Analysis of the Company Financial Manipulations

Kumar Rahul

20 September 2021

The analysis is on company financial manipulations and devise algorithm to identify a manipulater from a non manipulater based on the financial ratios reported by the companies. There are a total of 1239 observations in the data set. Out of these 1239 observations, there are 1200 non manipulaters and 39 manipulaters.

1. Look for different types of model which can be built using R. Also has a guideline for fine tuning paramters
2. Refer link to know random forest and Refer to know about OOB error
3. Demonstration of some of the bagging and boosting algorithm
4. Understand the logic for bagging in logistic regression
5. Interpret the tree structure generated out of random forest model

---

Preparing data

Read data from a specified location

```
raw.data <- read.csv("/Users/Rahul/Documents/Rahul Office/IIMB/Work @
IIMB/Company Fraud/IMB 575 FRAUD ANALYTICS.csv",head=TRUE,na.strings=c("", "
", "NA"), sep=",")

filter.data <- raw.data[,-c(1)]
```

Define an 70%/30% train/test split of the dataset

```
set.seed(4121)
trainIndex <- createDataPartition(filter.data$Manipulater, p = 0.70,
list=FALSE)
data.train <- filter.data[ trainIndex,]
data.test <- filter.data[-trainIndex,]
```

Prepare and run numerical summaries

```
summary(data.train) #summary of the data

##       DSRI               GMI                 AQI
##  Min.   : 0.0000   Min.   :-20.8118   Min.   :-21.7338
##  1st Qu.: 0.8876   1st Qu.:  0.9253   1st Qu.:  0.7856
##  Median : 1.0200   Median :  1.0000   Median :  1.0079
##  Mean   : 1.1387   Mean   :  0.9778   Mean   :  1.0763
##  3rd Qu.: 1.1872   3rd Qu.:  1.0507   3rd Qu.:  1.2110
##  Max.   :15.3435   Max.   : 46.4667   Max.   : 52.8867
##       SGI                DEPI               SGAI               ACCR
##  Min.   : 0.06454   Min.   :0.06882   Min.   : 0.0000   Min.   :-0.68226
```

```
##  1st Qu.: 0.97341   1st Qu.:0.93554   1st Qu.: 0.9008   1st Qu.:-0.07631
##  Median : 1.09614   Median :1.00000   Median : 1.0002   Median :-0.03004
##  Mean   : 1.13740   Mean   :1.02915   Mean   : 1.1073   Mean   :-0.03045
##  3rd Qu.: 1.20608   3rd Qu.:1.07637   3rd Qu.: 1.1290   3rd Qu.: 0.02016
##  Max.   :13.06465   Max.   :5.39387   Max.   :49.3018   Max.   : 0.95989
##       LEVI          Manipulater C.MANIPULATOR
##  Min.   : 0.0000   No :840     Min.   :0.00000
##  1st Qu.: 0.9232   Yes: 28     1st Qu.:0.00000
##  Median : 1.0133               Median :0.00000
##  Mean   : 1.0574               Mean   :0.03226
##  3rd Qu.: 1.1154               3rd Qu.:0.00000
##  Max.   :13.0586               Max.   :1.00000

data.train <- na.omit(data.train) # listwise deletion of missing
data.test <- na.omit(data.test) # listwise deletion of missing
```

Train and test dataset with needed variables

```
model.data <- as.data.frame(filter.data[,c(#"DSRI",
                                       #"GMI",
                                       "AQI",
                                       #"SGI",
                                       "DEPI",
                                       "SGAI",
                                       "ACCR",
                                       "LEVI",
                                       "Manipulater"
)])

model.train <- as.data.frame(data.train[,c(#"DSRI",
                                          #"GMI",
                                          "AQI",
                                          #"SGI",
                                          "DEPI",
                                          "SGAI",
                                          "ACCR",
                                          "LEVI",
                                          "Manipulater"
)])

model.test <- as.data.frame(data.test[,c(#"DSRI",
                                        #"GMI",
                                        "AQI",
                                        #"SGI",
                                        "DEPI",
                                        "SGAI",
                                        "ACCR",
                                        "LEVI",
                                        "Manipulater"
)])
```

