Paper 157-2011

# Data mining in SAS® with open source software

Zhengping Ma, Eli Lilly and Company

## ABSTRACT

It is common in many industries for data to exist in SAS format. Statisticians are often more familiar with the SAS programming environment in comparison to other systems. Therefore, SAS Enterprise Miner is usually the first choice for data mining projects under such circumstances. On the other hand, WEKA is a data mining suite which uses open source code and is available free of charge. More importantly, it offers opportunities to modify the source code for algorithm customization. It also re-implements many classic data mining algorithms, including C4.5 which is called J48 in WEKA. An additional advantage of using WEKA is that it can run hundreds of variations of a model with command mode or customized coding, which is often necessary in research and key for developing reusable tools.

This paper presents ways to connect SAS to WEKA and R and to run packages defined outside SAS for data mining projects. It enables us to develop tools for tasks such as subgroup identification utilizing features available in WEKA, rattle, etc. In addition, it helps to reduce cost by using free of charge open source packages for exploratory analysis.

## INTRODUCTION

Today, there are many options available to choose from as tools for data mining projects. In many industries such as pharmaceuticals, SAS is commonly used for data analysis. As a result of this, data is often available in SAS format. Most statisticians are highly familiar with the SAS programming environment. With its state-of-art packages, SAS Enterprise Miner is usually the tool of choice for data mining projects, especially when SAS is the preferred software for data analysis. SAS EM comes with an easy-to-use graphical user interface. When necessary, it is also possible to do some customization with extension nodes.

Alternatively, WEKA, as an open source data mining tool with perhaps the largest number of algorithms of any data mining tool, is available free of charge. Since it is open source software, it is easy to modify its source code and recompile for algorithm customization in WEKA. It is also very straightforward to run many variations of similar models using the same algorithm with a simple loop.

Rattle is another data mining application built upon R. While an understanding of R is not required in order to use Rattle, it does empower a user to conduct more sophisticated data mining projects by taking advantage of all features available in R. Rattle is simple to use, quick to deploy, and allows a user to rapidly work through the steps from data preparation, modeling, to evaluation in a data mining project. On the other hand, R provides a very powerful platform for performing data mining well beyond the limitations that are embodied in any graphical user interface. A significant benefit of using rattle is that when we can easily migrate from rattle to R to fine tune and further develop our data mining projects when it's necessary. Migration into R is very straightforward since rattle exposes all of the underlying R code in its log window, which can be either directly deployed within R, or saved in R scripts for future reference. These tool generated R scripts can also be modified and used as a tool for similar data mining projects in the future.

For a variety of reasons, sometimes it is desirable that we can use SAS as a working environment and are able to take advantage of tools such as WEKA, rattle, and R in general at the same time. This paper presents ways to connect SAS with WEKA and R, use algorithms defined in WEKA/R package and return the results back into SAS for further processing. The recently rolled-out SAS/IML studio 3.2 comes with a number of useful new features such as calling SAS, R etc. We will use it as a tool for general SAS programming and connecting to external tools/systems.

## CONFIGURE SAS/IML STUDIO FOR WEKA CLASSES

Unfortunately, as of today, there is no direct connection between SAS and WEKA available. However, since both SAS/IML studio and WEKA are commonly built upon Java technology, we can use WEKA classes from SAS/IML studio as user defined classes with Java as the bridge.

Below are steps to configure SAS/IML studio so that it is able to find and load the class definitions of WEKA classes and use them within SAS/IML studio. Please refer to appendix 4 for more details with visual illustration.

1. Download WEKA from http://www.cs.waikato.ac.nz/ml/weka/, save the file weka.jar to the desired location of your choice.
2. Add the path to file weka.jar into SAS/IML Studio by clicking on tools, then options
3. Click on 'Directories' tab, from the 'show directories for' drop down list, select 'Classes'

4. Click the 'Add' button, then choose 'File'

5. Navigate to the location of your choice where file weka.jar is stored, then click on 'weka.jar', and then click the 'Open' button

6. Click the 'OK'.

## USING WEKA CLASSES IN SAS/IML STUDIO

With the appropriate set-up illustrated above, you are ready to take advantage of features in WEKA. However, using WEKA classes in SAS/IML studio directly requires some knowledge of java programming. Typically, you will need to import the relevant packages before you can use methods defined for classes, or create instances for model building.

