FIT3152 Data analytics. Tutorial 09: Classification: Ensemble methods, ANNs

Topics Covered:

- Ensemble classifiers, Artificial Neural Networks

References:
Alfaro, Gámez and García, adabag: An R Package for Classification with Boosting and Bagging., Journal of Statistical Software, 54(2) 2013
Breiman, Random Forests. Machine Learning 45, 5 – 32, 2001.
James et al., An Introduction to Statistical Learning with Applications in R. Springer. Chapter 8.
Günther and Fritsch, neuralnet: Training of Neural Networks, The R Journal Vol. 2/1, June 2010.
Reference Manuals to each of the packages used (listed below), available from CRAN.

1 Work through the examples in the lecture slides. For the examples using R you could download and install the packages using the script below.

```
# set working directory to desktop
setwd("~/Desktop")
# clean up the environment before starting
rm(list = ls())
install.packages("tree")
library(tree)
install.packages("e1071")
library(e1071)
install.packages(("ROCR"))
library(ROCR)
install.packages("randomForest")
library(randomForest)
install.packages("adabag")
library(adabag)
install.packages("rpart")
library(rpart)
```

2 Using the references above, in addition to your lecture notes and any other references you locate write short answers to the following:

a. Describe the general philosophy behind, ensemble classifiers and the way they work. That is, what do all the ensemble classifiers we studied have in common?

b. Describe the similarities and differences between each of the ensemble classifiers studied: boosting, bagging, random forests. In particular, you should be aware of how each classifier is unique within the group.

3 The Japanese credit data “JapaneseCredit.csv” has 690 records relating to credit card applications, and whether they were approved or not. All attribute names and values have been anonymised. Ref. http://archive.ics.uci.edu/ml/datasets/Japanese+Credit+Screening

There are a variety of attribute types: continuous, and nominal (some with a small number of different values, and some with large). There are also some missing values that have been

indicated as NA. The class value indicates "+" or "-" indicating whether or not a credit applicant was approved.

(a) Split the data into a 70% training and a 30% test set and create a classification model using each of the following techniques in turn:

- Decision Tree
- Naïve Bayes
- Bagging
- Boosting
- Random Forest

(b) Using the test data, classify each of the test cases into "+" or "-". Create a confusion matrix for each and report the accuracy of each model.

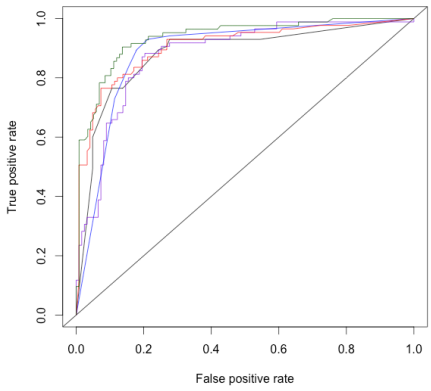
(c) Now, calculate the confidence of predicting a "+" outcome for each of the test cases and construct an ROC curve for each classifier. You should be able to plot all the curves on the same axis. Use a different colour for each classifier.

What does the ROC curve tell you about each of the classifiers. Is there a single "best" classifier?

