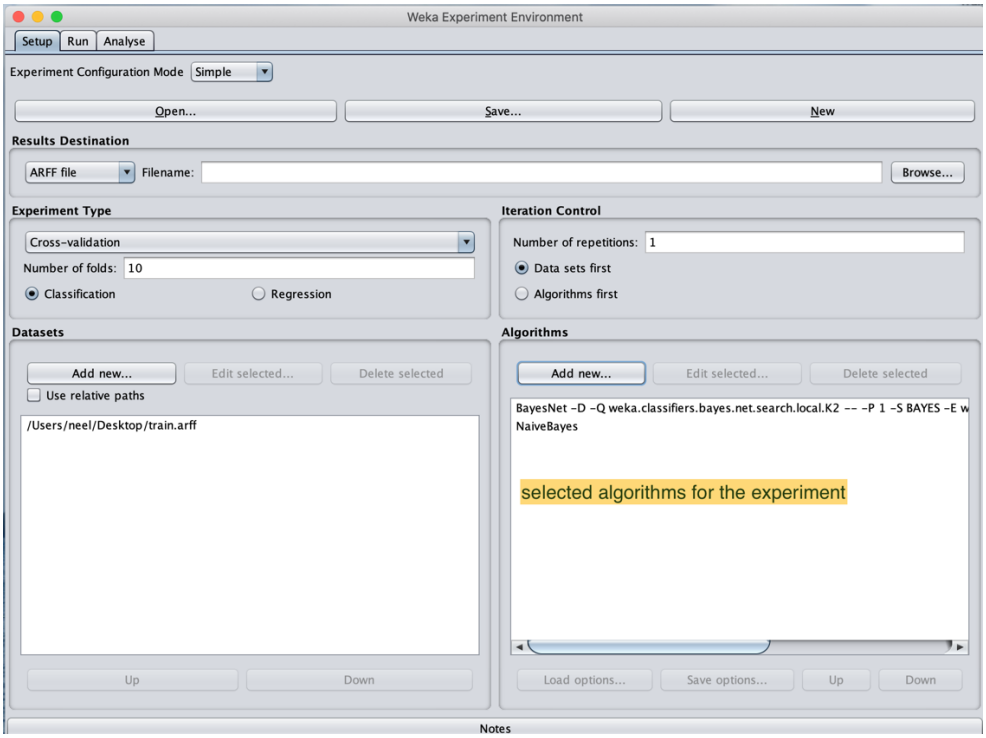
6. In Datasets > add new > select arff file > open.
7. In Algorithms > add new > select algorithm > Ok

