**FINAL TERM PROJECT**

**OPTION 1 SUPERVISED DATA MINING (CLASSIFICATION)**

**Data Mining - CS 634**

**Instructor: Jason T. Wang (P)**

**Submitted By: Darshan A Patel**

**UCID: dap83@njit.edu**

**NJIT ID: 31415244**

**Table of Contents Page**

1) Introduction…………………………………………………………………………...3

2) Tools Used……………………………………………………………………………3

3) Dataset Used…………………………………………………………………………3

4) Operating System Used…………………………………………………………….3

5) Programming Language Used……………………………………………………..3

6) Hardware Used………………………………………………………………………3

7) Algorithms Used……………………………………………………………………..4

**Naïve Bayes:**

7.1.1) Naïve Bayes Algorithms introduction………………………………… 4

7.1.2) Naïve Bayes (Source code)…………………………………………... 5

7.1.3) Classification of Naïve Bayes Algorithm…………………………….. 24

7.1.4) Output of Classification of Naïve Bayes Algorithm………………….26

**Random Forest:**

7.2.1) Random Forest Algorithm introduction……………………………….30

7.2.2) Random Forest (Source code)………………………………………..31

7.2.3) Classification of Random Forest Algorithm……………………….....43

7.2.4) Output of Classification of Random Forest Algorithm………………44

8) Comparison of both algorithms/Output…………………………………………...46

9) Multiple Roc curve for Naïve Bayes and Random Forest………………………53

10) Conclusion………………………………………………………………………….60

11) Links…………………………………………………………………………………60

**1) Introduction:**

An aim of this project is to implement two algorithms using same dataset and compare accuracy between both algorithms.

**2) Tool used:**

WEKA - <http://www.cs.waikato.ac.nz/ml/weka/downloading.html>

**3) Dataset used:**

<https://archive.ics.uci.edu/ml/machine-learning-databases/00372/>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Set Characteristics:** | Multivariate | **Number of Instances:** | 17898 | **Area:** | Physical |
| **Attribute Characteristics:** | Real | **Number of Attributes:** | 9 | **Date Donated** | 2017-02-14 |
| **Associated Tasks:** | Classification, Clustering | **Missing Values?** | N/A | **Number of Web Hits:** | 5979 |

**Source:**

Dr. Robert Lyon, University of Manchester, School of Physics and Astronomy, Alan Turing Building, Manchester M13 9PL, United Kingdom, robert.lyon '@'anchester.ac.uk

**4) Operating System Used:**

* Windows 10

**5) Programming language used:**

* Java

**6) Hardware Used:**

* **Processor:** Intel(R) Core(TM) i3-3227U CPU @1.90GHz
* **RAM:** 8.00 GB
* **System Type**: 64-bit OS, x64-based processor

**7.1.1) Naïve Bayes Algorithms introduction**

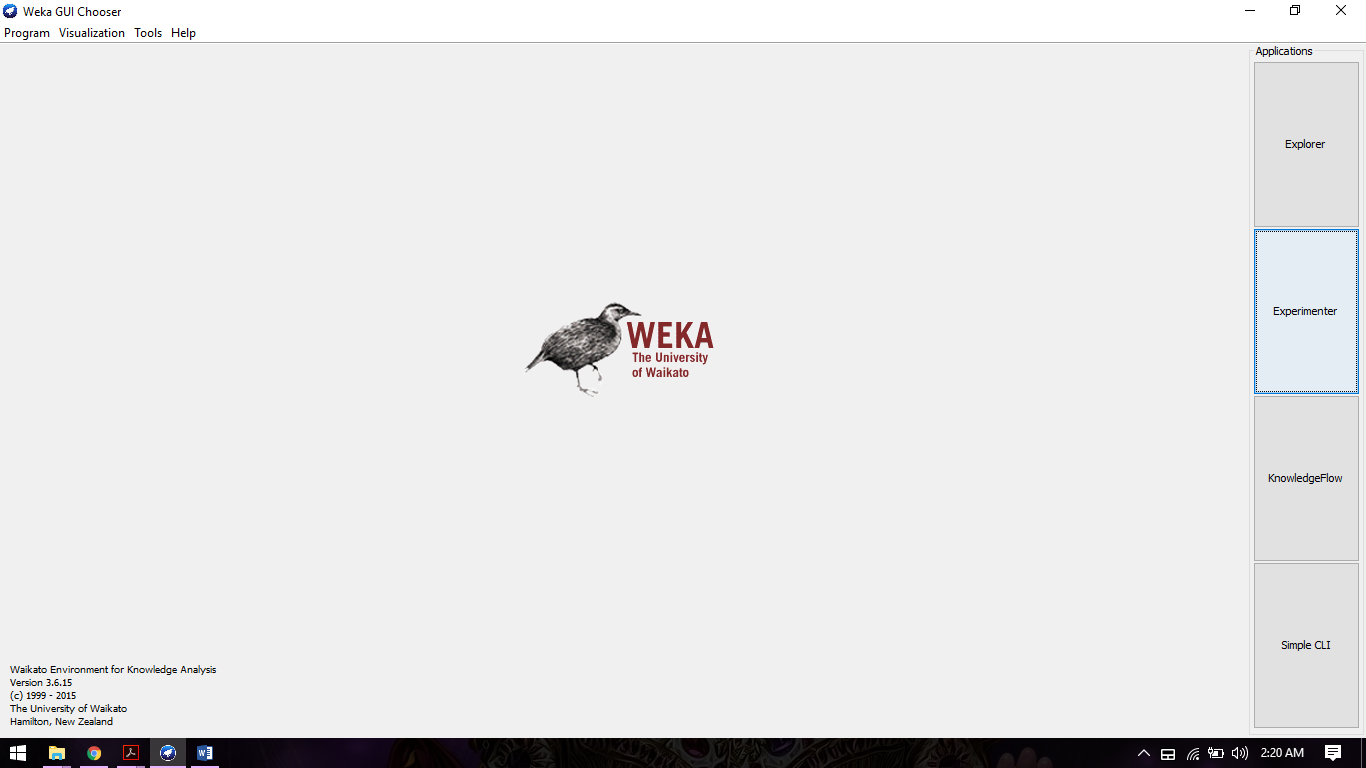
Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

|  |
| --- |
| **7.1.2) Naïve Bayes(Source Code)**  /\*  \* NaiveBayes.java  \* Copyright (C) 1999 University of Waikato, Hamilton, New Zealand  \*  \*/  package weka.classifiers.bayes;  import weka.classifiers.Classifier;  import weka.core.Attribute;  import weka.core.Capabilities;  import weka.core.Instance;  import weka.core.Instances;  import weka.core.Option;  import weka.core.OptionHandler;  import weka.core.RevisionUtils;  import weka.core.TechnicalInformation;  import weka.core.TechnicalInformationHandler;  import weka.core.Utils;  import weka.core.WeightedInstancesHandler;  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;  import java.util.Enumeration;  import java.util.Vector;  /\*\*  <!-- 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: 5516 $  \*/  public class NaiveBayes extends Classifier  implements OptionHandler, WeightedInstancesHandler,  TechnicalInformationHandler {  /\*\* 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  \*/  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  \*/  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  \*/  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 enu = m\_Instances.enumerateAttributes();  while (enu.hasMoreElements()) {  Attribute attribute = (Attribute) enu.nextElement();  // If the attribute is numeric, determine the estimator  // numeric precision from differences between adjacent values  double numPrecision = DEFAULT\_NUM\_PRECISION;  if (attribute.type() == Attribute.NUMERIC) {  m\_Instances.sort(attribute);  if ((m\_Instances.numInstances() > 0)  && !m\_Instances.instance(0).isMissing(attribute)) {  double lastVal = m\_Instances.instance(0).value(attribute);  double currentVal, deltaSum = 0;  int distinct = 0;  for (int i = 1; i < m\_Instances.numInstances(); i++) {  Instance currentInst = m\_Instances.instance(i);  if (currentInst.isMissing(attribute)) {  break;  }  currentVal = currentInst.value(attribute);  if (currentVal != lastVal) {  deltaSum += currentVal - lastVal;  lastVal = currentVal;  distinct++;  }  }  if (distinct > 0) {  numPrecision = deltaSum / distinct;  }  }  }  for (int j = 0; j < m\_Instances.numClasses(); j++) {  switch (attribute.type()) {  case Attribute.NUMERIC:  if (m\_UseKernelEstimator) {  m\_Distributions[attIndex][j] =  new KernelEstimator(numPrecision);  } else {  m\_Distributions[attIndex][j] =  new NormalEstimator(numPrecision);  }  break;  case Attribute.NOMINAL:  m\_Distributions[attIndex][j] =  new DiscreteEstimator(attribute.numValues(), true);  break;  default:  throw new Exception("Attribute type unknown to NaiveBayes");  }  }  attIndex++;  }  // Compute counts  Enumeration enumInsts = m\_Instances.enumerateInstances();  while (enumInsts.hasMoreElements()) {  Instance instance =  (Instance) enumInsts.nextElement();  updateClassifier(instance);  }  // Save space  m\_Instances = new Instances(m\_Instances, 0);  }  /\*\*  \* Updates the classifier with the given instance.  \*  \* @param instance the new training instance to include in the model  \* @exception Exception if the instance could not be incorporated in  \* the model.  \*/  public void updateClassifier(Instance instance) throws Exception {  if (!instance.classIsMissing()) {  Enumeration enumAtts = m\_Instances.enumerateAttributes();  int attIndex = 0;  while (enumAtts.hasMoreElements()) {  Attribute attribute = (Attribute) enumAtts.nextElement();  if (!instance.isMissing(attribute)) {  m\_Distributions[attIndex][(int)instance.classValue()].  addValue(instance.value(attribute), instance.weight());  }  attIndex++;  }  m\_ClassDistribution.addValue(instance.classValue(),  instance.weight());  }  }  /\*\*  \* Calculates the class membership probabilities for the given test  \* instance.  \*  \* @param instance the instance to be classified  \* @return predicted class probability distribution  \* @exception Exception if there is a problem generating the prediction  \*/  public double [] distributionForInstance(Instance instance)  throws Exception {  if (m\_UseDiscretization) {  m\_Disc.input(instance);  instance = m\_Disc.output();  }  double [] probs = new double[m\_NumClasses];  for (int j = 0; j < m\_NumClasses; j++) {  probs[j] = m\_ClassDistribution.getProbability(j);  }  Enumeration enumAtts = instance.enumerateAttributes();  int attIndex = 0;  while (enumAtts.hasMoreElements()) {  Attribute attribute = (Attribute) enumAtts.nextElement();  if (!instance.isMissing(attribute)) {  double temp, max = 0;  for (int j = 0; j < m\_NumClasses; j++) {  temp = Math.max(1e-75, 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.  \*/  public Enumeration listOptions() {  Vector newVector = new Vector(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"));  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  \*/  public void setOptions(String[] options) throws Exception {  boolean k = Utils.getFlag('K', options);  boolean d = Utils.getFlag('D', options);  if (k && d) {  throw new IllegalArgumentException("Can't use both kernel density " +  "estimation and discretization!");  }  setUseSupervisedDiscretization(d);  setUseKernelEstimator(k);  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  \*/  public String [] getOptions() {  String [] options = new String [3];  int current = 0;  if (m\_UseKernelEstimator) {  options[current++] = "-K";  }  if (m\_UseDiscretization) {  options[current++] = "-D";  }  if (m\_displayModelInOldFormat) {  options[current++] = "-O";  }  while (current < options.length) {  options[current++] = "";  }  return options;  }  /\*\*  \* Returns a description of the classifier.  \*  \* @return a description of the classifier as a string.  \*/  public String toString() {  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 (int i = 0; i < m\_Distributions.length; i++) {  for (int j = 0; j < m\_Instances.numClasses(); j++) {  if (m\_Distributions[i][0] instanceof NormalEstimator) {  // check mean/precision dev against maxWidth  NormalEstimator n = (NormalEstimator)m\_Distributions[i][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\_Distributions[i][0] instanceof KernelEstimator) {  containsKernel = true;  KernelEstimator ke = (KernelEstimator)m\_Distributions[i][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\_Distributions[i][0] instanceof DiscreteEstimator) {  DiscreteEstimator d = (DiscreteEstimator)m\_Distributions[i][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 enumAtts = m\_Instances.enumerateAttributes();  int attIndex = 0;  while (enumAtts.hasMoreElements()) {  Attribute 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;  }  /\*\*  \* Returns the revision string.  \*  \* @return the revision  \*/  public String getRevision() {  return RevisionUtils.extract("$Revision: 5516 $");  }  /\*\*  \* Main method for testing this class.  \*  \* @param argv the options  \*/  public static void main(String [] argv) {  runClassifier(new NaiveBayes(), argv);  }  } |

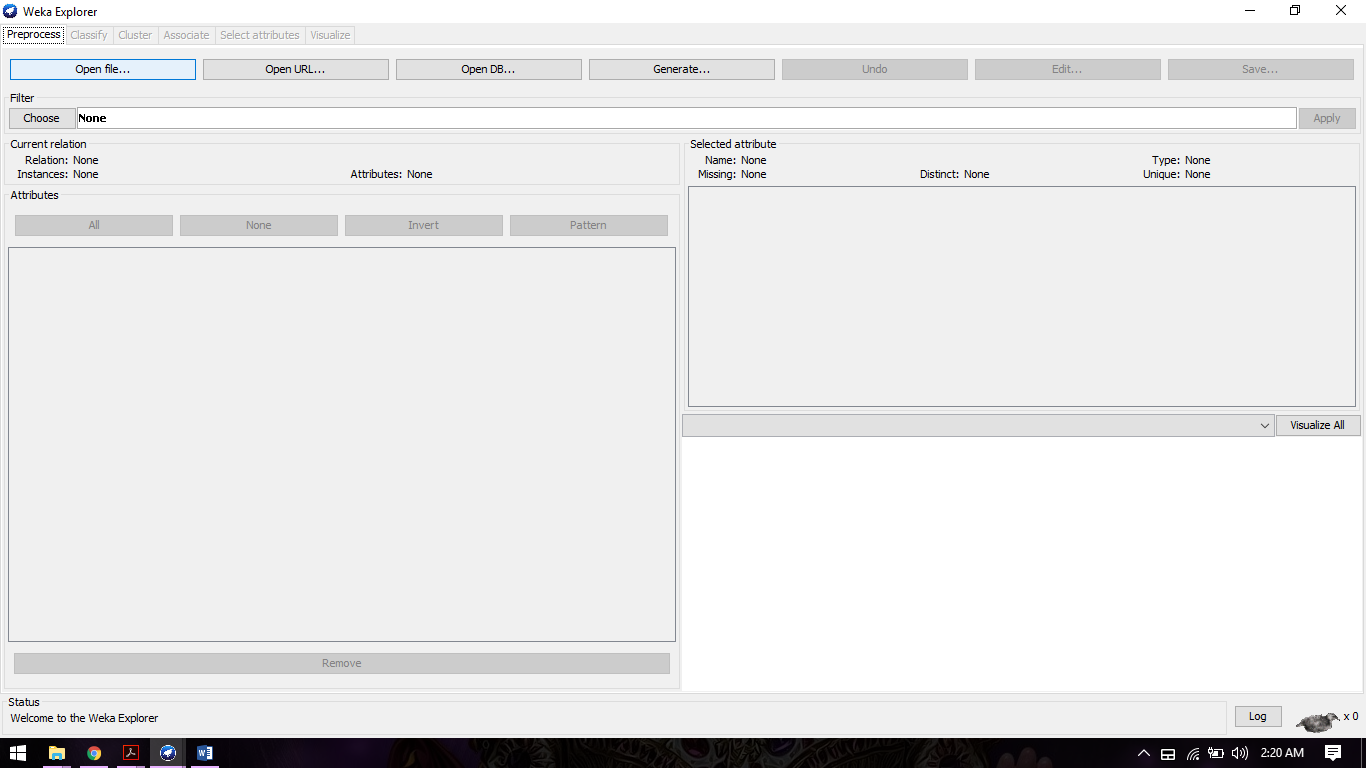
**7.1.3) Classification of Naïve Bayes algorithm:**

Step1: Open Weka GUI Tool, the screen will be look like below image.