(d) Examining each of the models, determine the most important variables in predicting whether or not an applicant would be granted a loan.

```
set.seed(9999) #random seed
JC <- read.csv("JapaneseCredit.csv", stringsAsFactors = T)
# delete rows with missing values
JC = JC[complete.cases(JC),]
train.row = sample(1:nrow(JC), 0.7*nrow(JC))
JC.train = JC[train.row,]
JC.test = JC[-train.row,]

# Calculate a decision tree
JC.tree = tree(Class ~., data = JC.train)
#print(summary(JC.tree))
plot(JC.tree)
text(JC.tree, pretty = 0)

# do predictions as classes and draw a table
JC.predtree = predict(JC.tree, JC.test, type = "class")
t1=table(Predicted_Class = JC.predtree, Actual_Class = JC.test$Class)
cat("\n#Decision Tree Confusion\n")
print(t1)

# do predictions as probabilities and draw ROC
JC.pred.tree = predict(JC.tree, JC.test, type = "vector")
# computing a simple ROC curve (x-axis: fpr, y-axis: tpr)
# labels are actual values, predictors are probability of class
JCDpred <- prediction( JC.pred.tree[,2], JC.test$Class)
JCDperf <- performance(JCDpred,"tpr","fpr")
plot(JCDperf)
abline(0,1)

#####################################################################

# Calculate naive bayes
JC.bayes = naiveBayes(Class ~. , data = JC.train)
```

```
JC.predbayes = predict(JC.bayes, JC.test)
t2=table(Predicted_Class = JC.predbayes, Actual_Class = JC.test$Class)
cat("\n#NaiveBayes Confusion\n")
print(t2)

# outputs as confidence levels
JCpred.bayes = predict(JC.bayes, JC.test, type = 'raw')
JCBpred <- prediction( JCpred.bayes[,2], JC.test$Class)
JCBperf <- performance(JCBpred,"tpr","fpr")
plot(JCBperf, add=TRUE, col = "blueviolet")

#############################################################################

# Bagging
JC.bag <- bagging(Class ~. , data = JC.train, mfinal=5)
JCpred.bag <- predict.bagging(JC.bag, JC.test)
# JCpred.bag
JCBagpred <- prediction( JCpred.bag$prob[,2], JC.test$Class)
JCBagperf <- performance(JCBagpred,"tpr","fpr")
plot(JCBagperf, add=TRUE, col = "blue")
cat("\n#Bagging Confusion\n")
print(JCpred.bag$confusion)


#############################################################################

#Boosting
JC.Boost <- boosting(Class ~. , data = JC.train, mfinal=10)
JCpred.boost <- predict.boosting(JC.Boost, newdata=JC.test)
# JCpred.boost
JCBoostpred <- prediction( JCpred.boost$prob[,2], JC.test$Class)
JCBoostperf <- performance(JCBoostpred,"tpr","fpr")
plot(JCBoostperf, add=TRUE, col = "red")
cat("\n#Boosting Confusion\n")
print(JCpred.boost$confusion)

#############################################################################

# Random Forest
JC.rf <- randomForest(Class ~. , data = JC.train, na.action = na.exclude)
JCpredrf <- predict(JC.rf, JC.test)
t3=table(Predicted_Class = JCpredrf, Actual_Class = JC.test$Class)
cat("\n#Random Forest Confusion\n")
print(t3)

JCpred.rf <- predict(JC.rf, JC.test, type="prob")
# JCpred.rf
JCFpred <- prediction( JCpred.rf[,2], JC.test$Class)
JCFperf <- performance(JCFpred,"tpr","fpr")
plot(JCFperf, add=TRUE, col = "darkgreen")



#Attribute importance
cat("\n#Decision Tree Attribute  Importance\n")
print(summary(JC.tree))
cat("\n#Baging Attribute  Importance\n")
print(JC.bag$importance)
cat("\n#Boosting Attribute  Importance\n")
print(JC.Boost$importance)
cat("\n#Random Forest Attribute  Importance\n")
print(JC.rf$importance)
```



4 The Mushroom data set (mushroom.data.csv) contains 22 pieces of information about 8000+ species of mushrooms. This data set was obtained from the UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/Mushroom

Using the data, construct a decision models using each of the ensemble classifiers (bagging, random forest and boosting) to predict whether an unclassified mushroom is of "class" poisonous or edible based on the other attributes:

Create a training (70%) and test data set and report the accuracy of your model for each of the classifiers.

Which attributes are most important in distinguishing between poisonous and edible mushrooms? (Look at variable importance measures etc.) Is this consistent for all of the classifiers?

Which classifier works best?

```
M <- read.csv("mushroom.data.csv")

#random seed
set.seed(9999)
train.row = sample(1:nrow(M), 0.7*nrow(M))
M.train = M[train.row,]
M.test  = M[-train.row,]

# Fit a decision tree
M.tree = tree(class ~., data = M.train)
#print(summary(M.tree))
plot(M.tree)
text(M.tree, pretty = 0)

# do predictions as classes and draw a table
M.predtree = predict(M.tree, M.test, type = "class")
t1=table(Actual_Class = M.test$class, Predicted_Class = M.predtree)
cat("\n#Decsion Tree Confusion\n")
print(t1)

# do predictions as probabilities and draw ROC
M.pred.tree = predict(M.tree, M.test, type = "vector")
MDpred <- prediction( M.pred.tree[,2], M.test$class)
MDperf <- performance(MDpred,"tpr","fpr")
plot(MDperf)
abline(0,1)
```

```
#########################################################################

# Bagging
M.bag <- bagging(class ~. , data = M.train, mfinal=5)
Mpred.bag <- predict.bagging(M.bag, M.test)
# Mpred.bag
MBagpred <- prediction( Mpred.bag$prob[,2], M.test$class)
MBagperf <- performance(MBagpred,"tpr","fpr")
plot(MBagperf, add=TRUE, col = "blue")
cat("\n#Bagging Confusion\n")
print(Mpred.bag$confusion)


#########################################################################

#Boosting
M.Boost <- boosting(class ~. , data = M.train, mfinal=10)
Mpred.boost <- predict.boosting(M.Boost, newdata=M.test)
# Mpred.boost
MBoostpred <- prediction( Mpred.boost$prob[,2], M.test$class)
MBoostperf <- performance(MBoostpred,"tpr","fpr")
plot(MBoostperf, add=TRUE, col = "red")
cat("\n#Boosting Confusion\n")
print(Mpred.boost$confusion)

#########################################################################

# Random Forest
M.rf <- randomForest(class ~. , data = M.train, na.action = na.exclude)
Mpredrf <- predict(M.rf, M.test)
t3=table(Predicted_Class = Mpredrf, Actual_Class = M.test$class)
cat("\n#Random Forest Confusion\n")
print(t3)

Mpred.rf <- predict(M.rf, M.test, type="prob")
# Mpred.rf
MFpred <- prediction( Mpred.rf[,2], M.test$class)
MFperf <- performance(MFpred,"tpr","fpr")
plot(MFperf, add=TRUE, col = "darkgreen")

#Attribute importance
cat("\n#Decision Tree Attribute  Importance\n")
print(summary(M.tree))
cat("\n#Baging Attribute  Importance\n")
print(M.bag$importance)
cat("\n#Boosting Attribute  Importance\n")
print(M.Boost$importance)
cat("\n#Random Forest Attribute  Importance\n")
print(M.rf$importance)
```

5 Using the Japanese credit data fit an artificial neural network using only the numeric attributes to classify whether an applicant was approved or not. Using an 80% training set and 20% test set calculate the accuracy of your model.