The first step in a data mining project is to load/prepare data. Below is sample code for this step:

```
/* import WEKA classes, J48 is used as an example, import as many as needed */
import weka.classifiers.trees.J48;

/* Read file into data object */
declare BufferedReader reader = new BufferedReader(new FileReader("c:\weka\weka-3-
6-2\data\iris.arff"));
declare Instances mydata = new Instances(reader);
reader.close();
mydata.setClassIndex(mydata.numAttributes() - 1);
```

It is also very important for a data miner to know his/her data well before any model is applied to data. WEKA's graphic user interface comes with visualization tools to allow the user to see the distribution of all variables. Even though you can also use these features programmatically, it is much easier to use the GUI so that you can see those visualizations interactively.

Assuming that you decide to use the classical tree algorithm, which is implemented in WEKA class J48, below is the sample code:

```
/* create  model builder, an instance of J48 */
declare J48 myJ48=new J48();
myJ48.setUnpruned(true);
/* build model based on data */
myJ48.buildClassifier(mydata);
```

After a model has been built based on training data, you can do cross-validation, make prediction for new data based on the model, etc.

```
declare Instances test = ConverterUtils$DataSource.read("c:/weka/test.csv");
test.setClassIndex(test.numAttributes()-1);

/*evaluate model on test data */
declare EvaluationUtils evu= new EvaluationUtils();
declare weka.core.FastVector predics = evu.getTestPredictions((Classifier)myJ48,test);

/* get predicted probilities */
declare double [][] prob = new double [predics.size()][3];
declare NominalPrediction prd;
do i=0 to prob.length-1;
  do j=0 to 2;
      prd = (NominalPrediction)predics.elementAt(i);
      prob[i][j] = prd.distribution()[j];
  end;
end;
```

Those predicted probabilities can be used to create ROC curve. The details of programming in SAS/IML studio are beyond the scope of this paper. However, you can check the attached sample code for an example, which also demonstrates how to use SAS data steps to create simulation data for model evaluation/prediction, and how to use packages in R to draw ROC curves based on model predictions. You can also refer to SAS/IML help documents, or documents of WEKA API for more details.

## USING RATTLE/R IN SAS/IML STUDIO

One of the new features within the SAS/IML studio 3.2 is that it offers the capability to connect to R with a submit block. Since rattle is an R based tool, we can use this feature directly to run data mining projects with rattle from SAS, as the following code segment illustrates:

```
submit / R;

#your R code goes here

endsubmit;
```

There are two different ways to send data to R. The first is by using the method defined in ExportDataSetToR, which uses SAS dataset as source data, as illustrated below:

```
run ExportDataSetToR("work.mySASdata", "myRdata" );
```

The other is use ExportToR method defined in DataObject, which uses a DataObject as source, as illustrated below:

```
declare DataObject dobj;
dobj = DataObject.CreateFromFile( "myDataFile" );
dobj.ExportToR( "myRdataFrame" );
```

Similarly, there are two different ways to get data from R, we can either use ImportDataFromR module, as

```
run ImportDataSetFromR( "WORK.mySASdataset", "myRdataFrame" );
```

Or use CreateFromR method defined in DataObject, as illustrated below:

```
declare DataObject dobj;
dobj = DataObject.CreateFromR( "SAS_IML_DataObject", "R_DataFrame" );
```