Corelation amongst variable

The below chunk of code will show the co-relation if any between the numerical variables. The function **highlyCorelated()** shows the variables which are corelated with an absolute corelation of more than 0.6. In this case there are no variables which are highly corelated.

```
correlationMatrix <- cor(model.data[, c(1:5)])
print(correlationMatrix)
```

```
##               AQI        DEPI         SGAI        ACCR        LEVI
## AQI   1.000000000 -0.02124161  0.003712316 -0.04542383  0.07027302
## DEPI -0.021241615  1.00000000 -0.067247329 -0.01661336 -0.01271157
## SGAI  0.003712316 -0.06724733  1.000000000 -0.09066795  0.02174950
## ACCR -0.045423827 -0.01661336 -0.090667950  1.00000000 -0.01163113
## LEVI  0.070273016 -0.01271157  0.021749500 -0.01163113  1.00000000
```

```
# find attributes that are highly corrected (ideally >0.7)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff = 0.6, names =
TRUE)
print(highlyCorrelated)
```

```
## character(0)
```

Caret Package

caret is a useful and a robust package which helps to set a generic framework to implement any kind of model in R. Some of the algorithm's which can be implemented using caret package are:

```
names(getModelInfo())
```

```
##   [1] "ada"                 "AdaBag"              "AdaBoost.M1"
##   [4] "adaboost"            "amdai"               "ANFIS"
##   [7] "avNNet"              "awnb"                "awtan"
##  [10] "bag"                 "bagEarth"            "bagEarthGCV"
##  [13] "bagFDA"              "bagFDAGCV"           "bartMachine"
##  [16] "bayesglm"            "bdk"                 "binda"
##  [19] "blackboost"          "blasso"              "blassoAveraged"
##  [22] "Boruta"              "bridge"              "brnn"
##  [25] "BstLm"               "bstSm"               "bstTree"
##  [28] "C5.0"                "C5.0Cost"            "C5.0Rules"
##  [31] "C5.0Tree"            "cforest"             "chaid"
##  [34] "CSimca"              "ctree"               "ctree2"
##  [37] "cubist"              "dda"                 "deepboost"
##  [40] "DENFIS"              "dnn"                 "dwdLinear"
##  [43] "dwdPoly"             "dwdRadial"           "earth"
##  [46] "elm"                 "enet"                "enpls.fs"
##  [49] "enpls"               "evtree"              "extraTrees"
##  [52] "fda"                 "FH.GBML"             "FIR.DM"
##  [55] "foba"                "FRBCS.CHI"           "FRBCS.W"
##  [58] "FS.HGD"              "gam"                 "gamboost"
```

```
##  [61] "gamLoess"             "gamSpline"            "gaussprLinear"
##  [64] "gaussprPoly"          "gaussprRadial"        "gbm"
##  [67] "gcvEarth"             "GFS.FR.MOGUL"         "GFS.GCCL"
##  [70] "GFS.LT.RS"            "GFS.THRIFT"           "glm"
##  [73] "glmboost"             "glmnet"               "glmStepAIC"
##  [76] "gpls"                 "hda"                  "hdda"
##  [79] "hdrda"                "HYFIS"                "icr"
##  [82] "J48"                  "JRip"                 "kernelpls"
##  [85] "kknn"                 "knn"                  "krlsPoly"
##  [88] "krlsRadial"           "lars"                 "lars2"
##  [91] "lasso"                "lda"                  "lda2"
##  [94] "leapBackward"         "leapForward"          "leapSeq"
##  [97] "Linda"                "lm"                   "lmStepAIC"
## [100] "LMT"                  "loclda"               "logicBag"
## [103] "LogitBoost"           "logreg"               "lssvmLinear"
## [106] "lssvmPoly"            "lssvmRadial"          "lvq"
## [109] "M5"                   "M5Rules"              "manb"
## [112] "mda"                  "Mlda"                 "mlp"
## [115] "mlpML"                "mlpSGD"               "mlpWeightDecay"
## [118] "mlpWeightDecayML"     "multinom"             "nb"
## [121] "nbDiscrete"           "nbSearch"             "neuralnet"
## [124] "nnet"                 "nnls"                 "nodeHarvest"
## [127] "oblique.tree"         "OneR"                 "ordinalNet"
## [130] "ORFlog"               "ORFpls"               "ORFridge"
## [133] "ORFsvm"               "ownn"                 "pam"
## [136] "parRF"                "PART"                 "partDSA"
## [139] "pcaNNet"              "pcr"                  "pda"
## [142] "pda2"                 "penalized"            "PenalizedLDA"
## [145] "plr"                  "pls"                  "plsRglm"
## [148] "polr"                 "ppr"                  "protoclass"
## [151] "pythonKnnReg"         "qda"                  "QdaCov"
## [154] "qrf"                  "qrnn"                 "randomGLM"
## [157] "ranger"               "rbf"                  "rbfDDA"
## [160] "Rborist"              "rda"                  "relaxo"
## [163] "rf"                   "rFerns"               "RFlda"
## [166] "rfRules"              "ridge"                "rlda"
## [169] "rlm"                  "rmda"                 "rocc"
## [172] "rotationForest"       "rotationForestCp"     "rpart"
## [175] "rpart1SE"             "rpart2"               "rpartCost"
## [178] "rpartScore"           "rqlasso"              "rqnc"
## [181] "RRF"                  "RRFglobal"            "rrlda"
## [184] "RSimca"               "rvmLinear"            "rvmPoly"
## [187] "rvmRadial"            "SBC"                  "sda"
## [190] "sddaLDA"              "sddaQDA"              "sdwd"
## [193] "simpls"               "SLAVE"                "slda"
## [196] "smda"                 "snn"                  "sparseLDA"
## [199] "spikeslab"            "spls"                 "stepLDA"
## [202] "stepQDA"              "superpc"              "svmBoundrangeString"
## [205] "svmExpoString"        "svmLinear"            "svmLinear2"
## [208] "svmLinearWeights"     "svmPoly"              "svmRadial"
```

```
## [211] "svmRadialCost"       "svmRadialSigma"      "svmRadialWeights"
## [214] "svmSpectrumString"   "tan"                 "tanSearch"
## [217] "treebag"             "vbmpRadial"          "vglmAdjCat"
## [220] "vglmContRatio"       "vglmCumulative"      "widekernelpls"
## [223] "WM"                  "wsrf"                "xgbLinear"
## [226] "xgbTree"             "xyf"

#getModelInfo()$glm
```

Bagging Model

Bagging is the process of taking bootstrap sample and then aggreagting the model learned on each sample. Each of the models are trained independently on the N observations picked randomly from N observations in the original dataset (with replacement). The models can be trained parallely as the training is based on independent samples. Since models are trained on different but overlapping samples of the original data, the predictions from different models will be different.

Bagging models in R

The algorithms in bagging are:

1. Bagged Adaboost: ***adabag()*** Required Package is **adabag, plyr**
2. Bagged CART: ***treebag()*** Required Package is **ipred, e1071, plyr**
3. Bagged Flexible Discriminant Analysis: ***bagFDA()*** Required Package is **earth, mda**
4. Bagged Logic Regression: ***logicBag()*** Required Package is **logicFS**
5. Bagged MARS: ***bagEarth()*** Required Package is **earth**
6. Bagged Model: ***bag()*** Required Package is **caret**
7. Ensemble of Generalized Linear Models: ***randomGLM()*** Required Package is **randomGLM**
8. Model Averaged Neural Network: ***avNNET()*** Required Package is **nnet**
9. Quantile Regression Neural Network: ***qrnn()*** Required Package is **qrnn**
10. Random Ferns: ***rFerns()*** Required Package is **rFerns**