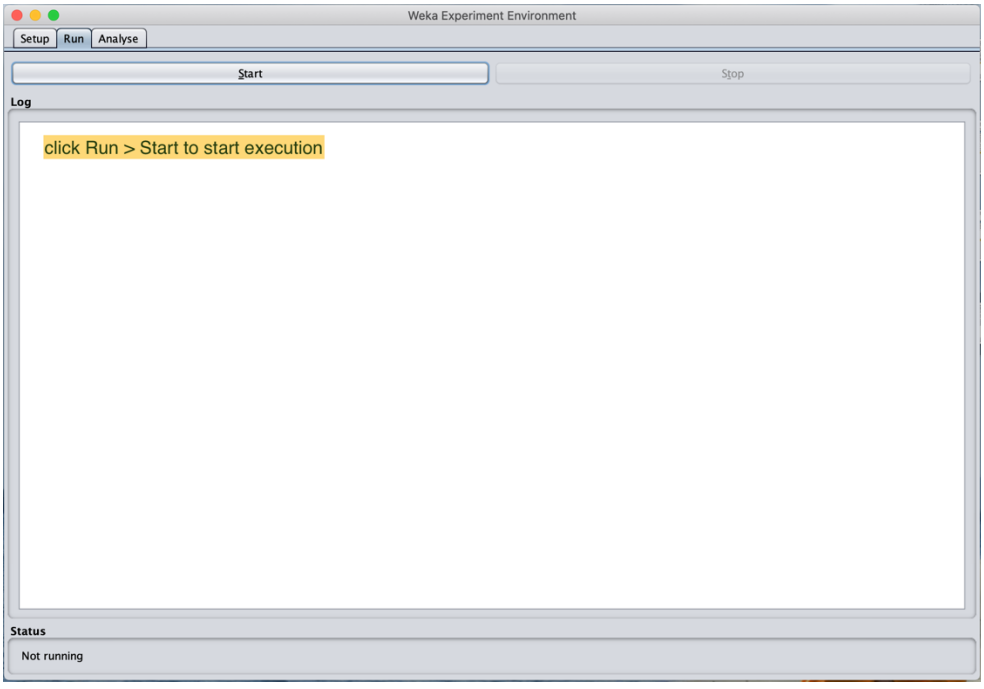


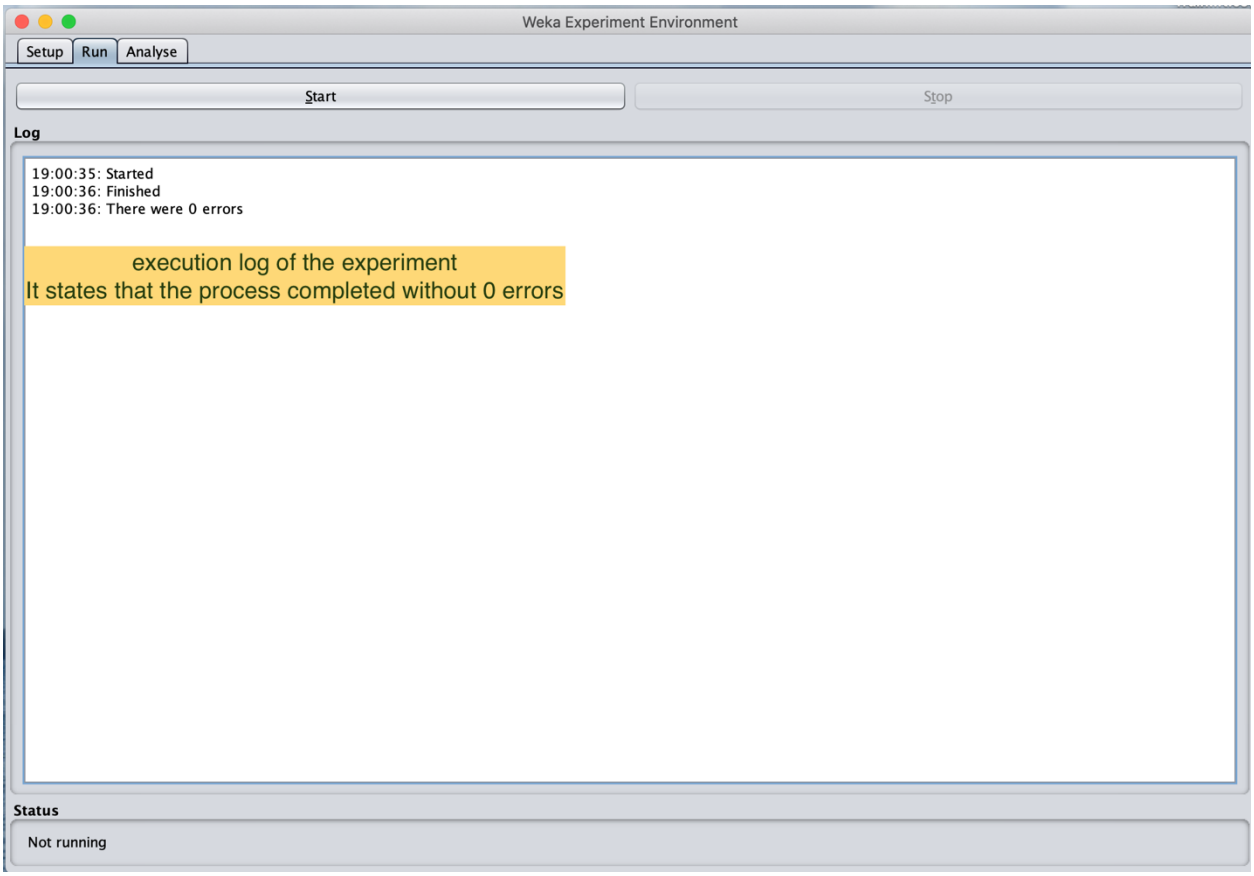



8.selected algorithms will be shown in algorithms field.
9.After completing setup, click Run from the top toolbar > click start to execute.