As illustrated above, submitting code to R for processing is very straightforward. For example, with package rattle installed into R, we can send the following code to R to prepare the data:

```
submit / R;
# Load the data.
crs$dataset <- read.arff("file:///C:/weka/weka-3-6-2/data/iris.arff")

# Partition the source data into the training/validate/test datasets
set.seed(42)
crs$sample <- crs$train <- sample(nrow(crs$dataset), 105)
crs$validate <- sample(setdiff(seq_len(nrow(crs$dataset)), crs$train), 22)
crs$test <- setdiff(setdiff(seq_len(nrow(crs$dataset)), crs$train), crs$validate)
endsubmit;
```

We can then add following code into the same submit block to build a decision tree model based on the data:

```
crs$rpart <- rpart(class ~ .,
    data=crs$dataset[crs$train, ], method="class", parms=list(split="information"),
    control=rpart.control(minsplit=5, minbucket=2, usesurrogate=0, maxsurrogate=0))
```

And make predictions based on the model with following code:

```
p<-predict(crs$rpart, newdata=test, type="prob")
```

Again, the details of R programming are beyond the scope of this paper. Please refer to appendix 2 for sample code, which is a modified copy created by rattle with it graphic user interface. It contains sample code for loading data, building a decision tree model, making model predictions, etc.

## USING WEKA THROUGH RWEKA FROM SAS/IML STUDIO

It should also be mentioned that there is an R package called RWeka, which serves as a bridge between WEKA and R and enables a user to use WEKA from R. Therefore, WEKA can also be used from SAS in an indirect way through RWeka, similar to how we use rattle, as described above. Note that you need to have both rJava and RWeka packages installed in your R installation before you can use them from SAS/IML studio. A sample code with some use cases of different models is also attached below, containing examples from loading data, building model to making model predications and creating ROC curves. In addition, it also provides examples of building models based on other algorithms. Overall, if R is a programming language you are comfortable with, it is an easier alternative of using WEKA for statisticians, since it hides some lower level technical details and does not require detailed

knowledge of WEKA classes. Please refer to appendix 3 for more details. Given the indirect approach we used here, from SAS/IML studio to R, and to WEKA through RWeka, this example can also serve as a reference of how to use WEKA from R if your preferred working environment is R instead of SAS.

I would like to point out that the techniques introduced in this paper can be applied in general to connect SAS to any java based and R based external tools, including using any user defined java classes or R functions for any project beyond the scope of the data mining field. It can be very useful to take advantage of past work, or share work completed in different systems with your colleagues. It is also worth mentioning that SAS/IML studio comes with powerful tools for graphics programming to create dynamically linked plots. By pulling data into IML studio from external environments, we are able to create customized visualizations of your results within SAS/IML studio.

## RESULTS FROM THE SAMPLES

We have shown how to prepare data, build models, and make predictions for new data based on the models built, with training data 'iris' which is included in WEKA. Note that the test data is created with SAS data step in the first sample code attached. Figure 1 shows the decision tree built with WEKA's J48, and ROC curve is created based on model predictions for simulated test data. Figure 2 shows the decision tree and ROC curve from rattle solution.