The below methods are all applicable to implement random forest as a bagging algorithm:

11. Parallel Random Forest: ***parRF()*** Required Package is **e1071, randomForest, foreach**
12. Quantile Random Forest: ***qrf()*** Required Package is **quantregForest**
13. Conditional Inference Random Forest: ***cforest()*** Required Package is **party**
14. Random Forest: ***ranger()*** Required Package is **e1071, ranger**
15. Random Forest: ***Rborist()*** Required Package is **Rborist**
16. Random Forest: ***rf()*** Required Package is **randomForest**
17. Random Forest by Randomization: ***extraTrees()*** Required Package is **extraTrees**
18. Random Forest rule based Model: ***rfRules()*** Required Package is **randomForest, inTrees, plyr**
19. Regularized Random Forest: ***RRF()*** Required Package is **randomForest, RRF**

20. Regularized Random Forest: ***RRFglobal()*** Required Package is **RRF**

21. Weighted Subspace Random Forest: ***wsrf()*** Required Package is **wsrf**

Random Forest with bootstrap sampling

Random forests is one of the algorithm which uses bagging as a technique. In the below code chunk we will use bootstrap sampling to implement bagging using rf method. This means that if there are 100 observations in a training dataset the resulting sample will select 100 samples with replacement.

The below code chunk sets some of the control parameters

```
objControl <- trainControl(method='boot', number = 1,
                           returnResamp='none',
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           classProbs = TRUE)
```

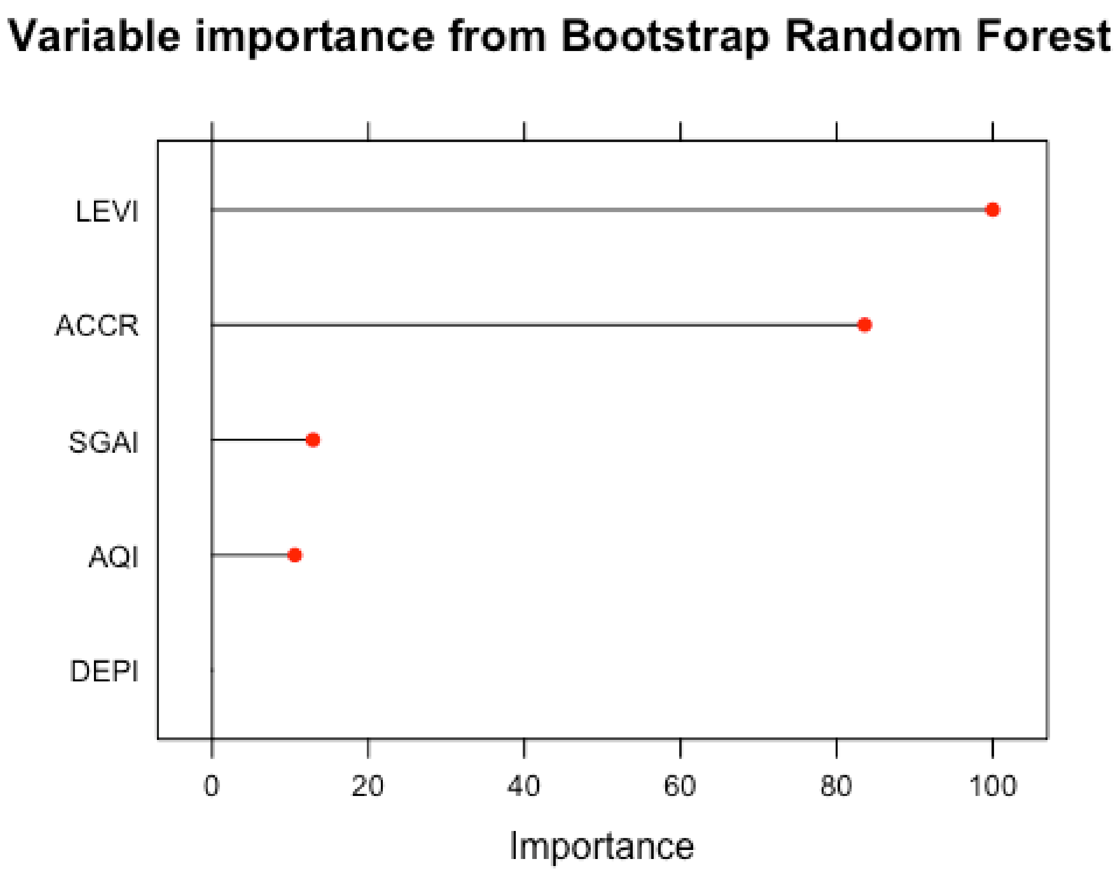
After setting the control paramters, the model is run

```
set.seed(4121)

rf.bootstrap.model <- train(model.train[,1:5], model.train[,6],
                  method='rf',
                  trControl=objControl,
                  metric = "ROC",
                  prox=TRUE,allowParallel=TRUE)
```