Step2: Click on Explorer button.

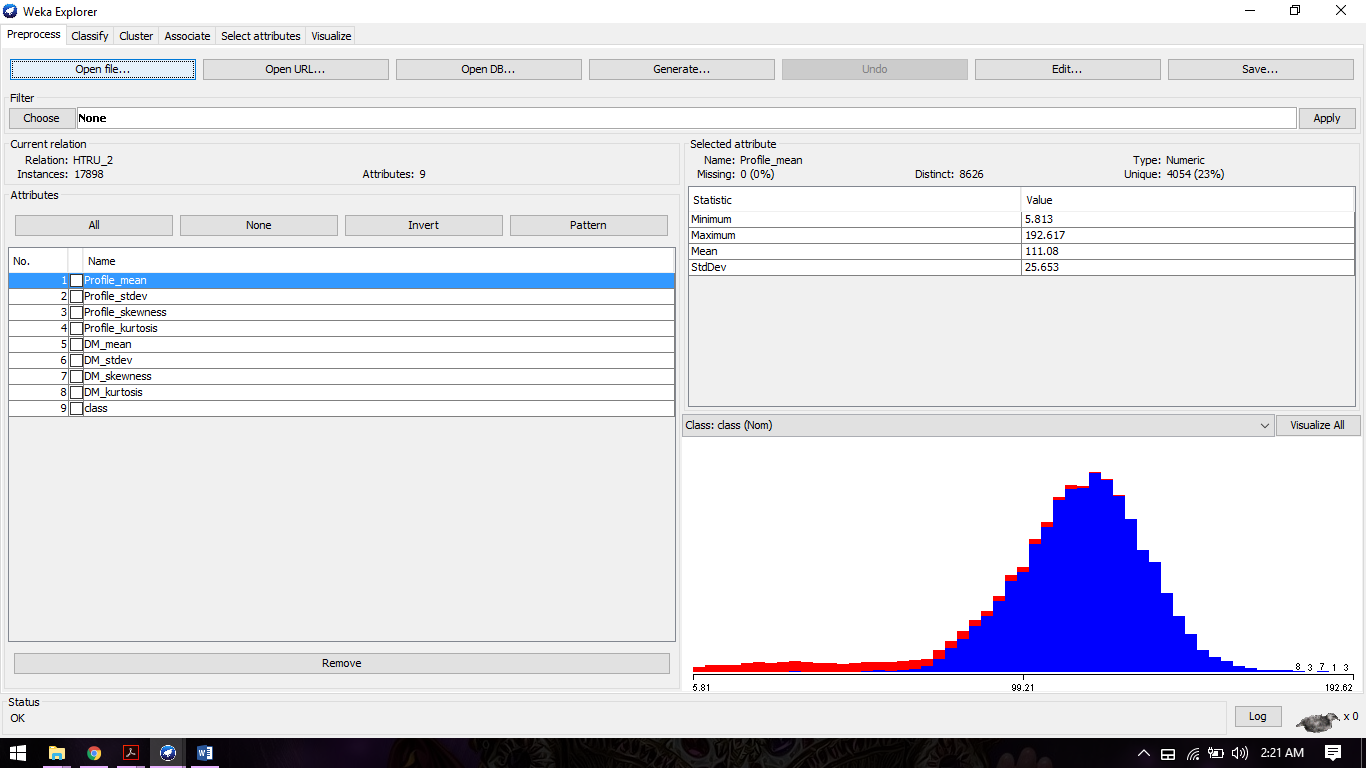


Step3: When we clicked on Explorer button screen will be look like below image.



Step4: Select preprocess tab and click on open file.

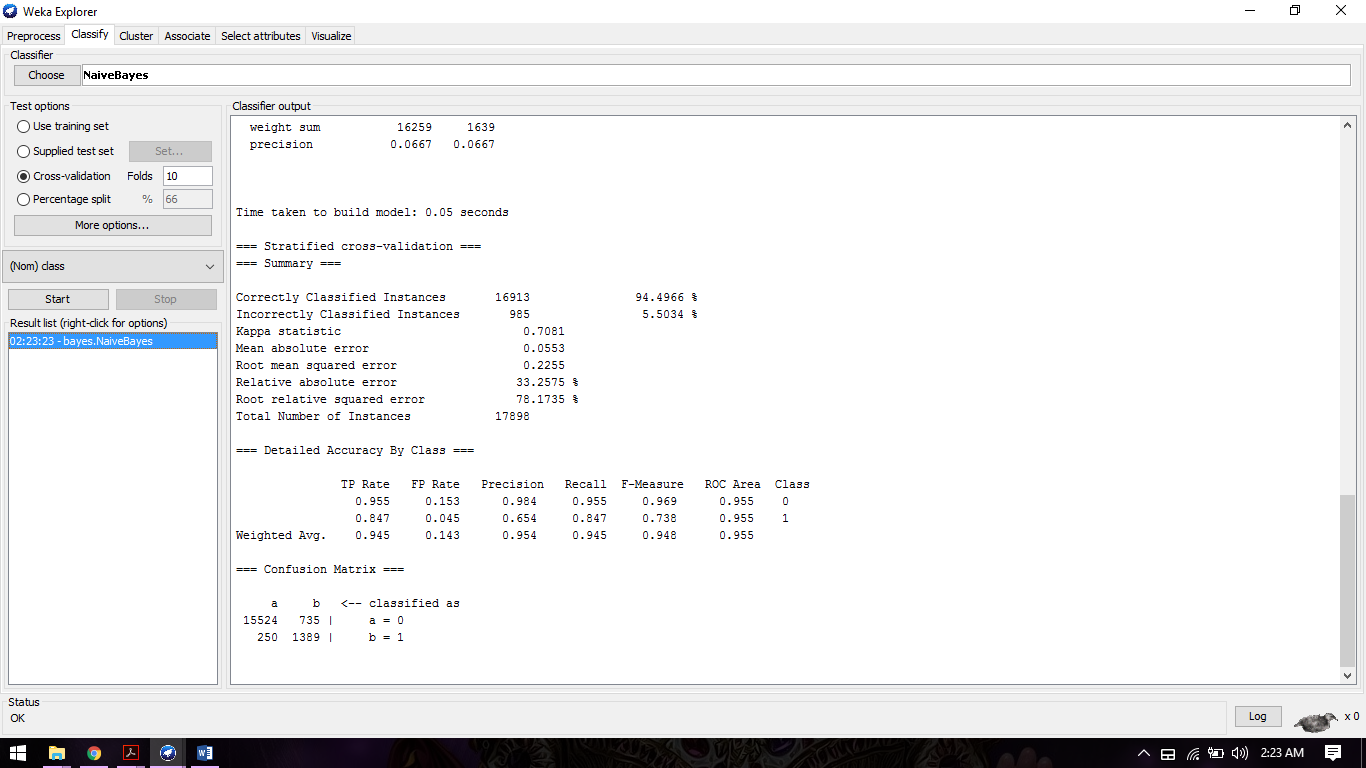
Step5: From the directory of file system choose relevant dataset from local system. After that screen look like below image. We can see all attributes and classes of that dataset which is used for classification of algorithms.



Step 6: Now click on classify tab on top and choose classifier as Naive Bayes algorithms.

Step 7: Choose radio button as cross validation and change Folds as 10.

Step 8: Select start button. The image will look like below image. Verify classification output of Naïve Bayes algorithms.



**7.1.4) Output:**

=== Run information ===

Scheme:weka.classifiers.bayes.NaiveBayes

Relation: HTRU\_2

Instances: 17898

Attributes: 9

Profile\_mean

Profile\_stdev

Profile\_skewness

Profile\_kurtosis

DM\_mean

DM\_stdev

DM\_skewness

DM\_kurtosis

class

Test mode:10-fold cross-validation

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

Naive Bayes Classifier

Class

Attribute 0 1

(0.91) (0.09)

=====================================

Profile\_mean

mean 116.5627 56.6904

std. dev. 17.4754 29.9985

weight sum 16259 1639

precision 0.0217 0.0217

Profile\_stdev

mean 47.3397 38.7106

std. dev. 6.1828 8.0312

weight sum 16259 1639

precision 0.0041 0.0041

Profile\_skewness

mean 0.2104 3.1307

std. dev. 0.3346 1.8723

weight sum 16259 1639

precision 0.0006 0.0006

Profile\_kurtosis

mean 0.3808 15.5536

std. dev. 1.0278 13.9929

weight sum 16259 1639

precision 0.0039 0.0039

DM\_mean

mean 8.8632 49.8264

std. dev. 24.4107 45.2739

weight sum 16259 1639

precision 0.0248 0.0248

DM\_stdev

mean 23.288 56.4689

std. dev. 16.6509 19.7251

weight sum 16259 1639

precision 0.0058 0.0058

DM\_skewness

mean 8.8627 2.757

std. dev. 4.2385 3.105

weight sum 16259 1639

precision 0.0021 0.0021

DM\_kurtosis

mean 113.6202 17.9312

std. dev. 106.7186 50.8811

weight sum 16259 1639

precision 0.0667 0.0667

Time taken to build model: 0.05 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances 16913 94.4966 %

Incorrectly Classified Instances 985 5.5034 %

Kappa statistic 0.7081

Mean absolute error 0.0553

Root mean squared error 0.2255

Relative absolute error 33.2575 %

Root relative squared error 78.1735 %

Total Number of Instances 17898

=== Detailed Accuracy By Class ===

TP Rate FP Rate Precision Recall F-Measure ROC Area Class

0.955 0.153 0.984 0.955 0.969 0.955 0

0.847 0.045 0.654 0.847 0.738 0.955 1

Weighted Avg. 0.945 0.143 0.954 0.945 0.948 0.955

=== Confusion Matrix ===

a b <-- classified as

15524 735 | a = 0

250 1389 | b = 1

**7.2.1) Random Forest Algorithm introduction**

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

The first algorithm for random decision forests was created by Tin Kam Ho using the random subspace method which in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