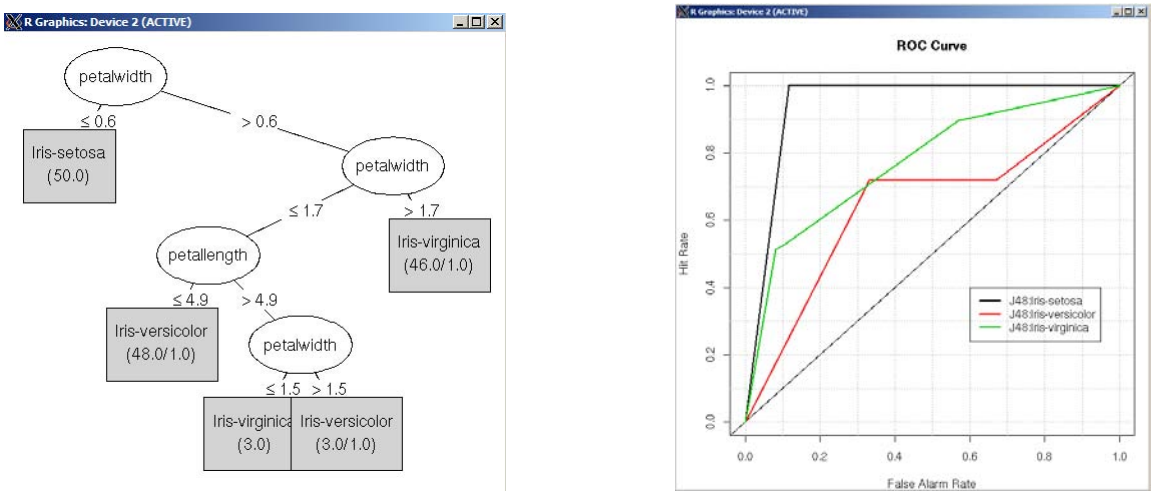



**Figure 1. Decision tree and ROC curves from WEKA/RWeka**

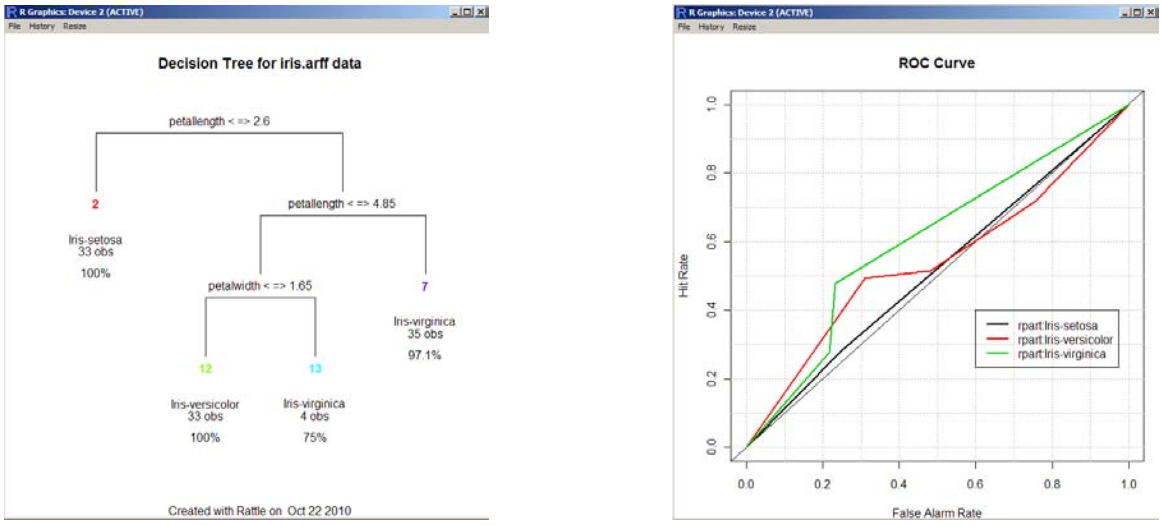



**Figure 2. Decision tree and ROC curves from rattle**

As expected, we can see that both the decision tree and ROC curve from calling WEKA directly from SAS/IML studio exactly match with the results from calling RWeka. The difference between results from rattle and results from WEKA are mainly due to the partitioning used in rattle solution and different default options in two algorithms.

## CONCLUSION

The recently rolled-out SAS/IML studio 3.2 has useful new features, such as submitting code to SAS engine to use other SAS procedures and data step, to R for any features available in R packages, or directly using user defined java classes. It enables us to re-use past work, even in other languages, or utilize new features available in open source software. For data mining projects when SAS Enterprise Miner is not available, or when it is necessary either to customize algorithms or build customized tools, open source software such as WEKA and rattle are very powerful alternatives. This paper demonstrates techniques to use external features available in WEKA/rattle from SAS. In general, the methods demonstrated in this paper can be very useful tools for a SAS programmer looking to take advantage of new functionalities developed outside SAS, especially when the data is available in SAS but when it is desirable to use powerful macros from past projects in the current one.