Confusion Matrix for bootstrap sampling on train set

```
#rf.bootstrap.model$finalModel #rf.bootstrap.model$results
print(rf.bootstrap.model)

## Random Forest
##
## 868 samples
##   5 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Bootstrapped (1 reps)
## Summary of sample sizes: 868
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens      Spec
##   2     0.9356250  0.996875  0.1
##   3     0.9296875  0.996875  0.1
##   5     0.9207812  0.996875  0.2
##
## ROC was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```r
confusionMatrix.train(rf.bootstrap.model)

## Bootstrapped (1 reps) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##             Reference
## Prediction    No  Yes
##        No   96.7  2.7
##        Yes   0.3  0.3
##
##  Accuracy (average) : 0.9697

plot(varImp(rf.bootstrap.model), main = "Variable importance from Bootstrap
Random Forest", col = 2, lwd = 2)
```



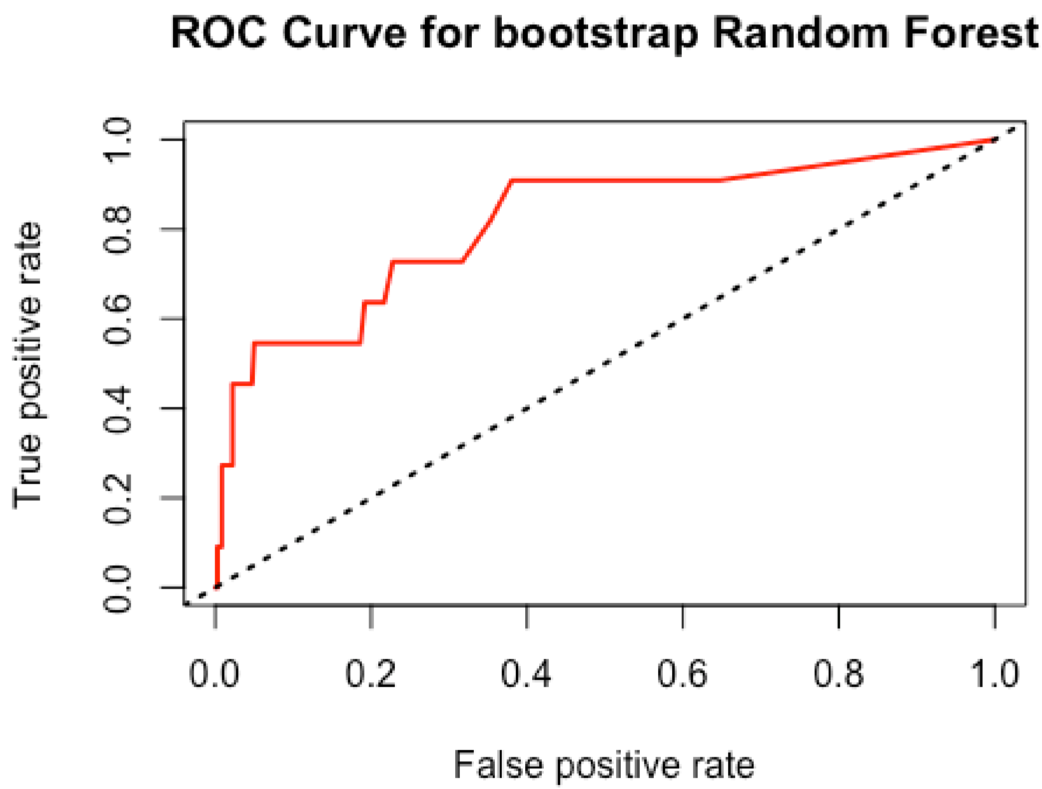
Confusion Matrix for bootstrap sampling on test set

```r
caretPredictedClass <- predict(rf.bootstrap.model, model.test, type = "raw")
confusionMatrix(caretPredictedClass,model.test$Manipulater)

## Confusion Matrix and Statistics
##
##             Reference
```

```
## Prediction  No Yes
##        No  359  10
##        Yes   1   1
##
##                Accuracy : 0.9704
##                  95% CI : (0.9476, 0.9851)
##     No Information Rate : 0.9704
##     P-Value [Acc > NIR] : 0.57928
##
##                   Kappa : 0.1461
##  Mcnemar's Test P-Value : 0.01586
##
##             Sensitivity : 0.99722
##             Specificity : 0.09091
##          Pos Pred Value : 0.97290
##          Neg Pred Value : 0.50000
##              Prevalence : 0.97035
##          Detection Rate : 0.96765
##    Detection Prevalence : 0.99461
##       Balanced Accuracy : 0.54407
##
##        'Positive' Class : No
##
```