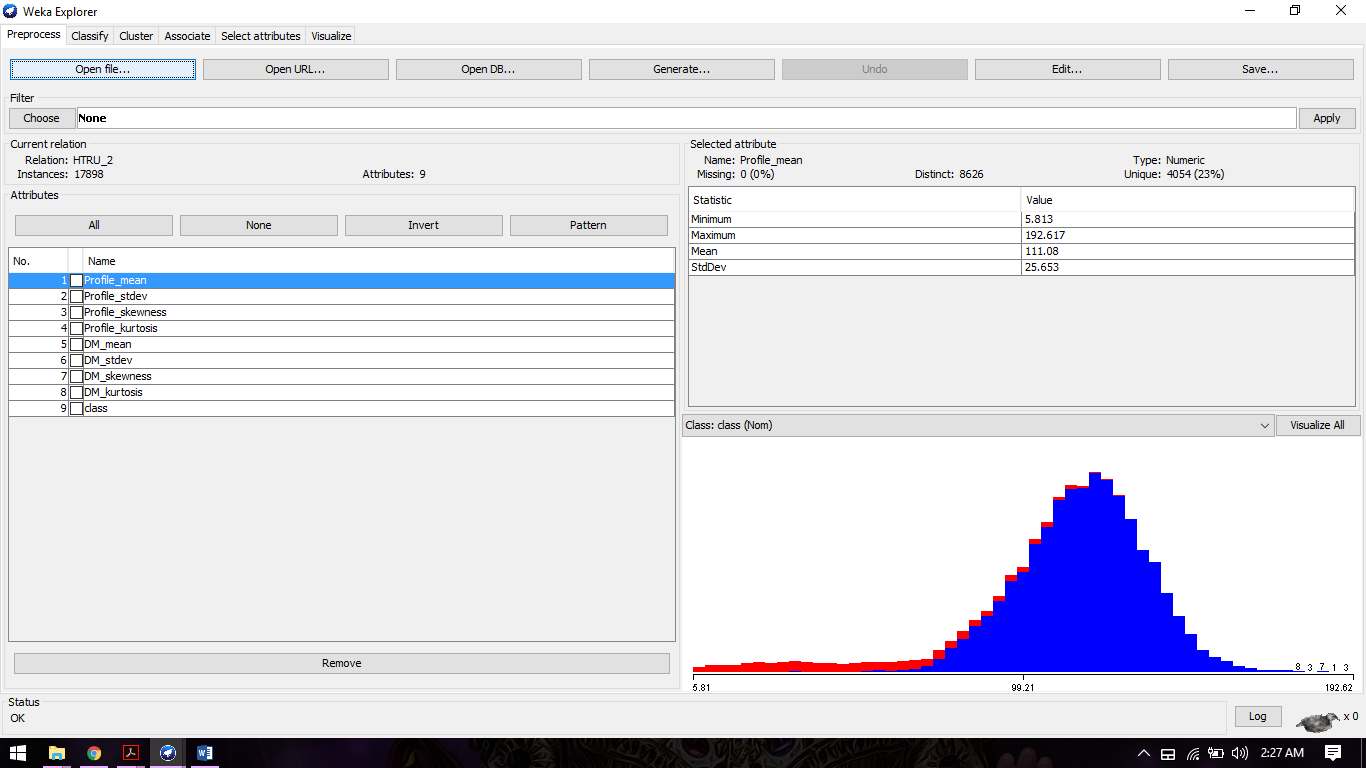
|  |
| --- |
| **7.2.2)** **Random Forest Algorithm (Source Code)**  \* RandomForest.java  \* Copyright (C) 2001 University of Waikato, Hamilton, New Zealand  \*  \*/  package weka.classifiers.trees;  import weka.classifiers.Classifier;  import weka.classifiers.meta.Bagging;  import weka.core.AdditionalMeasureProducer;  import weka.core.Capabilities;  import weka.core.Instance;  import weka.core.Instances;  import weka.core.Option;  import weka.core.OptionHandler;  import weka.core.Randomizable;  import weka.core.RevisionUtils;  import weka.core.TechnicalInformation;  import weka.core.TechnicalInformationHandler;  import weka.core.Utils;  import weka.core.WeightedInstancesHandler;  import weka.core.TechnicalInformation.Field;  import weka.core.TechnicalInformation.Type;  import java.util.Enumeration;  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.  \* <p/>  <!-- 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},  Option 1 Supervised Data Mining (Classification) Page 23  \* year = {2001}  \* }  \* </pre>  \* <p/>  <!-- technical-bibtex-end -->  \*  <!-- options-start -->  \* Valid options are: <p/>  \*  \* <pre> -I &lt;number of trees&gt;  \* Number of trees to build.</pre>  \*  \* <pre> -K &lt;number of features&gt;  \* Number of features to consider (&lt;1=int(logM+1)).</pre>  \*  \* <pre> -S  \* Seed for random number generator.  \* (default 1)</pre>  \*  \* <pre> -depth &lt;num&gt;  \* The maximum depth of the trees, 0 for unlimited.  \* (default 0)</pre>  \*  \* <pre> -D  \* If set, classifier is run in debug mode and  \* may output additional info to the console</pre>  \*  <!-- options-end -->  \*  \* @author Richard Kirkby (rkirkby@cs.waikato.ac.nz)  \* @version $Revision: 1.13 $  \*/  public class RandomForest  extends Classifier  implements OptionHandler, Randomizable, WeightedInstancesHandler,  AdditionalMeasureProducer, TechnicalInformationHandler {  /\*\* for serialization \*/  static final long serialVersionUID = 4216839470751428698L;  /\*\* Number of trees in forest. \*/  protected int m\_numTrees = 10;  /\*\* Number of features to consider in random feature selection.  If less than 1 will use int(logM+1) ) \*/  protected int m\_numFeatures = 0;  /\*\* The random seed. \*/  protected int m\_randomSeed = 1;  /\*\* Final number of features that were considered in last build. \*/  Option 1 Supervised Data Mining (Classification) Page 24  protected int m\_KValue = 0;  /\*\* The bagger. \*/  protected Bagging m\_bagger = null;  /\*\* The maximum depth of the trees (0 = unlimited) \*/  protected int m\_MaxDepth = 0;  /\*\*  \* 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  \*/  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;  }  /\*\*  \* Returns the tip text for this property  \* @return tip text for this property suitable for  \* displaying in the explorer/experimenter gui  \*/  public String numTreesTipText() {  return "The number of trees to be generated.";  }  Option 1 Supervised Data Mining (Classification) Page 25  /\*\*  \* Get the value of numTrees.  \*  \* @return Value of numTrees.  \*/  public int getNumTrees() {  return m\_numTrees;  }  /\*\*  \* Set the value of numTrees.  \*  \* @param newNumTrees Value to assign to numTrees.  \*/  public void setNumTrees(int newNumTrees) {  m\_numTrees = newNumTrees;  }  /\*\*  \* 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 "The number of attributes to be used in random selection (see RandomTree).";  }  /\*\*  \* Get the number of features used in random selection.  \*  \* @return Value of numFeatures.  \*/  public int getNumFeatures() {  return m\_numFeatures;  }  /\*\*  \* Set the number of features to use in random selection.  \*  \* @param newNumFeatures Value to assign to numFeatures.  \*/  public void setNumFeatures(int newNumFeatures) {  m\_numFeatures = newNumFeatures;  }  /\*\*  Option 1 Supervised Data Mining (Classification) Page 26  \* Returns the tip text for this property  \* @return tip text for this property suitable for  \* displaying in the explorer/experimenter gui  \*/  public String seedTipText() {  return "The random number seed to be used.";  }  /\*\*  \* Set the seed for random number generation.  \*  \* @param seed the seed  \*/  public void setSeed(int seed) {  m\_randomSeed = seed;  }  /\*\*  \* Gets the seed for the random number generations  \*  \* @return the seed for the random number generation  \*/  public int getSeed() {  return m\_randomSeed;  }  /\*\*  \* 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 "The maximum depth of the trees, 0 for unlimited.";  }  /\*\*  \* Get the maximum depth of trh tree, 0 for unlimited.  \*  \* @return the maximum depth.  \*/  public int getMaxDepth() {  return m\_MaxDepth;  }  /\*\*  \* Set the maximum depth of the tree, 0 for unlimited.  \*  \* @param value the maximum depth.  Option 1 Supervised Data Mining (Classification) Page 27  \*/  public void setMaxDepth(int value) {  m\_MaxDepth = value;  }  /\*\*  \* Gets the out of bag error that was calculated as the classifier was built.  \*  \* @return the out of bag error  \*/  public double measureOutOfBagError() {  if (m\_bagger != null) {  return m\_bagger.measureOutOfBagError();  } else return Double.NaN;  }  /\*\*  \* Returns an enumeration of the additional measure names.  \*  \* @return an enumeration of the measure names  \*/  public Enumeration enumerateMeasures() {  Vector newVector = new Vector(1);  newVector.addElement("measureOutOfBagError");  return newVector.elements();  }  /\*\*  \* Returns the value of the named measure.  \*  \* @param additionalMeasureName 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  \*/  public double getMeasure(String additionalMeasureName) {  if (additionalMeasureName.equalsIgnoreCase("measureOutOfBagError")) {  return measureOutOfBagError();  }  else {throw new IllegalArgumentException(additionalMeasureName  + " not supported (RandomForest)");  }  }  /\*\*  \* Returns an enumeration describing the available options.  \*  \* @return an enumeration of all the available options  \*/  Option 1 Supervised Data Mining (Classification) Page 28  public Enumeration listOptions() {  Vector newVector = new Vector();  newVector.addElement(new Option(  "\tNumber of trees to build.",  "I", 1, "-I <number of trees>"));  newVector.addElement(new Option(  "\tNumber of features to consider (<1=int(logM+1)).",  "K", 1, "-K <number of features>"));  newVector.addElement(new Option(  "\tSeed for random number generator.\n"  + "\t(default 1)",  "S", 1, "-S"));  newVector.addElement(new Option(  "\tThe maximum depth of the trees, 0 for unlimited.\n"  + "\t(default 0)",  "depth", 1, "-depth <num>"));  Enumeration enu = super.listOptions();  while (enu.hasMoreElements()) {  newVector.addElement(enu.nextElement());  }  return newVector.elements();  }  /\*\*  \* Gets the current settings of the forest.  \*  \* @return an array of strings suitable for passing to setOptions()  \*/  public String[] getOptions() {  Vector result;  String[] options;  int i;  result = new Vector();  result.add("-I");  result.add("" + getNumTrees());  result.add("-K");  result.add("" + getNumFeatures());  result.add("-S");  result.add("" + getSeed());  Option 1 Supervised Data Mining (Classification) Page 29  if (getMaxDepth() > 0) {  result.add("-depth");  result.add("" + getMaxDepth());  }  options = super.getOptions();  for (i = 0; i < options.length; i++)  result.add(options[i]);  return (String[]) result.toArray(new String[result.size()]);  }  /\*\*  \* Parses a given list of options. <p/>  \*  <!-- options-start -->  \* Valid options are: <p/>  \*  \* <pre> -I &lt;number of trees&gt;  \* Number of trees to build.</pre>  \*  \* <pre> -K &lt;number of features&gt;  \* Number of features to consider (&lt;1=int(logM+1)).</pre>  \*  \* <pre> -S  \* Seed for random number generator.  \* (default 1)</pre>  \*  \* <pre> -depth &lt;num&gt;  \* The maximum depth of the trees, 0 for unlimited.  \* (default 0)</pre>  \*  \* <pre> -D  \* If set, classifier is run in debug mode and  \* may output additional info to the console</pre>  \*  <!-- options-end -->  \*  \* @param options the list of options as an array of strings  \* @throws Exception if an option is not supported  \*/  public void setOptions(String[] options) throws Exception{  String tmpStr;  tmpStr = Utils.getOption('I', options);  if (tmpStr.length() != 0) {  m\_numTrees = Integer.parseInt(tmpStr);  } else {  m\_numTrees = 10;  }  Option 1 Supervised Data Mining (Classification) Page 30  tmpStr = Utils.getOption('K', options);  if (tmpStr.length() != 0) {  m\_numFeatures = Integer.parseInt(tmpStr);  } else {  m\_numFeatures = 0;  }  tmpStr = Utils.getOption('S', options);  if (tmpStr.length() != 0) {  setSeed(Integer.parseInt(tmpStr));  } else {  setSeed(1);  }  tmpStr = Utils.getOption("depth", options);  if (tmpStr.length() != 0) {  setMaxDepth(Integer.parseInt(tmpStr));  } else {  setMaxDepth(0);  }  super.setOptions(options);  Utils.checkForRemainingOptions(options);  }  /\*\*  \* Returns default capabilities of the classifier.  \*  \* @return the capabilities of this classifier  \*/  public Capabilities getCapabilities() {  return new RandomTree().getCapabilities();  }  /\*\*  \* Builds a classifier for a set of instances.  \*  \* @param data the instances to train the classifier with  \* @throws Exception if something goes wrong  \*/  public void buildClassifier(Instances data) throws Exception {  // can classifier handle the data?  getCapabilities().testWithFail(data);  // remove instances with missing class  data = new Instances(data);  data.deleteWithMissingClass();  m\_bagger = new Bagging();  Option 1 Supervised Data Mining (Classification) Page 31  RandomTree rTree = new RandomTree();  // set up the random tree options  m\_KValue = m\_numFeatures;  if (m\_KValue < 1) m\_KValue = (int) Utils.log2(data.numAttributes())+1;  rTree.setKValue(m\_KValue);  rTree.setMaxDepth(getMaxDepth());  // set up the bagger and build the forest  m\_bagger.setClassifier(rTree);  m\_bagger.setSeed(m\_randomSeed);  m\_bagger.setNumIterations(m\_numTrees);  m\_bagger.setCalcOutOfBag(true);  m\_bagger.buildClassifier(data);  }  /\*\*  \* Returns the class probability distribution for an instance.  \*  \* @param instance the instance to be classified  \* @return the distribution the forest generates for the instance  \* @throws Exception if computation fails  \*/  public double[] distributionForInstance(Instance instance) throws Exception {  return m\_bagger.distributionForInstance(instance);  }  /\*\*  \* Outputs a description of this classifier.  \*  \* @return a string containing a description of the classifier  \*/  public String toString() {  if (m\_bagger == null)  return "Random forest not built yet";  else  return "Random forest of " + m\_numTrees  + " trees, each constructed while considering "  + m\_KValue + " random feature" + (m\_KValue==1 ? "" : "s") + ".\n"  + "Out of bag error: "  + Utils.doubleToString(m\_bagger.measureOutOfBagError(), 4) + "\n"  + (getMaxDepth() > 0 ? ("Max. depth of trees: " + getMaxDepth() + "\n") : (""))  + "\n";  }  /\*\*  \* Returns the revision string.  \*  \* @return the revision  Option 1 Supervised Data Mining (Classification) Page 32  \*/  public String getRevision() {  return RevisionUtils.extract("$Revision: 1.13 $");  }  /\*\*  \* Main method for this class.  \*  \* @param argv the options  \*/  public static void main(String[] argv) {  runClassifier(new RandomForest(), argv);  }  } |

**7.2.3) Classification of Random Forest Algorithms:**

Step1: Open Weka GUI and select Explorer button.

Step2: Select preprocessor tab, now select open file.

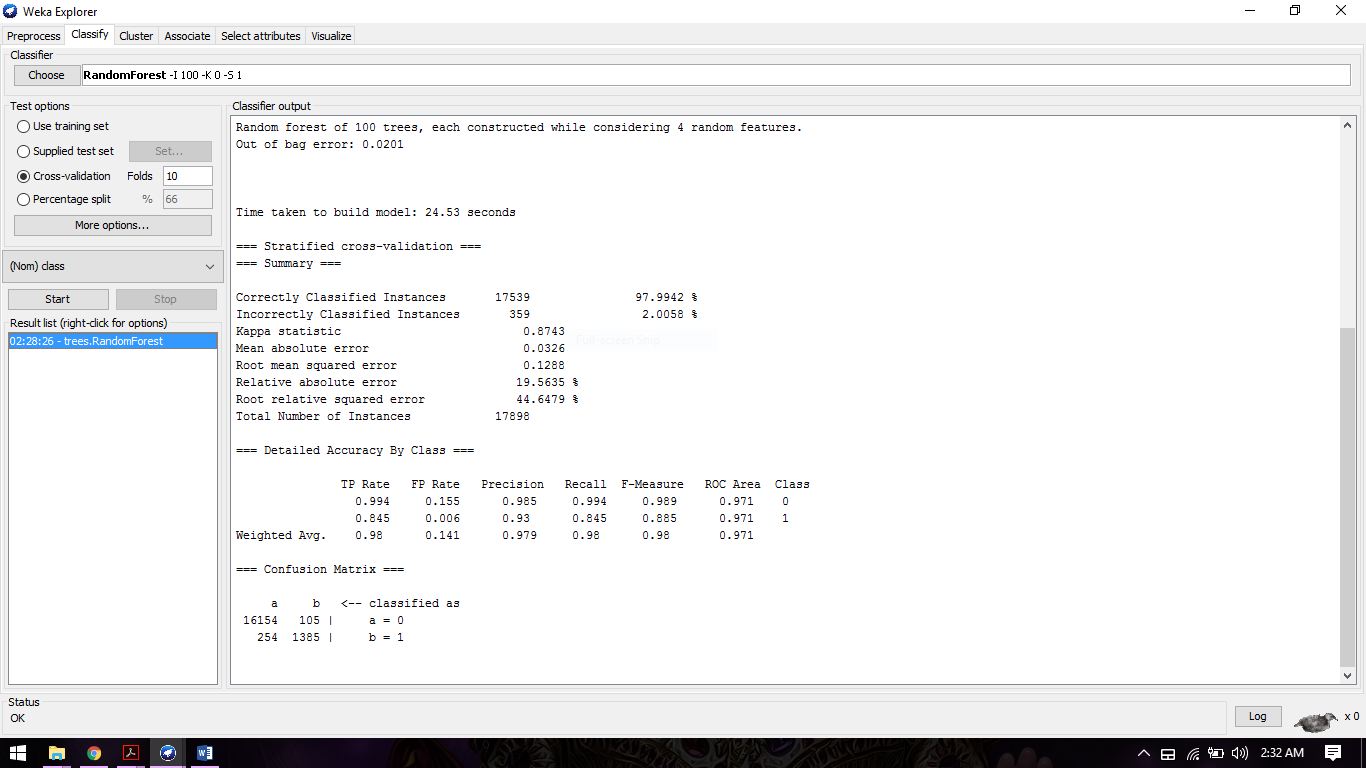
Step 3: Choose your dataset from the local file system. The image will be look like below. The image shows all attributes as well as relevant graphs.



Step 4: select Classify tab, now choose classifier as Random Forest.

Step 5: make sure that cross validation folds should be 10. And Press start button.

Step 6: the image will be look like below image, verify classification output of Random Forest algorithms.



**7.2.4) Output:**

=== Run information ===

Scheme:weka.classifiers.trees.RandomForest -I 100 -K 0 -S 1

Relation: HTRU\_2

Instances: 17898

Attributes: 9

Profile\_mean

Profile\_stdev

Profile\_skewness

Profile\_kurtosis

DM\_mean

DM\_stdev

DM\_skewness

DM\_kurtosis

class

Test mode:10-fold cross-validation

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

Random forest of 100 trees, each constructed while considering 4 random features.