## REFERENCES

1) Ian H. Witten & Eibe Frank 2005. Data Mining: Practical Machine Learning Tools and Techniques by Elsevier Inc.

2) Simon Urbanek 2009. How to talk to strangers: ways to leverage connectivity between R, Java and Objective C. Comput Stat 24:3003-311

3) Kurt Hornik, Christian Buchta, Achim Zeileis 2009. Open-source Machine Learning: R meets Weka. Comput Stat 24:225-232

4) Kurt Hornik 2010. Documentation for package 'RWeka' available at http://cran.r-project.org/web/packages/RWeka/RWeka.pdf

5) WEKA API documentation available at http://weka.sourceforge.net/doc/

6) Graham Williams 2010. Documentation for package 'rattle' available at http://cran.r-project.org/web/packages/rattle/rattle.pdf

## ACKNOWLEDGMENTS

I'd like to thank Ilya Lipkovich and Zhiyan Ma for their review and suggestions for improvement.

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Zhengping Ma
Enterprise: Eli Lilly and Company
Address: Lilly Corporate Center
City, State ZIP: Indianapolis, IN 46285
E-mail: mazh@lilly.com

## Appendix 1: Sample code of data mining from SAS using WEKA

```
/* import java classes */
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Random;
import java.io.BufferedWriter;
import java.io.FileWriter;

/* import WEKA classes */
import weka.core.Instances;
import weka.classifiers.trees.J48;
import weka.classifiers.meta.FilteredClassifier;
import weka.core.converters.ConverterUtils$DataSource;
import weka.classifiers.evaluation.*;
import weka.classifiers.Classifier;

/* read data file to create a WEKA data object*/
declare BufferedReader reader = new BufferedReader(new FileReader("c:\weka\weka-3-6-
2\data\iris.arff"));
declare Instances mydata = new Instances(reader);
reader.close();
mydata.setClassIndex(mydata.numAttributes() - 1);

/* create a WEKA classifier based on J48 */
declare J48 myJ48=new J48();
myJ48.setUnpruned(false);
myJ48.buildClassifier(mydata);

/* check results */
print(myJ48.graph());

/* create a dataset for predicting */
submit;
data test (drop=i);
format class $15.;
do i=1 to 10000;
       sepallength=5.84+0.83*normal(34569);
       sepalwidth=3.05+0.43*normal(34569);
       petallength=3.76+1.76*normal(34569);
       petalwidth=1.2+0.76*normal(34569);
       if petalwidth <=0.6 and ranbin(-1,1,0.5) then class='Iris-setosa';
       else if petalwidth<=1.7 and ranbin(-1,1,0.5) then do;
              if petallength<=4.9 and ranbin(-1,1,0.5) then class='Iris-versicolor';
              else if petalwidth<=1.5 and ranbin(-1,1,0.5) then class='Iris-virginica';
              else  class='Iris-versicolor';
       end;
       else class='Iris-virginica';
       output;
end;
run;

/* move variable class to be the last variable in the dataset*/
data test1; set test(keep=class); run;
data test2; set test(drop=class); run;
data test; merge test2 test1; run;
endsubmit;
declare DataObject testData;
testData= DataObject.CreateFromServerDataSet("work.test");

/*send dataset to R and save them as csv file*/
run ExportDataSetToR("work.test", "test" );
submit / R;
```

```
write.csv(test, file = "c:/weka/test.csv", row.names = FALSE)
summary(test)
endsubmit;

declare Instances test = ConverterUtils$DataSource.read("c:/weka/test.csv");
test.setClassIndex(test.numAttributes()-1);

/*evaluate model on validation data */
declare EvaluationUtils evu= new EvaluationUtils();
declare weka.core.FastVector predics = evu.getTestPredictions((Classifier)myJ48,test);

/* get prediction probilities */
declare double [][] prob = new double [predics.size()][3];
declare NominalPrediction prd;
do i=0 to prob.length-1;
  do j=0 to 2;
      prd = (NominalPrediction)predics.elementAt(i);
      prob[i][j] = prd.distribution()[j];
  end;
end;

/* Send prediction probabilities to R and create ROC */
run ExportMatrixToR( prob, "preds" );
submit / R;
#combine data with predictions
test.preds <- cbind(test,preds)
test.preds$class <- sub(" +$", "", test.preds$class)

#create binary values for each categories
test.preds$c1 <- ifelse(test.preds$class=="Iris-setosa", 1, 0)
test.preds$c2 <- ifelse(test.preds$class=="Iris-versicolor", 1, 0)
test.preds$c3 <- ifelse(test.preds$class=="Iris-virginica", 1, 0)

# package verification is needed for ROC
library('verification')
mod2 <- verify(test.preds$c2, test.preds$A2, bins = FALSE)
mod3 <- verify(test.preds$c3, test.preds$A3, bins = FALSE)
mod1 <- verify(test.preds$c1, test.preds$A1, bins = FALSE)

roc.plot(mod1, plot.thres = NULL)
lines.roc(mod2, col = 2, lwd = 2)
lines.roc(mod3, col = 3, lwd = 2)
leg.txt <- c("J48:Iris-setosa", "J48:Iris-versicolor", "J48:Iris-virginica" )
legend( 0.6, 0.4, leg.txt, col = c(1,2,3), lwd = 2)
endsubmit;
```