You will need to prepare your data before you can fit the ANN. The suggested order of data processing is:

1. Remove rows containing missing values
2. Recode the output Class as numeric
3. Create training and test sets
4. Fit neural network and test accuracy etc.

* Extension. Augment the input using some of the Attributes containing categorical variables. (chose an Attribute with 2 options. For example, A1 has two levels: a and b). You should make indicator columns before creating training and test sets (That is, after Step 2 above)

```
# clean up the environment before starting
rm(list = ls())
#install.packages("neuralnet")
library(neuralnet)
library(car)

options(digits=4)

JCC <- read.csv("JapaneseCredit.csv")
# delete rows with missing values
JCC = JCC[complete.cases(JCC),]
# convert Class to a numerical form
JCC$Class = recode(JCC$Class," '+' = '0';'-' = '1' ")
JCC$Class = as.numeric(JCC$Class)
# make training and test sets
set.seed(9999)
ind <- sample(2, nrow(JCC), replace = TRUE, prob=c(0.8, 0.2))

JCC.train = JCC[ind == 1,]
JCC.test = JCC[!ind == 1,]

JCC.nn = neuralnet(Class ~  A2 + A3 + A8 + A11 + A14, JCC.train, hidden=3)

JCC.pred = compute(JCC.nn, JCC.test[c(2, 3, 8, 11, 14)])
# now round these down to integers
JCC.pred = as.data.frame(round(JCC.pred$net.result,0))

# plot confusion matrix
table(observed = JCC.test$Class, predicted = JCC.pred$V1)

#########################################################################

#Abishek's improved solution
# clean up the environment before starting
rm(list = ls())
options(digits=4)
library(car)
library(neuralnet)
JC <- read.csv("JapaneseCredit.csv")

#remove na
JC = JC[complete.cases(JC),]

#convert into numeric
JC$Class = recode(JC$Class," '+' = '0';'-' = '1' ")
JC$Class = as.numeric(JC$Class)
train.row = sample(1:nrow(JC), 0.8*nrow(JC))
JC.train = JC[train.row,]
JC.test = JC[-train.row,]

#Binomial classification: predict the probability of belonging to class 1
and if the probability is less than 0.5 consider it predicted as class 0

JC.nn = neuralnet(Class == 1 ~  A2 + A3 + A8 + A11 + A14, JC.train,
hidden=3,linear.output = FALSE)
JC.pred = compute(JC.nn, JC.test[c(2, 3, 8, 11, 14)])
prob <- JC.pred$net.result
```

```
pred <- ifelse(prob>0.5, 1, 0)
#confusion matrix
table(observed = JC.test$Class, predicted = pred)
```

6 Using the modified Play Tennis data ("playtennistrainTF.csv" and playtennistestTF.csv) fit an ANN to classify whether a person should play tennis or not. Using the test data, evaluate the accuracy of your model.

You will need to prepare your data by creating indicator columns before you can fit the ANN. You can do this using the 'recode' function but a faster way is by using the 'model.matrix' function. The suggested order of data processing is:

1. Read training data
2. Read test data
3. Create a single file merging training and test data using 'rbind'
4. Create the indicator columns for each attribute using 'recode' or 'model.matrix'
5. Tidy up the recoded data frame or bind the model.matrix to the merged file
6. Separate the merged file into training and test data
7. Expand the training data set to 100 records by resampling
8. Fit the neural network

```
# clean up the environment before starting
rm(list = ls())
#install.packages("neuralnet")
library(neuralnet)

options(digits=4)

pttrainread <- read.csv("playtennistrainTF.csv")
pttestread <- read.csv("playtennistestTF.csv")

ptcombined = rbind(pttrainread,pttestread)

ptmm = model.matrix(~Outlook+Temperature+Humidity+Wind, data=ptcombined)
ptcombined = cbind(ptcombined,ptmm)
ptcombined = ptcombined[,c(1, 8, 9, 10, 11, 12, 13, 6)]

pttest = ptcombined[15:20,]
pttrain = ptcombined[1:14,]

# set.seed(9999) #random seed
# resampling with replacement to create a larger training set
pttrain = pttrain[sample(nrow(pttrain), 100, replace = TRUE),]
pttrain = as.data.frame(pttrain)

PT.nn = neuralnet(Play~ OutlookRain + OutlookSunny + TemperatureHot +
TemperatureMild
                  + HumidityNormal + WindWeak, pttrain, hidden=1)

PT.nn$result.matrix
PT.pred = compute(PT.nn, pttest[,2:7])

PT.predr = round(PT.pred$net.result,0)

table(observed = pttest$Play, predicted = PT.predr)
```