ROC plot for bootstrap random forest on test set

```
rf.bootstrap.pred <- predict(rf.bootstrap.model, model.test, type =
"prob")[,2]
rf.bootstrap.prediction <-
prediction(rf.bootstrap.pred,model.test$Manipulater)
rf.bootstrap.perf <- performance(rf.bootstrap.prediction, "tpr","fpr")

plot(rf.bootstrap.perf,main="ROC Curve for bootstrap Random
Forest",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```




```r
#AUC for the ROC plot
performance(rf.bootstrap.prediction, "auc")
```

```
## An object of class "performance"
## Slot "x.name":
## [1] "None"
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.8137626
##
##
## Slot "alpha.values":
## list()
```

The best model was

```
rf.bootstrap.model$bestTune
```

```
##   mtry
## 1    2
```

Visulaizing the rules coming out of random forest. We can loop and print all the trees built using up sampling. For simplicity, printing just one of the trees

Random Forest with up sampling

To incorporate up-sampling (sample the minority class to make their frequencies closer to the majority class.), random forest can use an upsampling strategy

The below code chunk sets some of the control parameters

```
objControl <- trainControl(method='boot', number = 1,
                           returnResamp='final',
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           classProbs = TRUE,
                           sampling="up")
```

After setting the control paramters, the model is run

```
set.seed(4121)

rf.up.model <- train(model.train[,1:5], model.train[,6],
                  method='rf',
                  trControl=objControl,
                  metric = "ROC",
                  prox=TRUE,allowParallel=TRUE)
```

Confusion Matrix for upsampling on train set