## Appendix 2: Sample code to run rattle (modified from rattle generated code)

```
# Load package rattle
library(rattle)
# The colorspace package is used to generate the colours used in plots.
library(colorspace)
# The 'foreign' package provides the 'read.arff' function.
require(foreign, quietly=TRUE)

# Load an ARFF file.
crs$dataset <- read.arff("file:///C:/weka/weka-3-6-2/data/iris.arff")
test <- read.arff("file:///C:/weka/test.arff")

# Build the training/validate/test datasets.
set.seed(42)
crs$sample <- crs$train <- sample(nrow(crs$dataset), 105)
crs$validate <- sample(setdiff(seq_len(nrow(crs$dataset)), crs$train), 22)
crs$test <- setdiff(setdiff(seq_len(nrow(crs$dataset)), crs$train), crs$validate)
# The following variable selections have been noted.
crs$input <- c("sepallength", "sepalwidth", "petallength", "petalwidth")
crs$target <- "class"
crs$risk <- NULL
crs$ident <- NULL
crs$ignore <- NULL

# The 'rpart' package provides the 'rpart' function for decision tree
require(rpart, quietly=TRUE)
# Reset the random number seed to obtain the same results each time.
set.seed(42)

# Build the Decision Tree model.
crs$rpart <- rpart(class ~ .,
    data=crs$dataset[crs$train, ], method="class", parms=list(split="information"),
    control=rpart.control(minsplit=5, minbucket=2, usesurrogate=0, maxsurrogate=0))

# Plot the resulting Decision Tree using Rattle and maptools support functions.
drawTreeNodes(crs$rpart)
title(main="Decision Tree for iris.arff data", sub=paste("Created with Rattle on ",
format(Sys.time(), "%b %d %Y")))

# Obtain the response from the Decision Tree model.
crs$pr <- predict(crs$rpart, crs$dataset[crs$validate,], type="class")

# Generate the error matrix showing counts.
table(crs$pr, crs$dataset[crs$validate,]$class, dnn=c("Predicted", "Actual"))

#Generate predictions for test data
p<-predict(crs$rpart, newdata=test, type="prob")
x <- test[,5]
c1 <- ifelse(x=="Iris-setosa", 1, 0)
c2 <- ifelse(x=="Iris-versicolor", 1, 0)
c3 <- ifelse(x=="Iris-virginica", 1, 0)

#load package verification
library('verification')
mod1 <- verify(c1, p[,1], bins = FALSE)
mod2 <- verify(c2, p[,2], bins = FALSE)
mod3 <- verify(c3, p[, 3], bins = FALSE)
roc.plot(mod1, plot.thres = NULL)
lines.roc(mod2, col = 2, lwd = 2)
lines.roc(mod3, col = 3, lwd = 2)
leg.txt <- c("rpart:Iris-setosa", "rpart:Iris-versicolor", "rpart:Iris-virginica" )
legend( 0.6, 0.4, leg.txt, col = c(1,2,3), lwd = 2)
```