Out of bag error: 0.0201

Time taken to build model: 24.53 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances 17539 97.9942 %

Incorrectly Classified Instances 359 2.0058 %

Kappa statistic 0.8743

Mean absolute error 0.0326

Root mean squared error 0.1288

Relative absolute error 19.5635 %

Root relative squared error 44.6479 %

Total Number of Instances 17898

=== Detailed Accuracy By Class ===

TP Rate FP Rate Precision Recall F-Measure ROC Area Class

0.994 0.155 0.985 0.994 0.989 0.971 0

0.845 0.006 0.93 0.845 0.885 0.971 1

Weighted Avg. 0.98 0.141 0.979 0.98 0.98 0.971

=== Confusion Matrix ===

a b <-- classified as

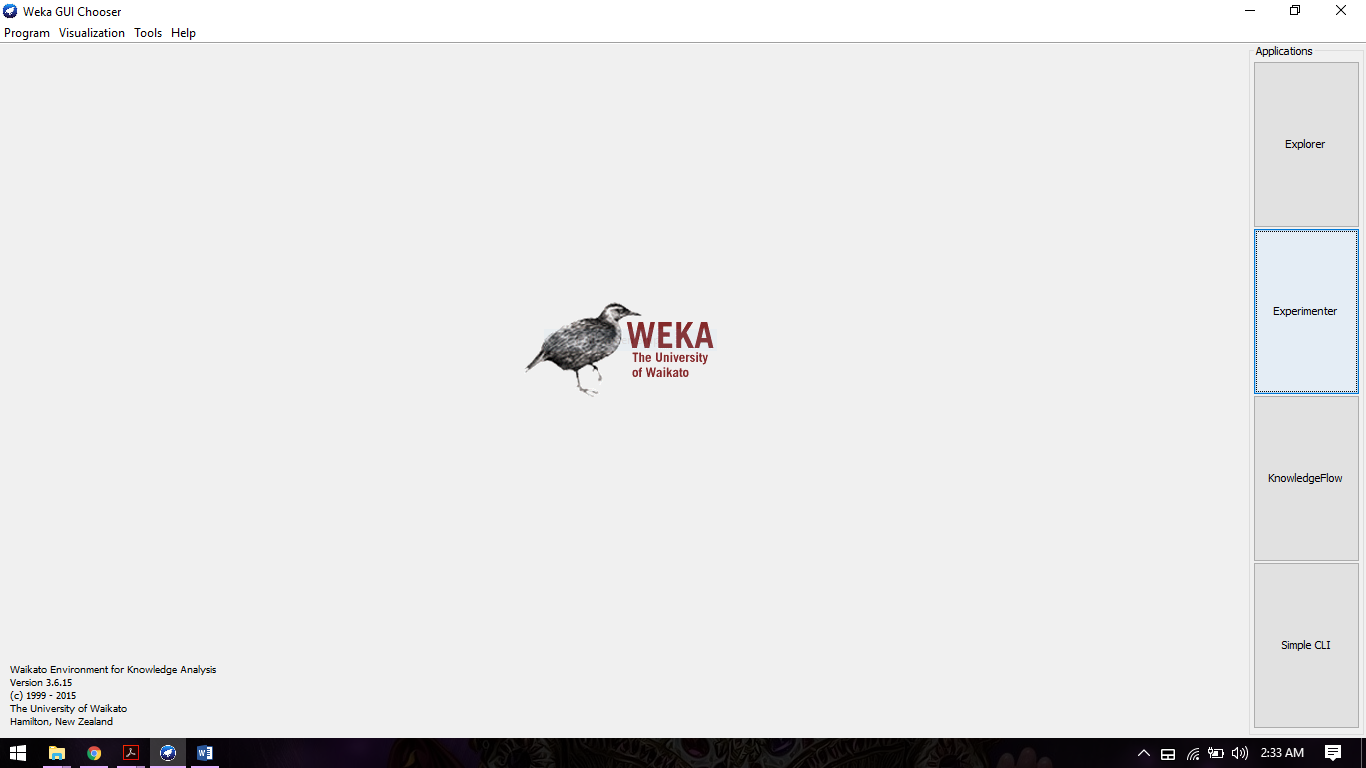
16154 105 | a = 0

254 1385 | b = 1

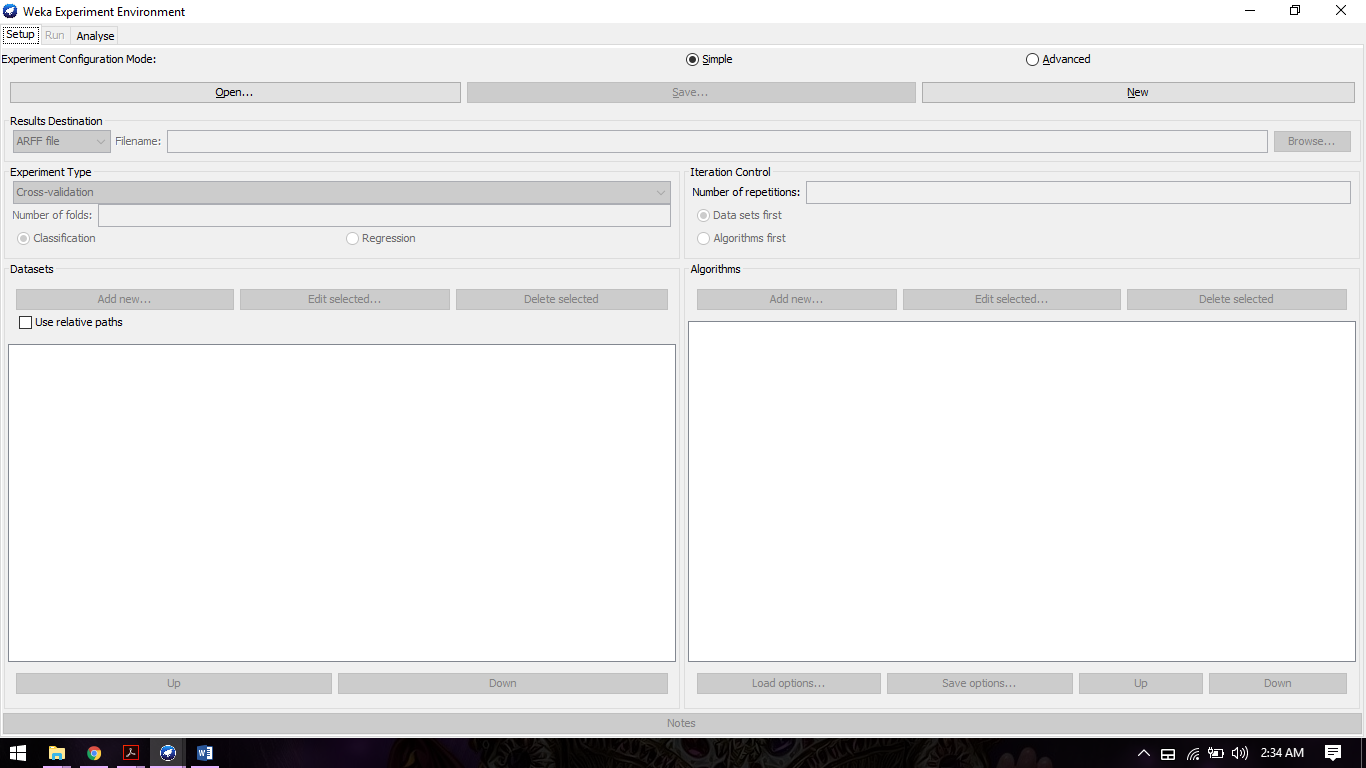
**8) Comparison of both algorithms:**

Now For Comparing Both algorithms follow below steps.

Step 1: Open Weka GUI.



Step 2: Select Experimenter button. Image will be look like below.

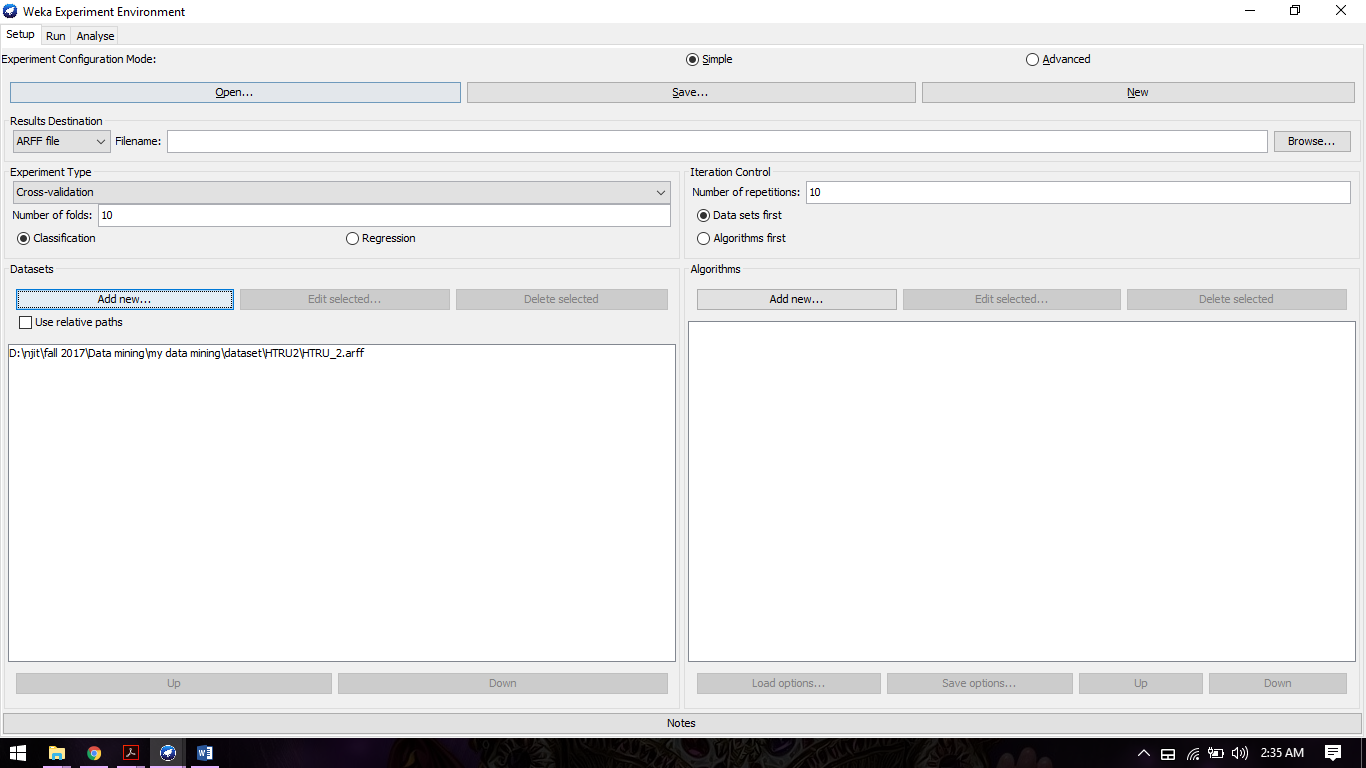


Step 3: Select New tab.

Step 4: Make sure that number of folds should be 10

Classification radio button should be active

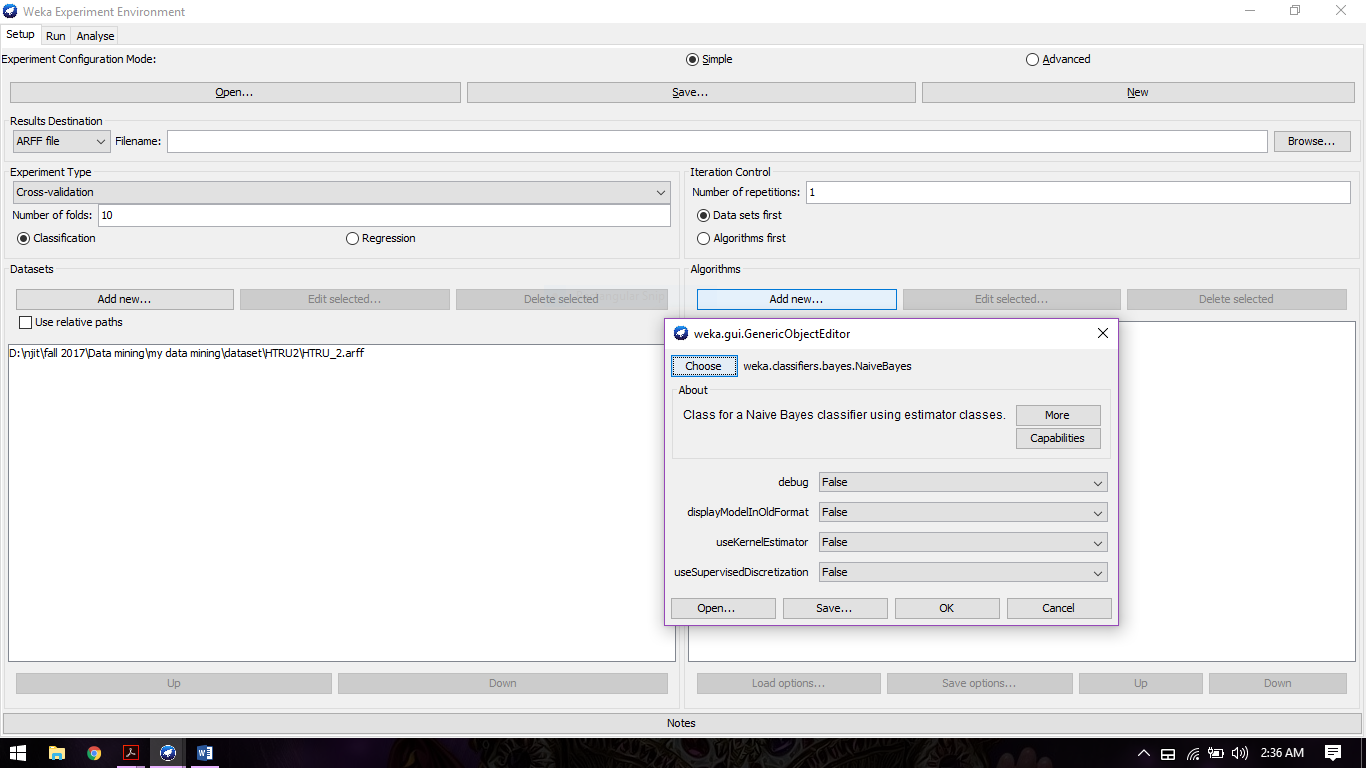
Step 5: Now select add New dataset from local file system.



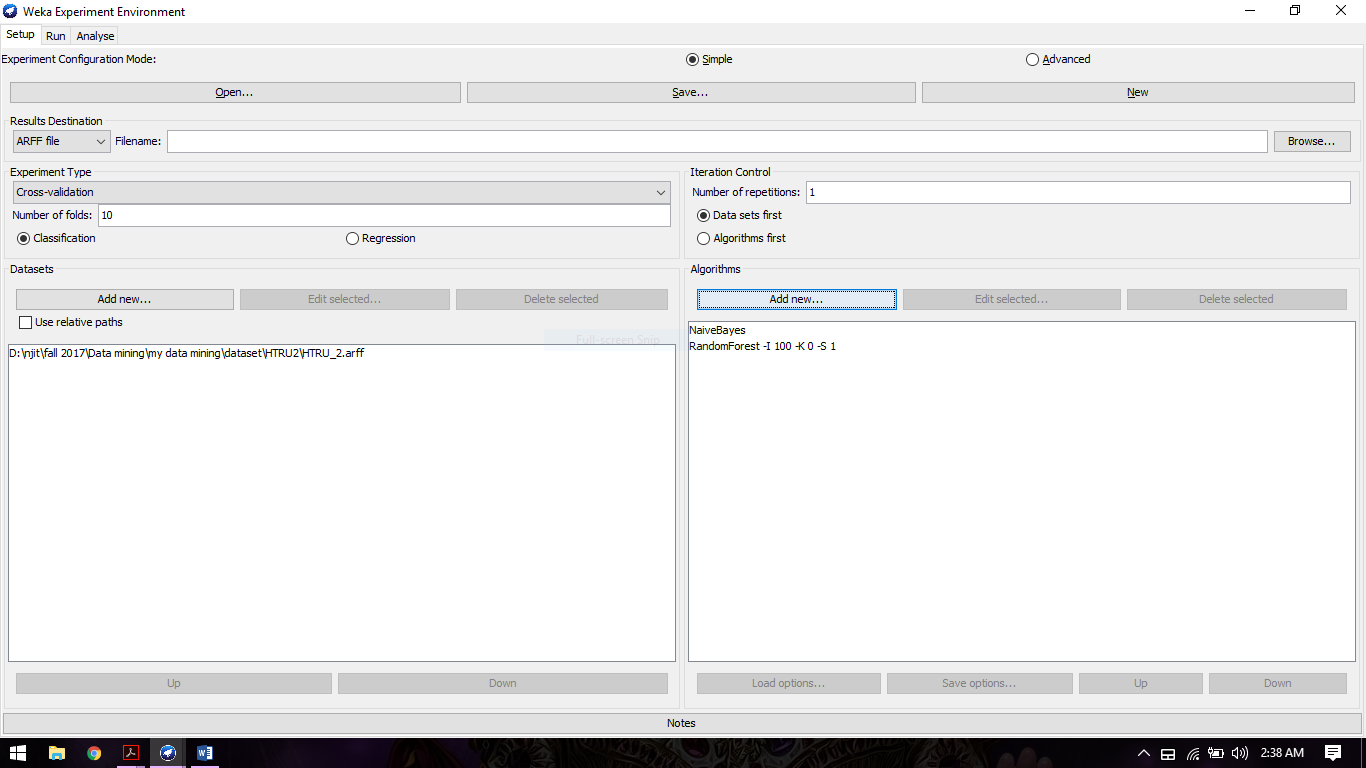
Step 6: Change number of repetition from 10 to 1.

Step 7: Select Add new algorithms button. Now one popup menu will open.