```
#rf.up.model$finalModel #rf.up.model$results

print(rf.up.model)
```

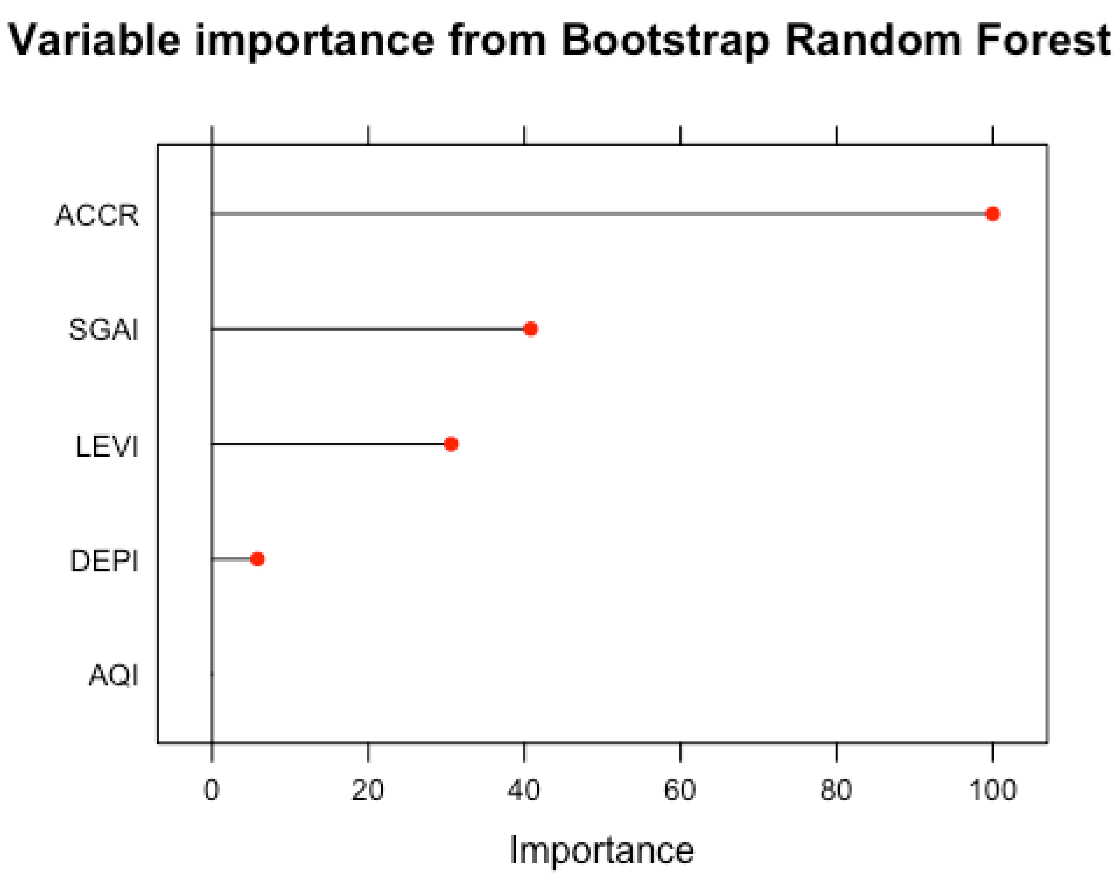
```
## Random Forest
##
## 868 samples
##   5 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Bootstrapped (1 reps)
## Summary of sample sizes: 868
## Addtional sampling using up-sampling
##
## Resampling results across tuning parameters:
```

```
##
##   mtry  ROC        Sens     Spec
##   2     0.8962500  0.99375  0.1
##   3     0.8714063  0.98750  0.2
##   5     0.8223437  0.98750  0.1
##
## ROC was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
confusionMatrix.train(rf.up.model)
```

```
## Bootstrapped (1 reps) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##             Reference
## Prediction    No  Yes
##        No  96.4  2.7
##        Yes  0.6  0.3
##
##  Accuracy (average) : 0.9667
```