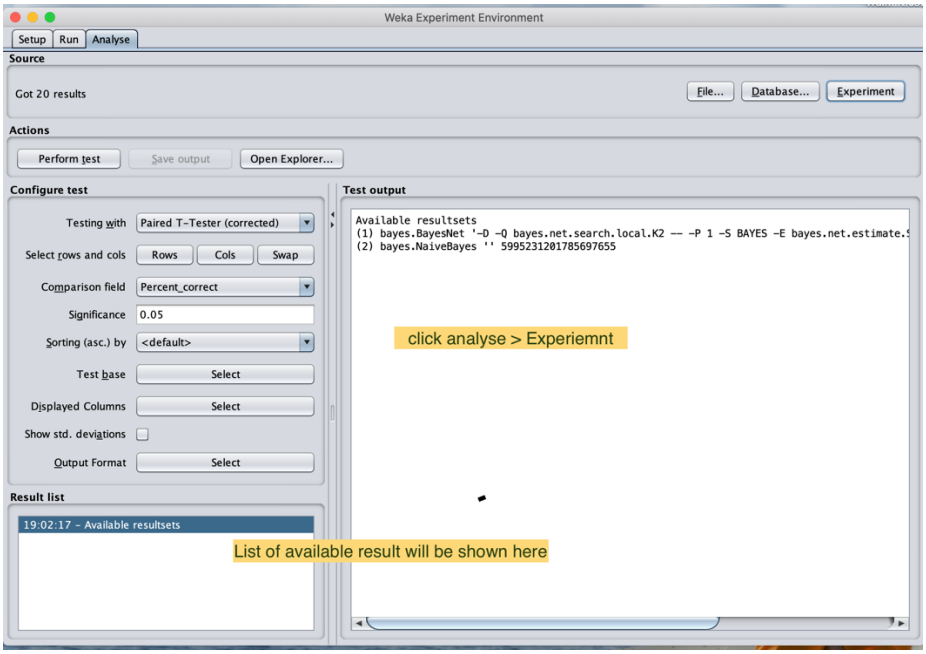


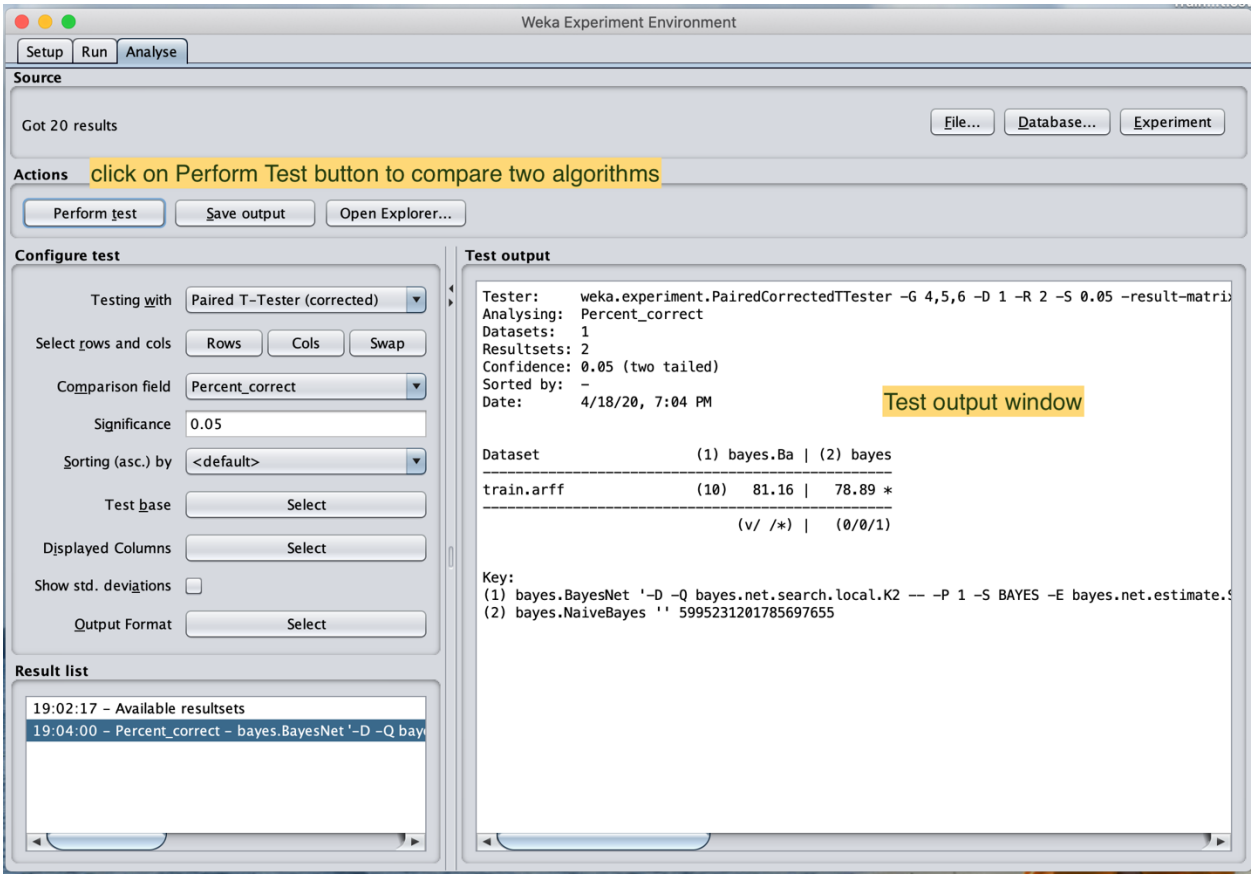



10.Activities of the experiment executing is shown in log.



11.Click Analyse from top tollbar > Experiment > List of results will be shown.
12.Click perform test to see comparative result in Test Output window.



## Conclusion :

From the above screenshots, we clearly see that the accuracy of Naïve Bayes( built time : 0.05) & Weka BayesNet( built time : 0.15) for the given dataset is 78.88% & 81.16% respectively. Thus BayesNet performed better in accuracy than NaiveBayes for given dataset.

# References


- https://archive.ics.uci.edu/ml/machine-learning-databases/00475/
- https://machinelearningmastery.com/standard-machine-learning-datasets/
- https://en.wikipedia.org/wiki/Bayesian_network
- https://towardsdatascience.com/introduction-to-bayesian-networks-81031eeed94e

# Learning Outcome

- Learned about Weka Software.
- Various Algorithms about classification.
- Bayesian Network & Naïve Bayes.
- UCI datasets and their attributes.