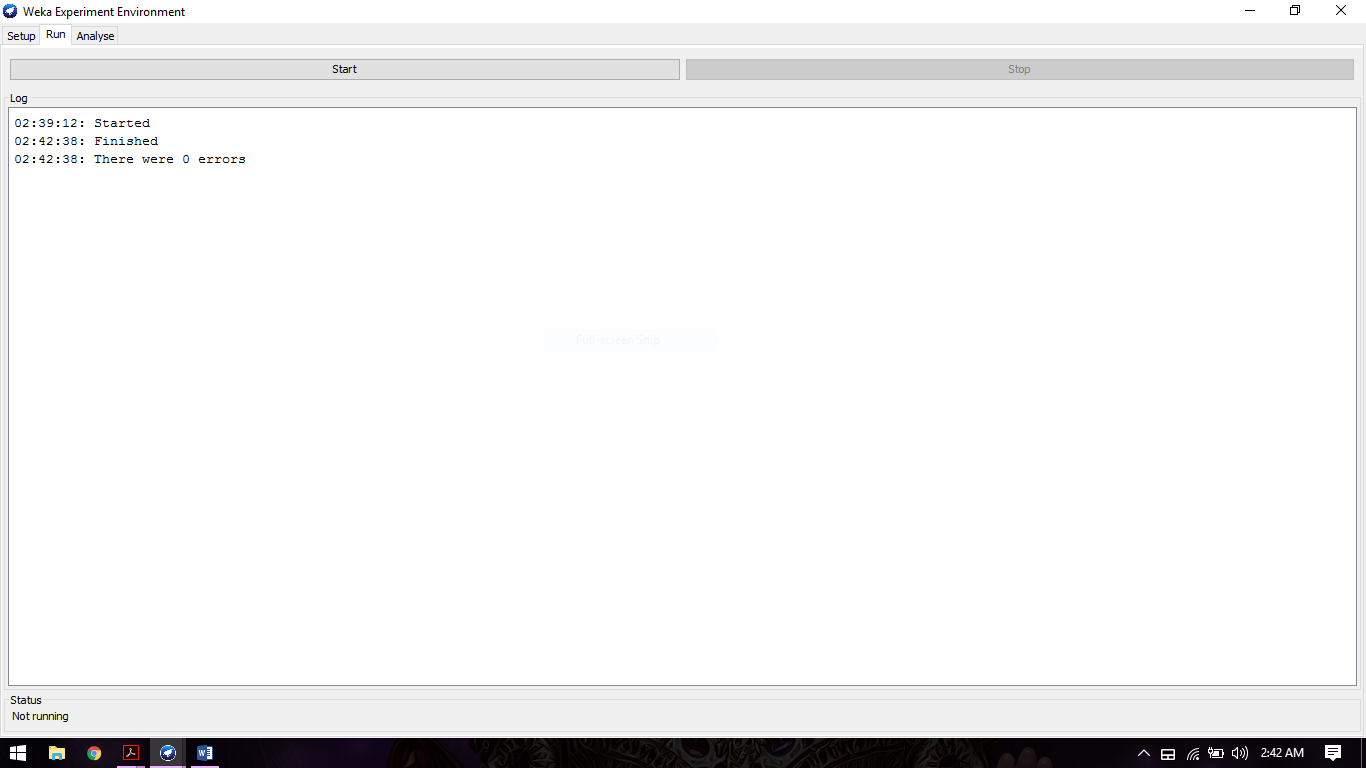
Step 8: Select choose and in that select Bayes ~>Naïve Bayes.



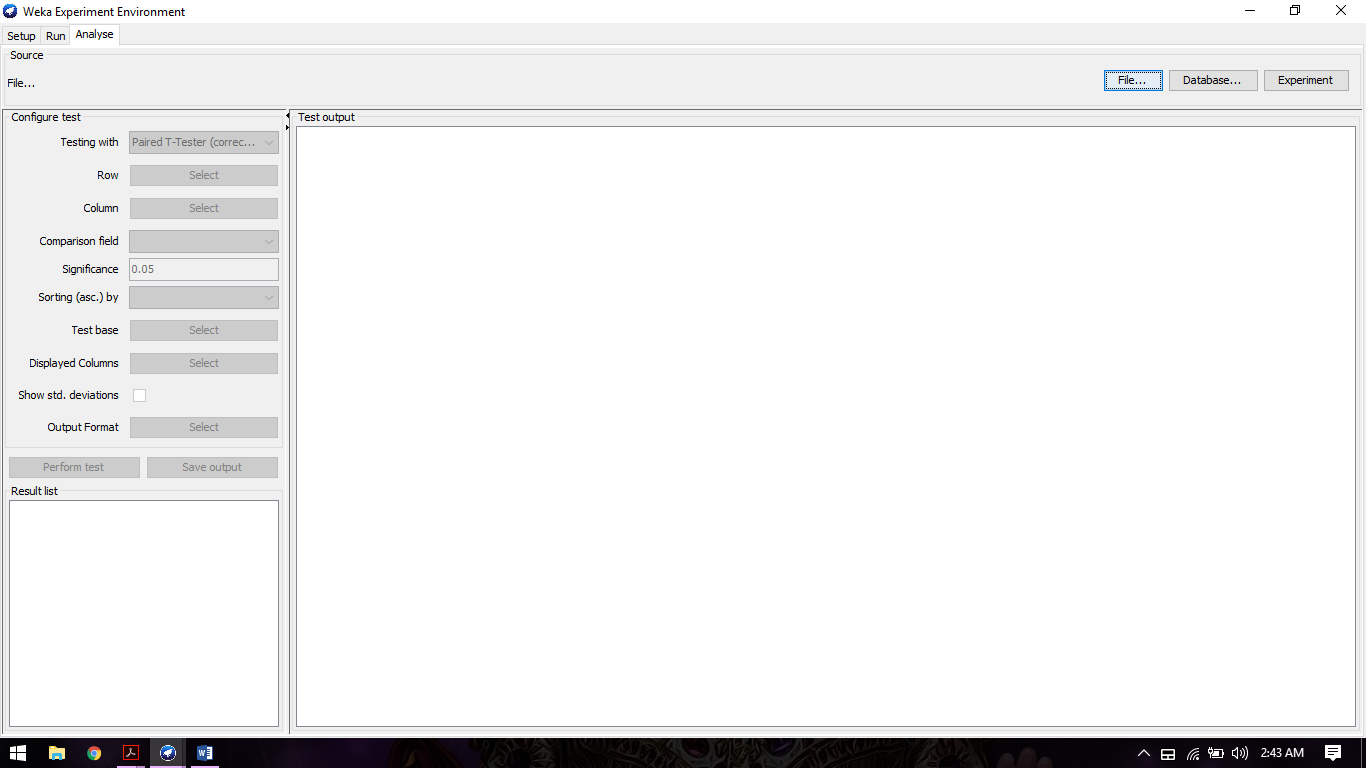
Step 9: follow same procedure for Random Forest Algorithms and the image will be look like below



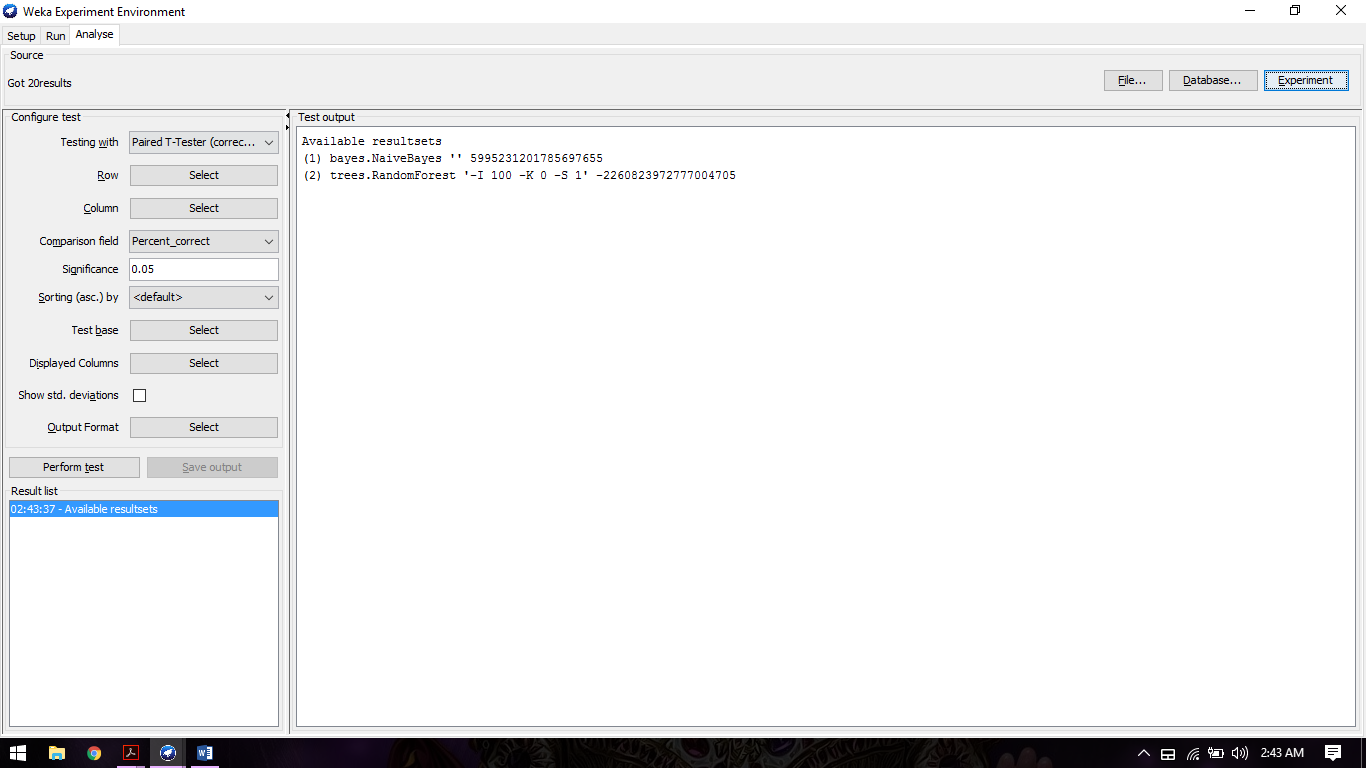
Step 10: Now go to Run tab. and press start. Watch logs till process finished. Check if there any error.



Step 11: Select Analyse tab. the image will be look like below.

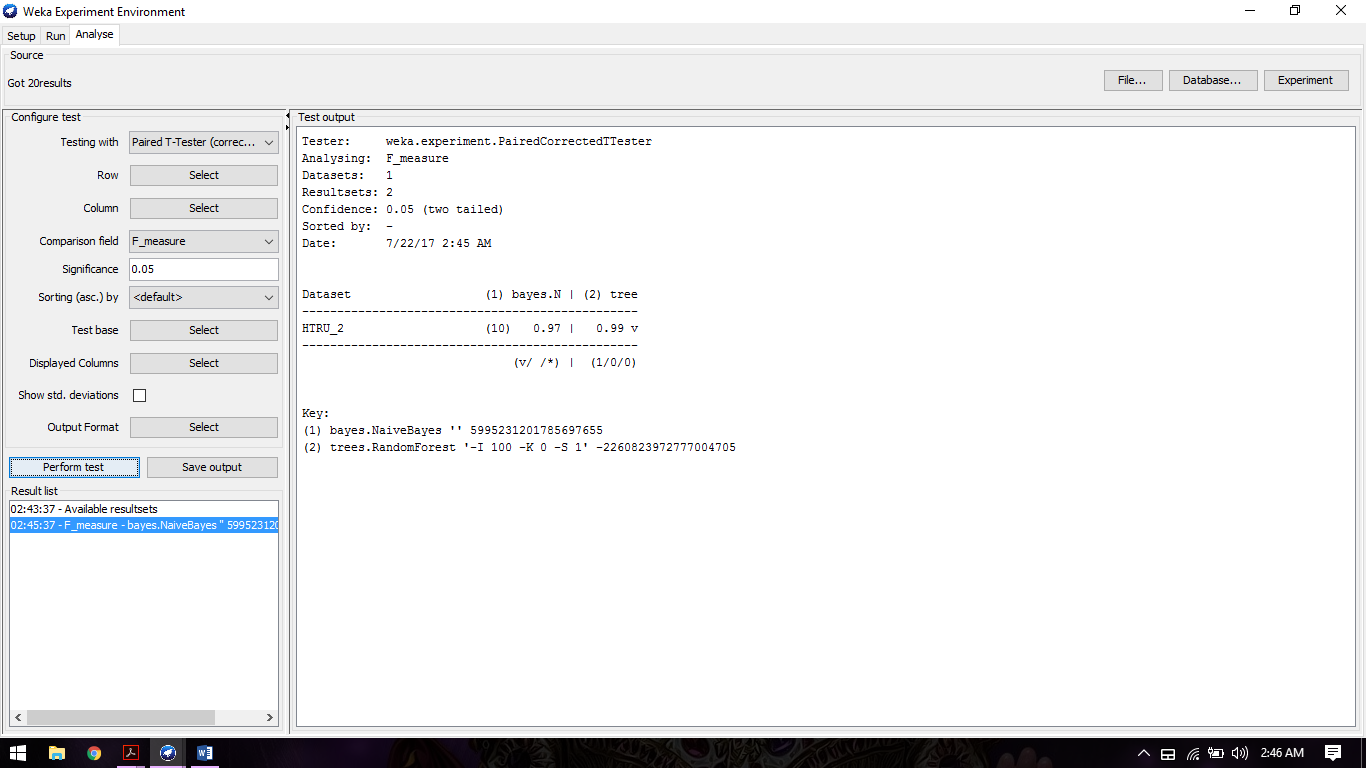


Step12: Now press Experiment tab. now look in configure test and change following tabs.



Step 13: Change comparison Field as F\_Measure, now select Perform Test

Step 14: Check output.