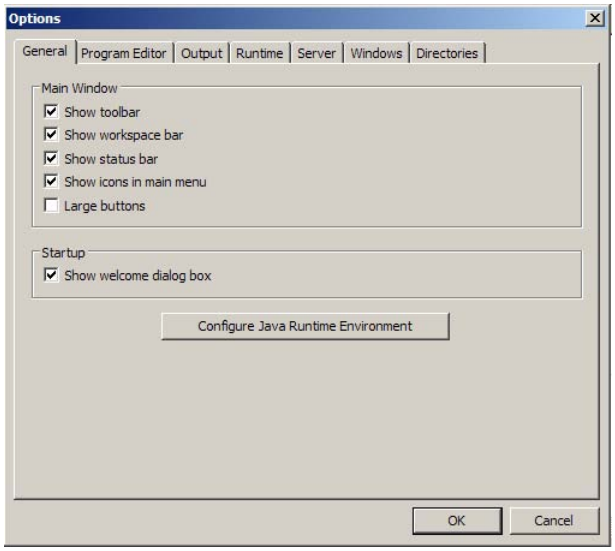
## Appendix 3: Sample code to run RWeka

```
submit / R;
#load packages needed for RWEka
library("rJava")
library("RWeka")

#load the data file
iris <- read.arff(system.file("arff", "iris.arff", package="RWeka"))

#build classifier with J48
m1 <- J48(class ~ ., data = iris)

# visualization use party package
if(require("party", quietly = TRUE)) plot(m1)

#evaluate model with 10 fold cross-validation
e<-evaluate_Weka_classifier(m1, numFolds=10, complexity=FALSE, seed=258, class=TRUE)
e

#evaluate model with new data
test<-read.arff("/home/c052308/R/test.arff")
e<-evaluate_Weka_classifier(m1, newdata=validate, complexity=FALSE, seed=258,
class=TRUE)
e

#predicted values for test data
p<-predict(m1, newdata=test, type="probability")
p
x <- iris[,5]
c1 <- ifelse(x=="Iris-setosa", 1, 0)
c2 <- ifelse(x=="Iris-versicolor", 1, 0)
c3 <- ifelse(x=="Iris-virginica", 1, 0)

# package verification is needed for ROC
library("verification")
mod1 <- verify(c1, p[,1], bins = FALSE)
mod2 <- verify(c2, p[,2], bins = FALSE)
mod3 <- verify(c3, p[, 3], bins = FALSE)
roc.plot(mod1, plot.thres = NULL)
lines.roc(mod2, col = 2, lwd = 2)
lines.roc(mod3, col = 3, lwd = 2)
leg.txt <- c("J48:Iris-setosa", "J48:Iris-versicolor", "J48:Iris-virginica" )
legend( 0.6, 0.4, leg.txt, col = c(1,2,3), lwd = 2)

# Learn J4.8 tree with reduced error pruning (-R) and minimum number of instances set
to 5 (-M 5):
m2 <-J48(class ~ ., data = iris, control = Weka_control(R = TRUE, M = 5))
p<-predict(m2, type="probability")

# using M5P
cpudata  <- read.arff(system.file("arff", "cpu.arff", package = "RWeka"))
m3 <- M5P(class ~ ., data = cpudata  )

# Logistic Model Tree.
weather  <- read.arff(system.file("arff", "weather.arff", package = "RWeka"))
m4 <- LMT(play ~ ., data = weather  )
table(weather  $play, predict(m4))
endsubmit;
```

