Business Data Analytics. Practice Session
Fraud Detection

1) Using XGBoost for Credit card fraud detection [1]. The class practice session has some additional commands and comments being added for your reference.


```
#Load the libraries
library(ggplot2) # Data visualization
library(readr) # CSV file I/O, e.g. the read_csv function
library(caret) # rich package -- for example in this code confusionMatrix -- for getting directly the confusion matrix values.
library(DMwR) # smote
library(xgboost) # for xgboost
library(Matrix) # you need bcos , you need to create matrix as inputs for xgboost
library(reshape) # for the melt function
library(pROC) # AUC

setwd("Path, where you have the library")
ccdata <- read.csv("creditcard.csv")

#Checking the data. Variables are normalized, except Amount and Time. Remove Time variable
str(ccdata)

# Check if the data is balanced or not ?
table(ccdata$Class)

# TO see in percentage
prop.table(table(ccdata$Class))

ccdata$Time <- NULL # We do not need time.

#Split data into train, cv and test.
set.seed(1900)

# For dividing the data in train and cross validation and test
inTrain <- createDataPartition(y = ccdata$Class, p = .6, list = F) #p: the percentage of data that goes to training
```
#list: logical - should the results be in a list (TRUE) or a matrix with the number of rows equal to 
floor(p * length(y)) and times columns.

```r
train <- ccdata[inTrain,]
testcv <- ccdata[-inTrain,]
inTest <- createDataPartition(y = testcv$Class, p = .5, list = F)
test <- testcv[inTest,]

train$Class <- as.factor(train$Class)
```

rm(inTrain, inTest, testcv) # Removing unwanted objects from memory. Good practice

#SMOTE
#Very imbalanced dataset, so let's see if using smote can improve this model.

```r
train_smote <- SMOTE(Class ~ ., as.data.frame(train), perc.over = 20000, perc.under=100)
```

# perc.over : percentage of over sampling -- 200 % increase of the minority class.
# perc.under : percentage of under sampling -- equivalent to total oversampled data : 100 %
# means equal proportion

```
# Now 2 class are almost equal.

table(train_smote$Class)

# Something to be used for xgboost
i <- grep("Class", colnames(train)) # Get index Class column
# you can simply assign to i <- ncol(train)

#Prepare data for XGBoost and set parameters. Use AUC as evaluation metric, as accuracy does
not make sense for such a imbalanced dataset.

# Back to numeric
train$Class <- as.numeric(levels(train$Class))[train$Class]
train_smote$Class <- as.numeric(levels(train_smote$Class))[train_smote$Class]

# As Matrix
train <- Matrix(as.matrix(train), sparse = TRUE)
train_smote <- Matrix(as.matrix(train_smote), sparse = TRUE)
test <- Matrix(as.matrix(test), sparse = TRUE)
cv <- Matrix(as.matrix(cv), sparse = TRUE)
```

# Create XGB Matrices
train_xgb <- xgb.DMatrix(data = train[,i], label = train[,i])
train_smote_xgb <- xgb.DMatrix(data = train_smote[,i], label = train_smote[,i])
test_xgb <- xgb.DMatrix(data = test[,i], label = test[,i])
cv_xgb <- xgb.DMatrix(data = cv[,i], label = cv[,i])

# Watchlist
watchlist <- list(train = train_xgb, cv = cv_xgb)

# set parameters:
parameters <- list(
  # General Parameters
  booster = "gbtree",
silent = 0,
  # Booster Parameters
  eta = 0.3,
gamma = 0,
  max_depth = 6,
  min_child_weight = 1,
  subsample = 1,
  colsample_bytree = 1,
  colsample_bylevel = 1,
  lambda = 1,
  alpha = 0,
  # Task Parameters
  objective = "binary:logistic",
  eval_metric = "auc",
  seed = 1900
)

# Some explanation
# eta= low -- model is more robust to overfitting
# gamma (default is 0) and with larger values we would like to have a more conservative
# algorithms and thus, we would like to avoid overfitting.
# subsample = lower values helps in overfitting.
# colsample_bytree
# missing = NA;

# Original
xgb.model <- xgb.train(parameters, train_xgb, nrounds = 25, watchlist)

# Plot:
melted <- melt(xgb.model$evaluation_log, id.vars="iter")
# For changing into long format.
ggplot(data=melted, aes(x=iter, y=value, group=variable, color = variable)) + geom_line()
#Try without group = variable ? What happens ?

# Smote
xgb_smote.model <- xgb.train(parameters, train_smote_xgb, nrounds = 25, watchlist)

#Plot:
melted <- melt(xgb_smote.model$evaluation_log, id.vars="iter")
ggplot(data=melted, aes(x=iter, y=value, group=variable, color = variable)) + geom_line()

# Feature importance
imp <- xgb.importance(colnames(train_xgb), model = xgb.model)
print(imp)
# Gain is the improvement in accuracy by a feature to the branches it is on.
xgb.plot.importance(imp)

# Threshold
q <- 0.5

# Original
xgb.predict <- predict(xgb.model, test)
xgb.predictboolean <- ifelse(xgb.predict >= q,1,0)
 roc <- roc(test[,i], predict(xgb.model, test, type = "prob"))
xgb.cm <- confusionMatrix(xgb.predictboolean, test[,i])
xgb.cmstable
print(paste("AUC of XGBoost is:" ,roc$auc))
print(paste("F1 of XGBoost is:" ,xgb.cm$byClass["F1"]))
xgb.cm$byClass

# SMOTE
roc_smote <- roc(test[,i], predict(xgb_smote.model, test, type = "prob"))
xgb_smote.predict <- predict(xgb_smote.model, test)
xgb_smote.predictboolean <- ifelse(xgb_smote.predict >= q,1,0)
xgb_smote.cm <- confusionMatrix(xgb_smote.predictboolean, test[,i])
xgb_smote.cm$stable
print(paste("AUC of SMOTE XGBoost is:" ,roc_smote$auc))
print(paste("F1 of SMOTE XGBoost is:" ,xgb_smote.cm$byClass["F1"]))
xgb_smote.cm$byClass

Related Material
2) For logistic regression, decision tree and random forest see [2]
3) For a basic tutorial on xgboost, look [3]
4) To look at the parameters of xgboost, see [4] or [5]
5) Comparison of xgboost with others, see [6]
7) Learn Gradient Boosting Algorithm for better predictions (with codes in R) [8]
8) Getting smart with Machine Learning – AdaBoost and Gradient Boost [9]

Addition Material

6) Complete Guide to Parameter Tuning in XGBoost (with codes in Python) [7]
9) How to avoid Over-fitting using Regularization? [10]

References:
[3] https://www.youtube.com/watch?v=woVTNwRrFHE
[9] https://www.analyticsvidhya.com/blog/2015/05/boosting-algorithms-simplified/
[10]https://www.analyticsvidhya.com/blog/2015/02/avoid-over-fitting-regularization/