**Output:**

Tester: weka.experiment.PairedCorrectedTTester

Analysing: F\_measure

Datasets: 1

Resultsets: 2

Confidence: 0.05 (two tailed)

Sorted by: -

Date: 7/22/17 2:45 AM

Dataset (1) bayes.N | (2) tree

------------------------------------------------

HTRU\_2 (10) 0.97 | 0.99 v

------------------------------------------------

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

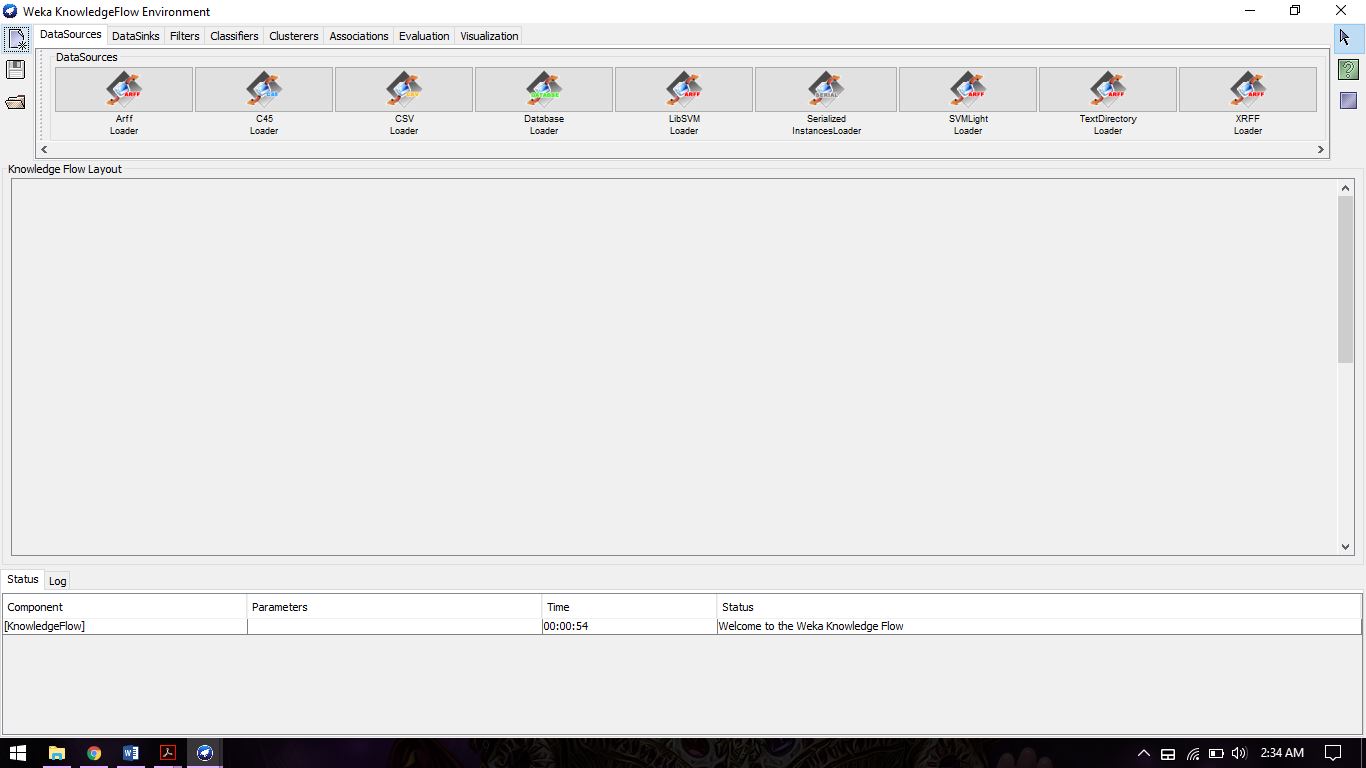
Key:

(1) bayes.NaiveBayes '' 5995231201785697655

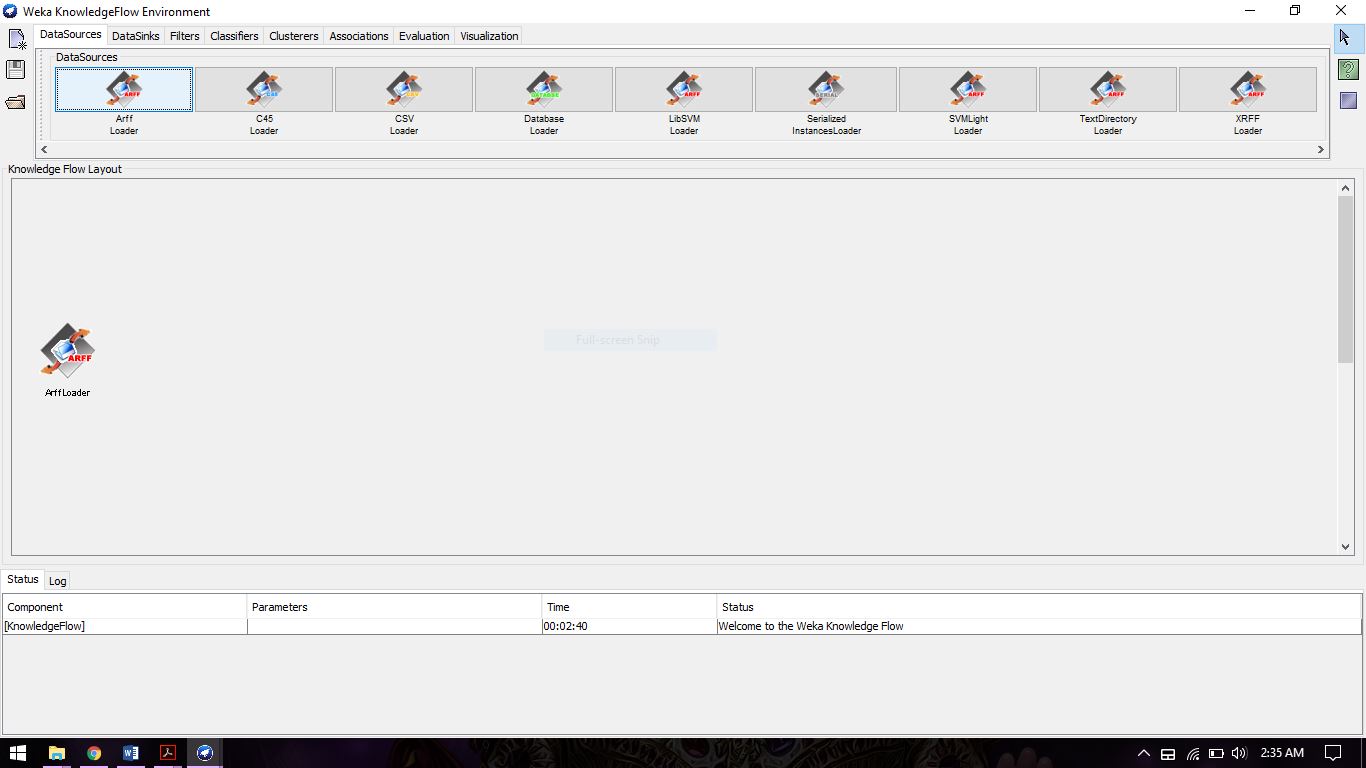
(2) trees.RandomForest '-I 100 -K 0 -S 1' -2260823972777004705

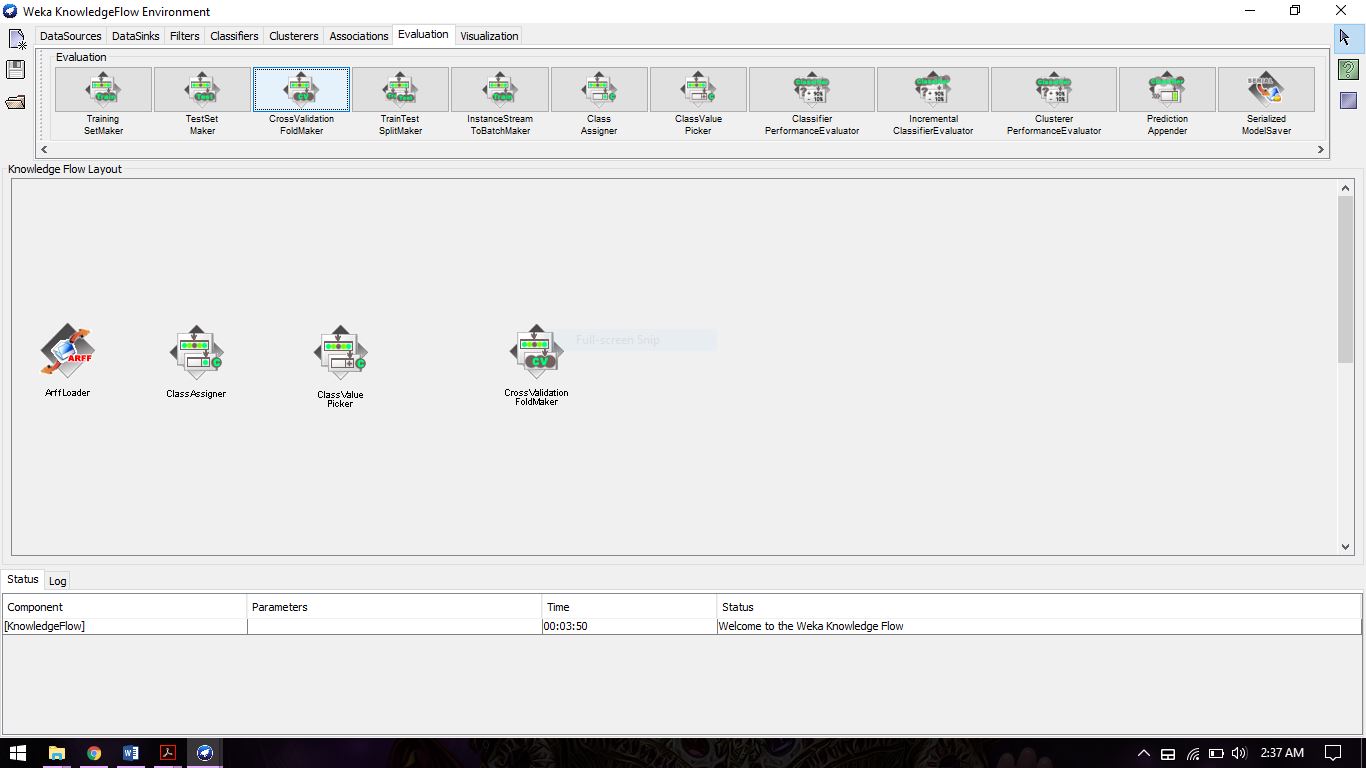
**9) Multiple Roc Curve:**

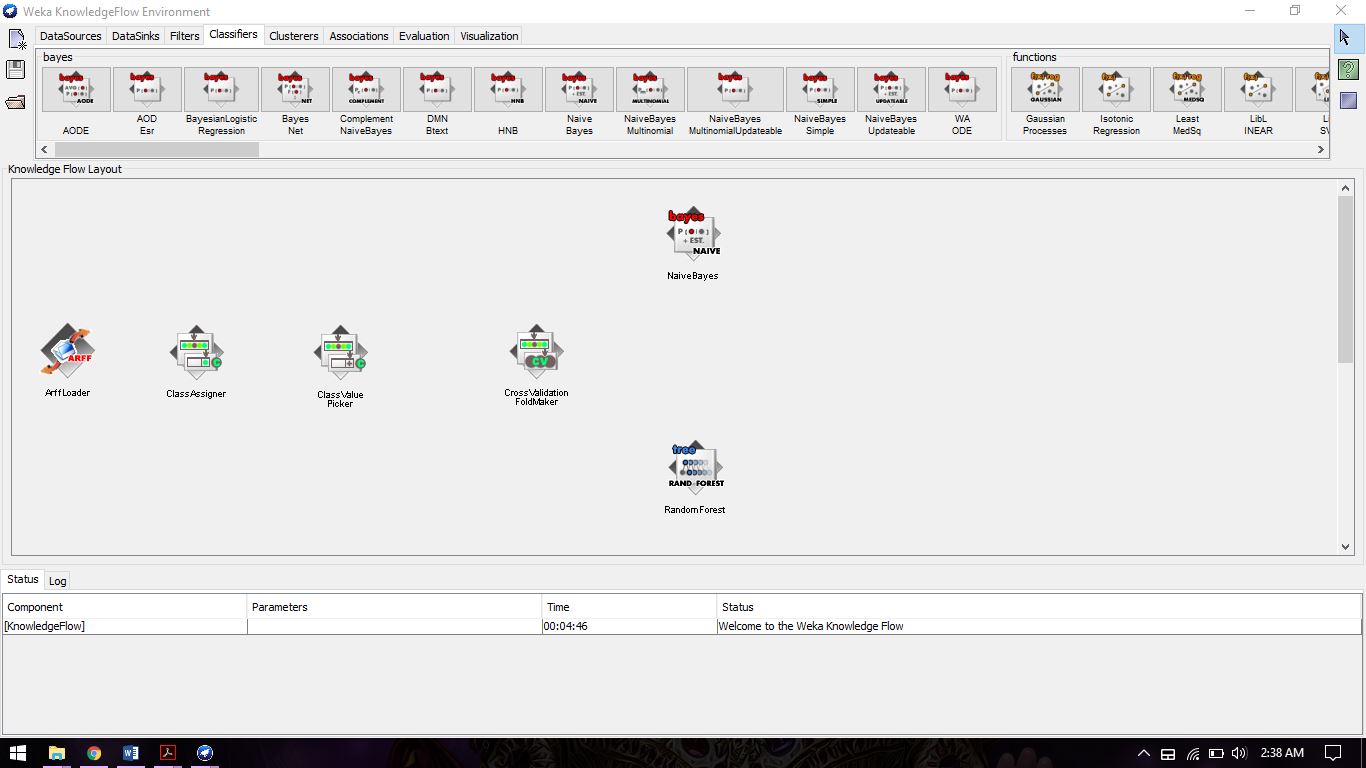
**Step 1:** Open Weka tool. Select KnowledgeFlow, the image will look like below