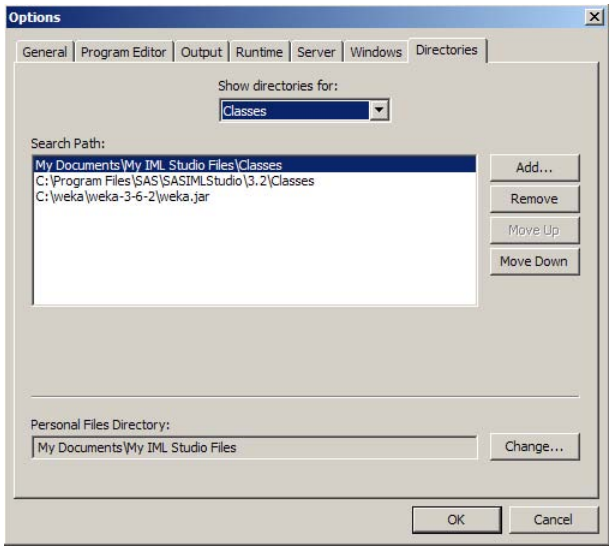
## Appendix 4: Steps to configure SAS/IML studio with visual illustration

Below are steps to configure SAS/IML studio so that it is able to find and load the class definitions of WEKA classes:

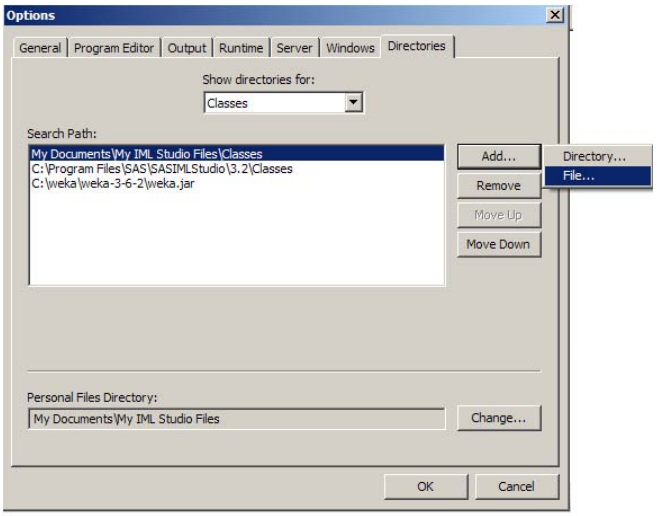
1. Download WEKA from http://www.cs.waikato.ac.nz/ml/weka/, save the file weka.jar to the desired location of your choice.
2. Add the path to file weka.jar into SAS/IML Studio by clicking on tools, then options



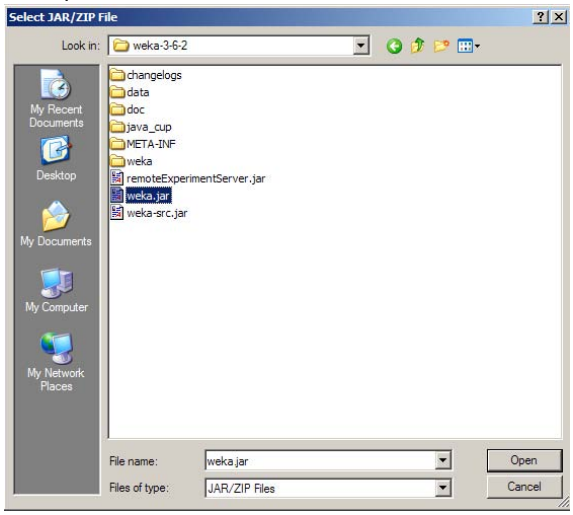

3. Click on 'Directories' tab, from the 'show directories for' drop down list, select 'Classes'



4. Click the ‘Add’ button, then choose ‘File’




5. Navigate to the location of your choice where file weka.jar is stored, then click on ‘weka.jar’, and then click the ‘Open’ button




6. Click the ‘OK’ button