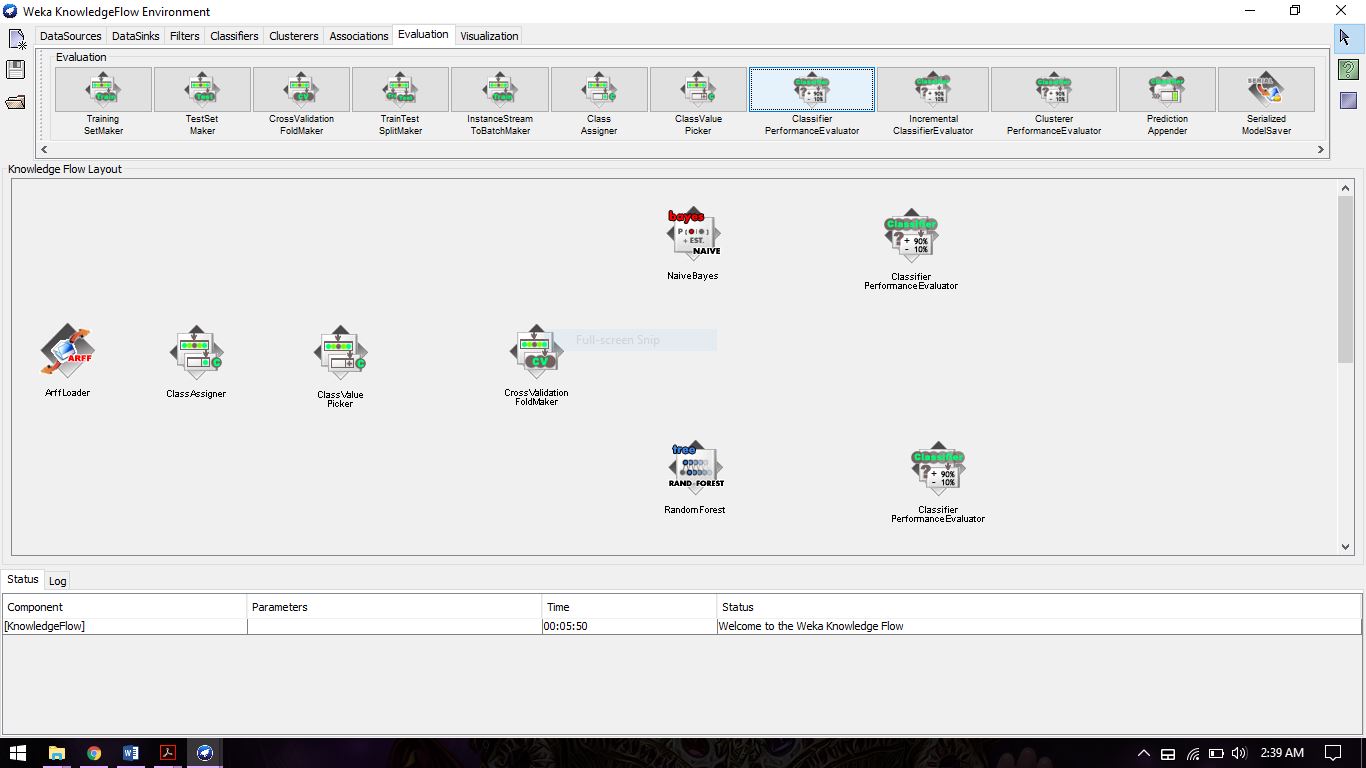
****

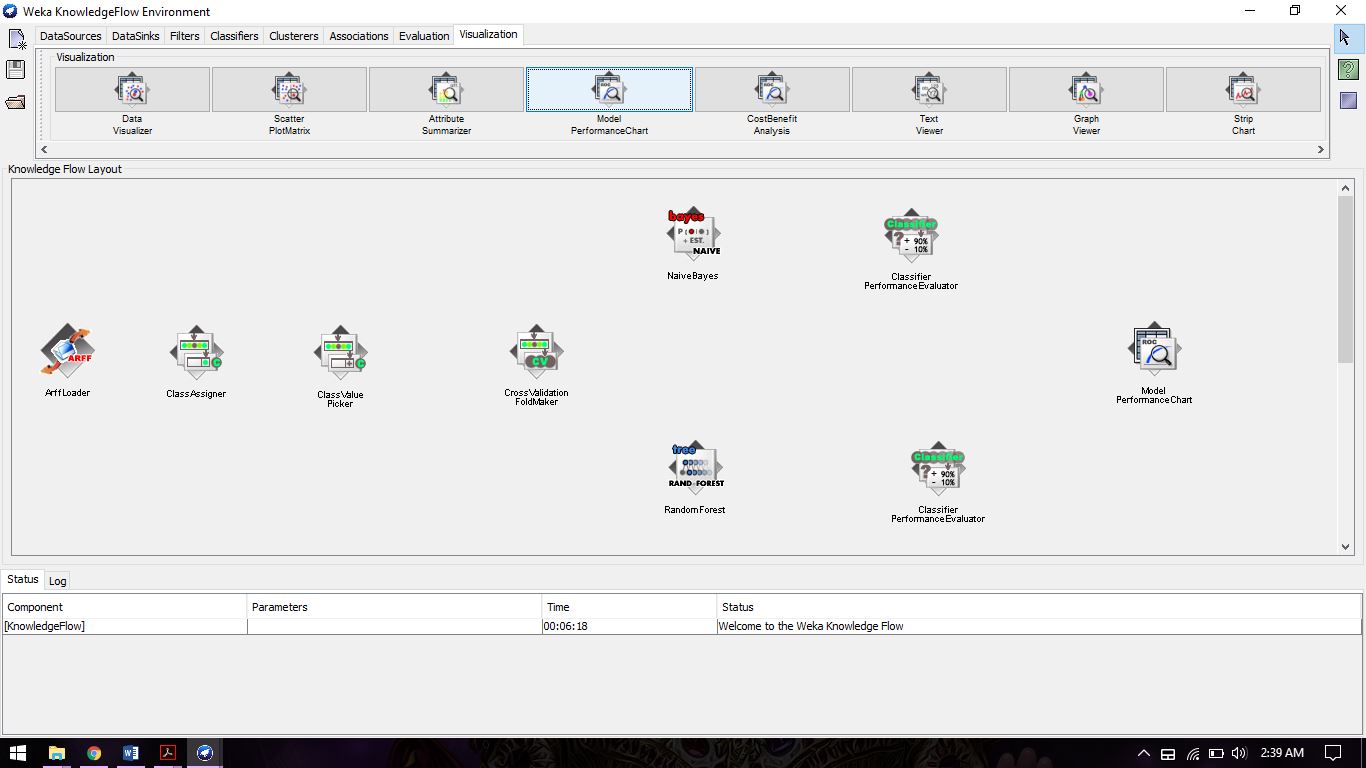
**Step 2:** Select Arff Loader and drag into Knowledge Flow Layout so that it will look like below image.

****

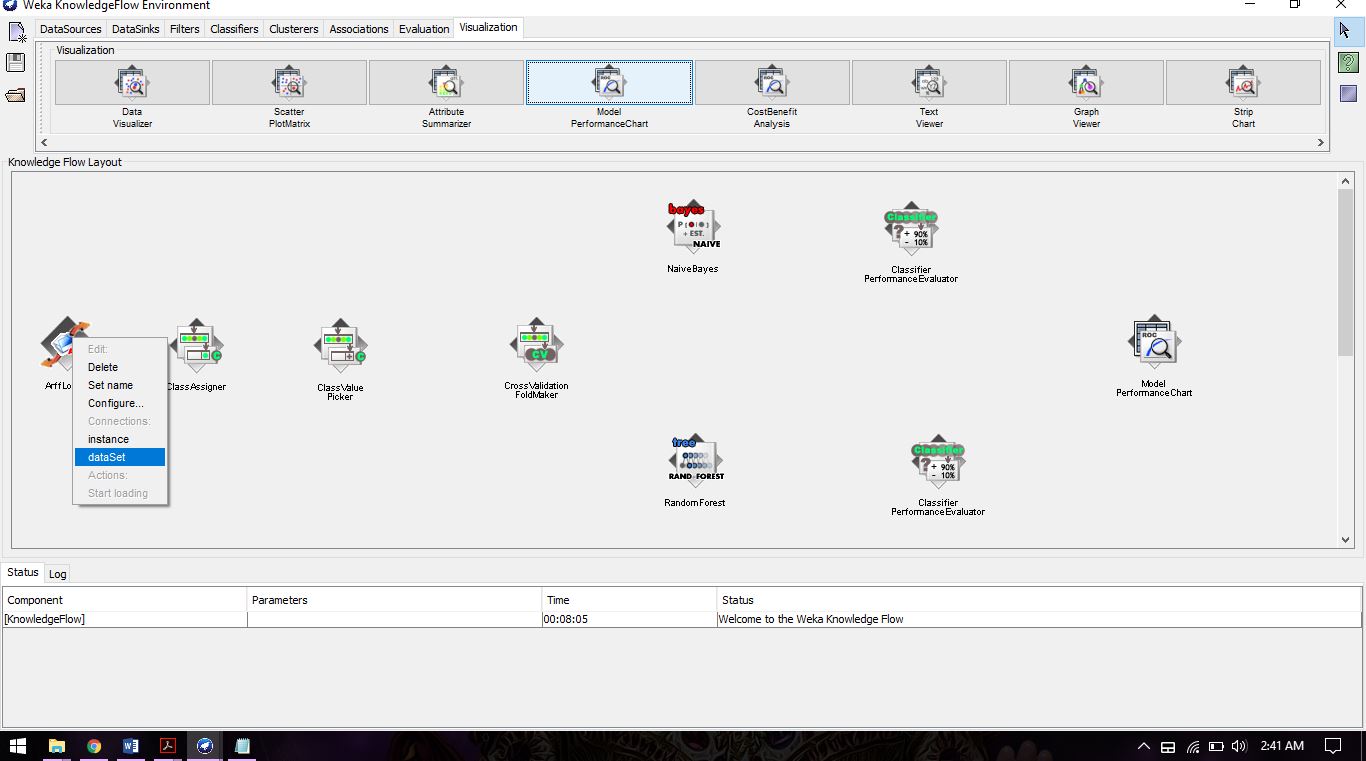
**Step3:** Now go to the Evolution tab and select and drag three icon which are ClassAssigner, ClassValue Picker and CrossValidationFoldMaker ****

**Step 4:** Go to Classifies tab, Select Naïve Bayes and Random Forest Icons and drag into Knowledge Flow Layout.****

**Step 5:** Now go to Evaluation tab and choose Classifier Performance Evaluator for both algorithms and drag into Knowledge Flow Layout space.****

**Step 6:** Now go to Visualization tab and in that select and drag Model Performance Chart. The image will look like below. ****

**Step 7:** Now we have to connect all icons with proper input values. Right click on ArffLoader and select Dataset.

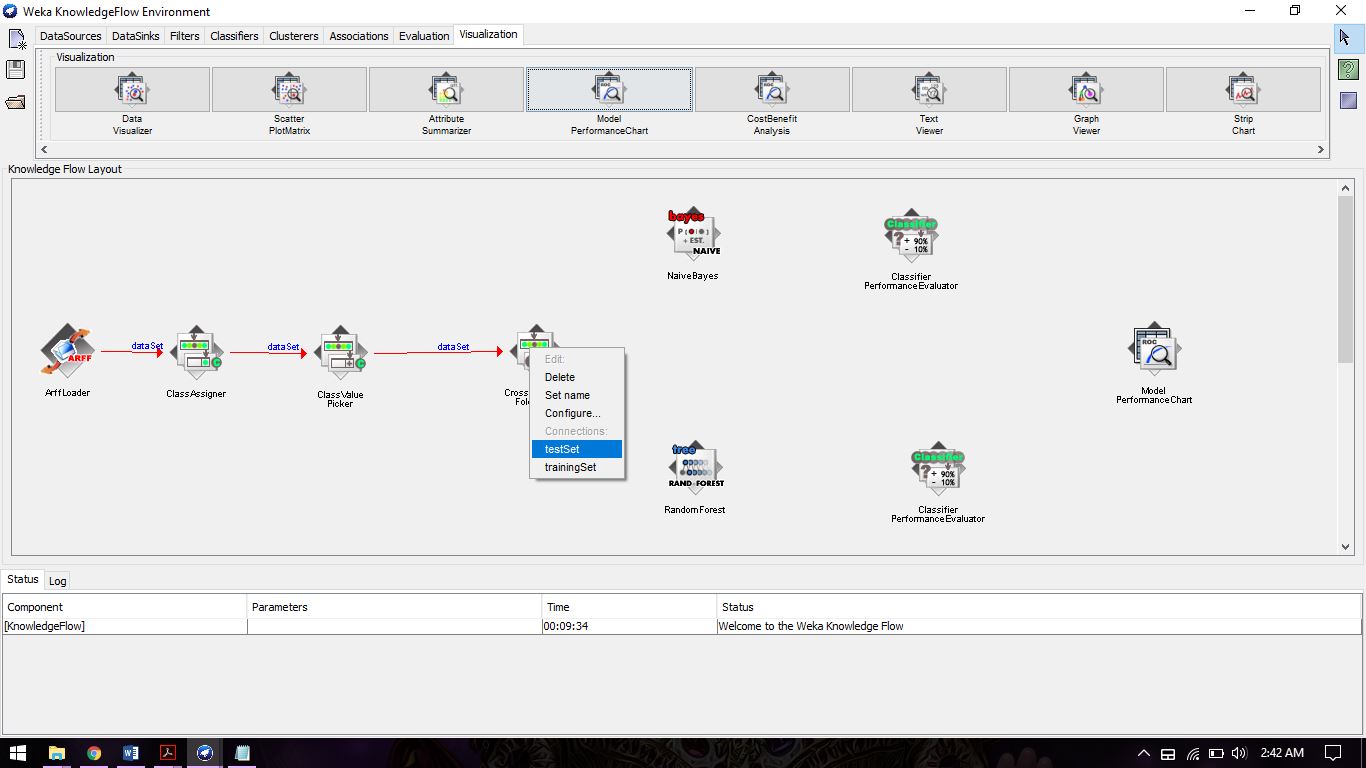
****

**Step 8:** After Select dataset connect it with classAssigner.

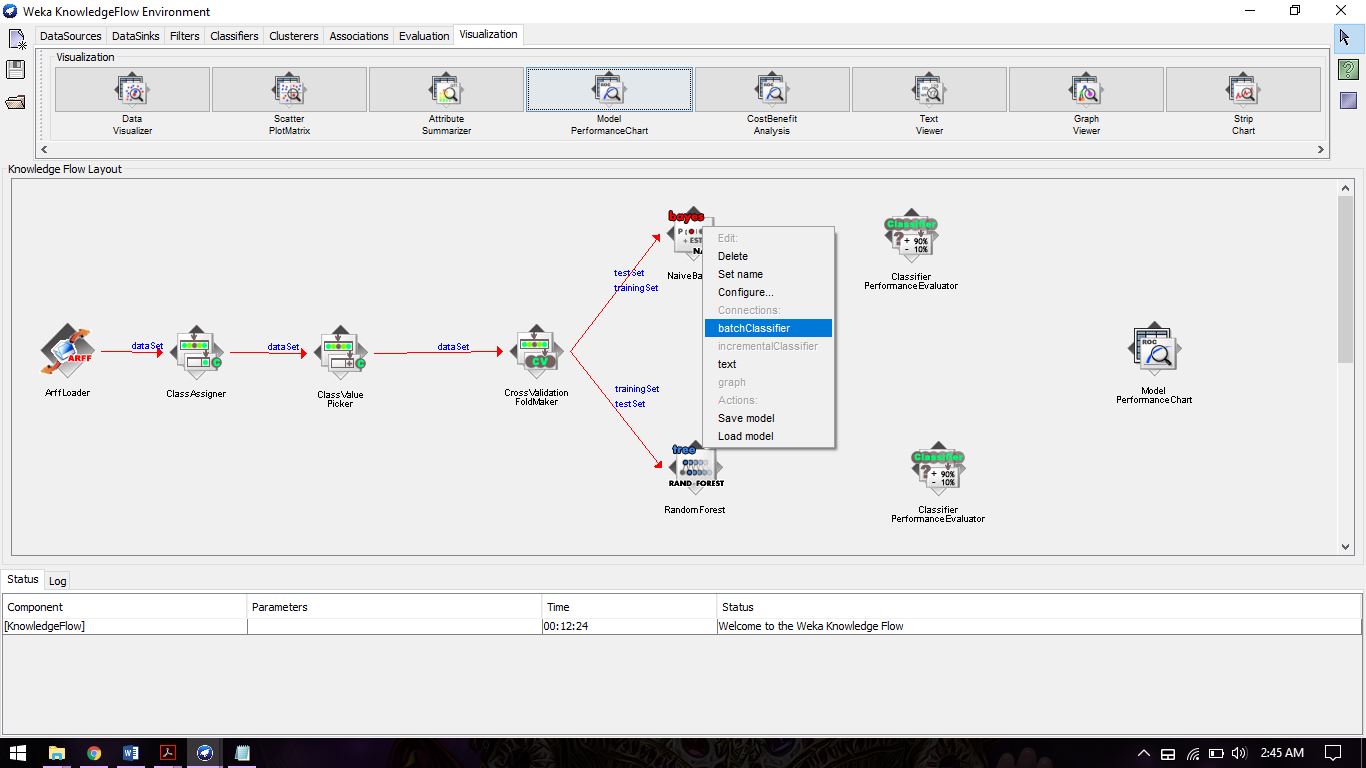
**Step 9:** do same step for next two icon.

**Step 10:** Right click on CrossValidationFoldMaker.

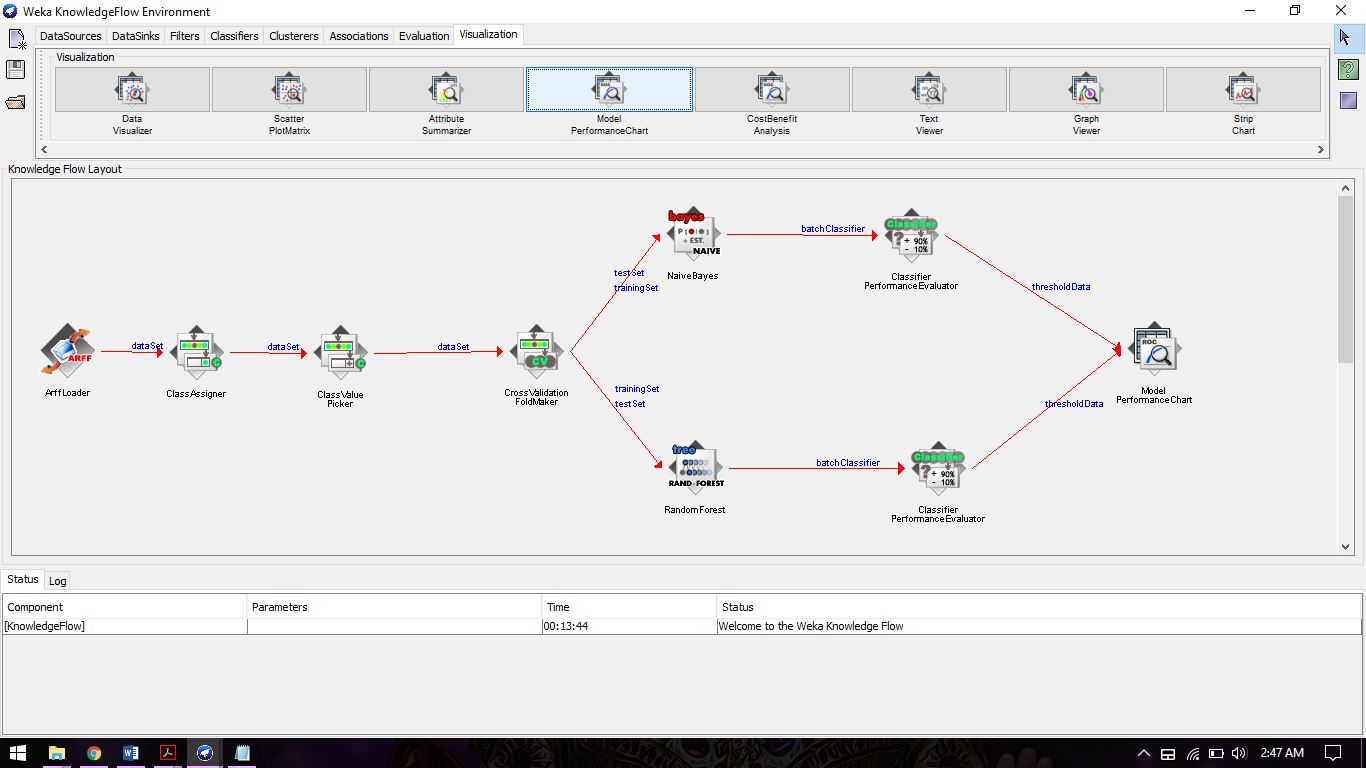
**Step 11:** select training set as well as test set and connect with Naïve Bayes and Random forest icon.

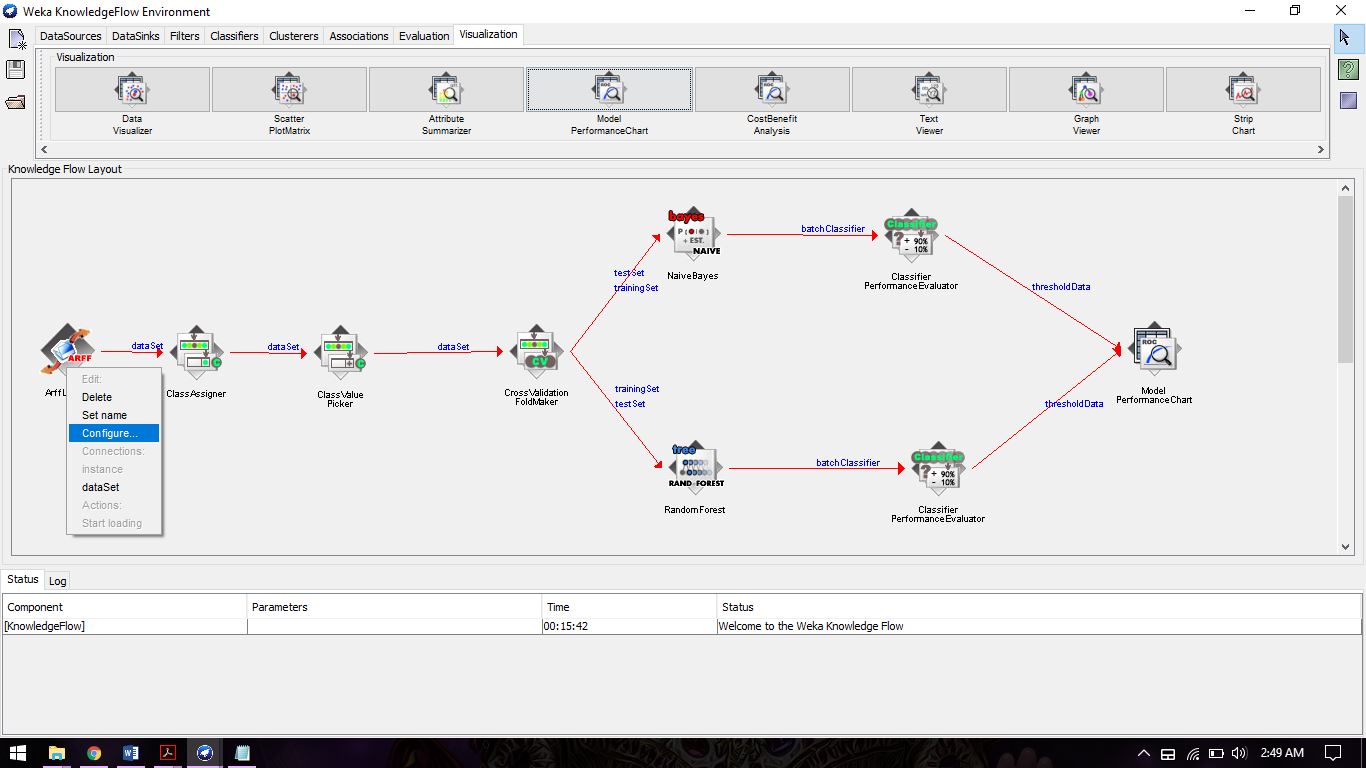
****

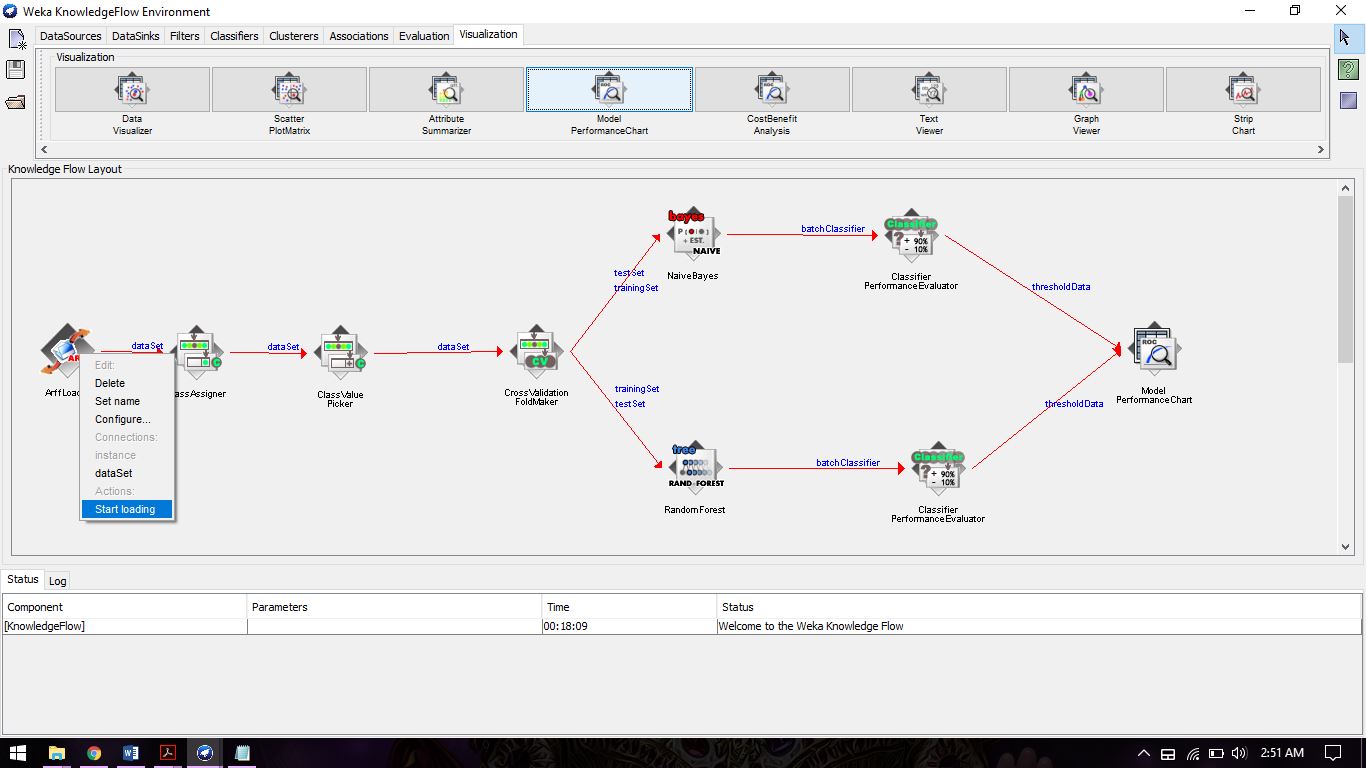
**Step 12:** Right click on Naïve Bayes and select batchClassifier and connect with ClassifierPerformanceEvaluator, Do same step for Random Forest icon too.

****

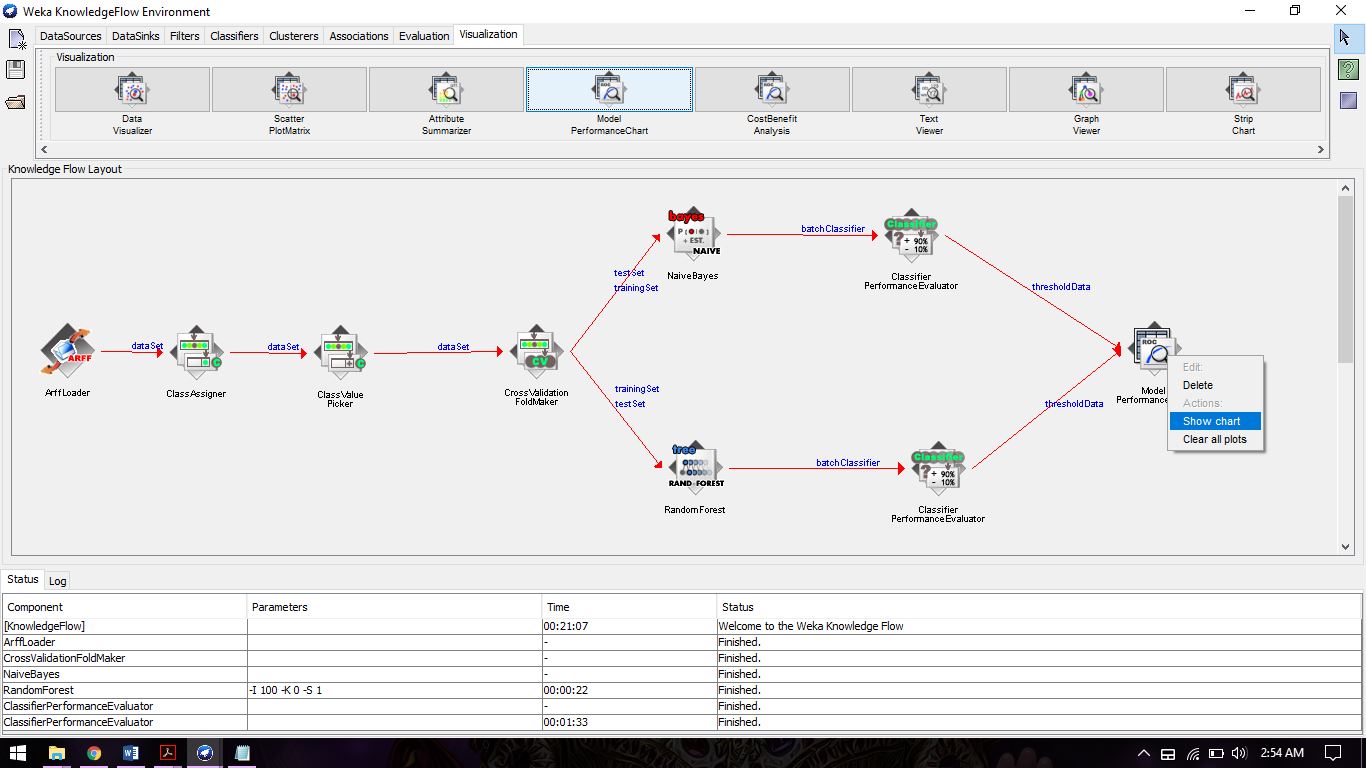
**Step 13:** Right click on ClassifierPerformanceEvaluator and from the popup menu select thresholdData then connect with ModelPerformanceChart icon. Do same step for other icon too.

****

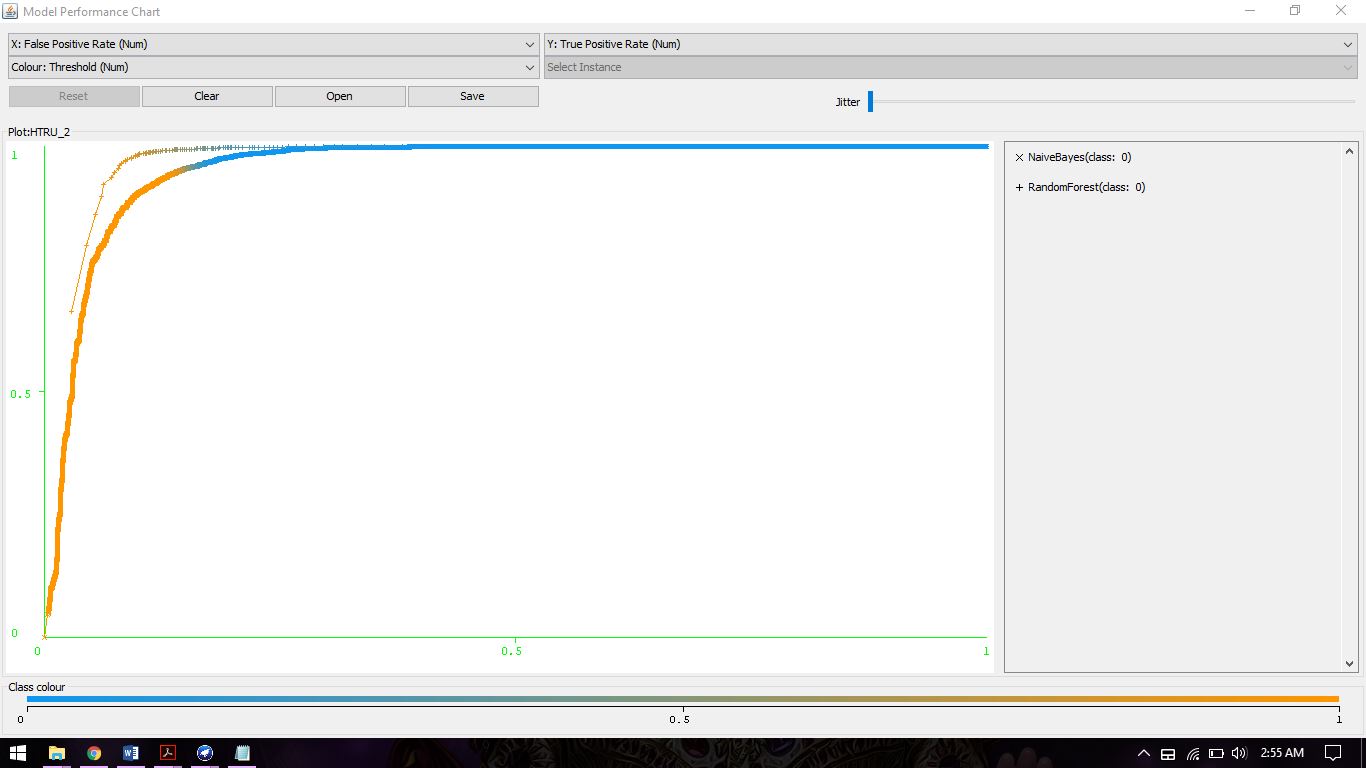
**Step 14:** Right click on ArffLoader. Select Configure and Import Dataset from the local system.****

**Step 15:** Right Click on ArffLoader and Select Start Loading.****

**Step 16:** After showing Finished status for both algorithms right click on Model Performance Chart icon. now select Show Chart.

****

**Step 17:** Roc curve for both classification algorithms will look like below image.

****

**10) Conclusion:** The results show that Random forest classifier has “V” sign which means it has higher percentage of accuracy than Naive Bayes classifier for a classification type on given dataset.

**11) Links:**

**Naive Bayes Classifier (Source Code):**

<https://svn.cms.waikato.ac.nz/svn/weka/branches/stable-3-6/weka/src/main/java/weka/classifiers/bayes/NaiveBayes.java>

**Random Forest (Source Code):**

<https://svn.cms.waikato.ac.nz/svn/weka/branches/stable-3-6/weka/src/main/java/weka/classifiers/trees/RandomForest.java>

**Multiple Roc Curve:**

<https://www.youtube.com/watch?v=rZHw3gGe7DA>