```
plot(varImp(rf.up.model), main = "Variable importance from Bootstrap Random
Forest", col = 2, lwd = 2)
```



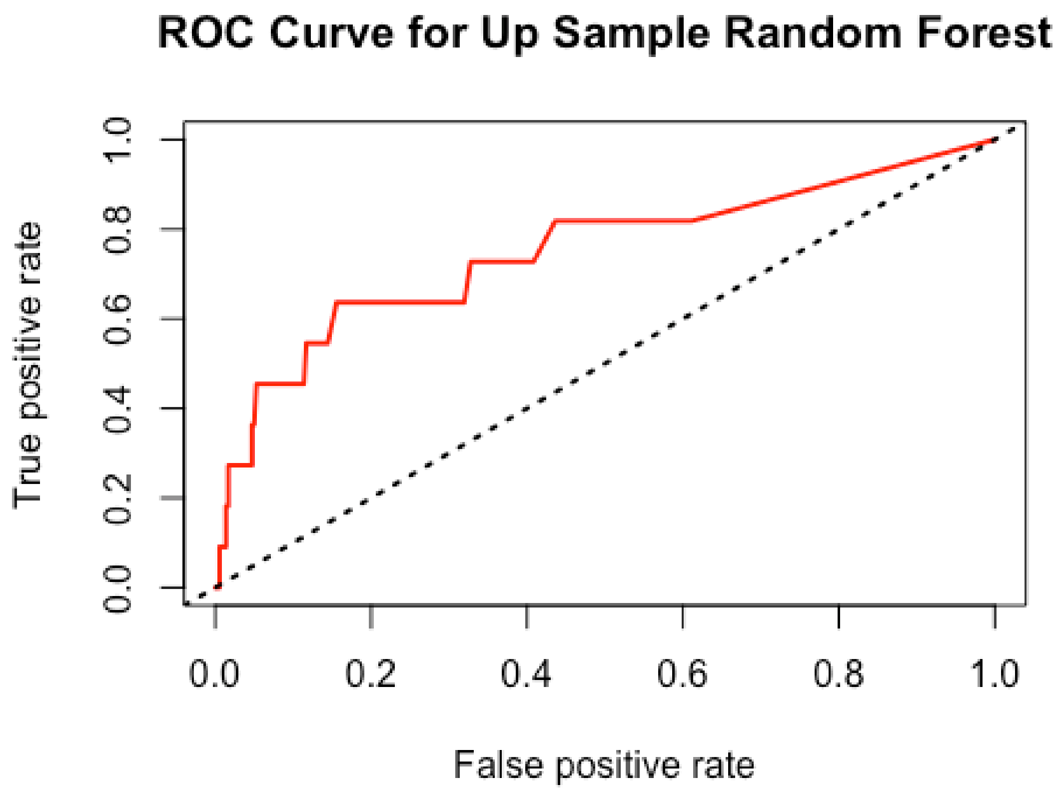
Confusion Matrix for upsampling on test set

```
caretPredictedClass <- predict(rf.up.model, model.test, type = "raw")
confusionMatrix(caretPredictedClass,model.test$Manipulater)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  358  11
##        Yes   2   0
##
##                Accuracy : 0.965
##                  95% CI : (0.9408, 0.9812)
##     No Information Rate : 0.9704
##     P-Value [Acc > NIR] : 0.7841
##
##                   Kappa : -0.0092
##  Mcnemar's Test P-Value : 0.0265
##
##             Sensitivity : 0.9944
##             Specificity : 0.0000
##          Pos Pred Value : 0.9702
##          Neg Pred Value : 0.0000
##              Prevalence : 0.9704
##          Detection Rate : 0.9650
##    Detection Prevalence : 0.9946
##       Balanced Accuracy : 0.4972
##
##        'Positive' Class : No
##
```

ROC plot for upsample random forest on test set

```
rf.up.pred <- predict(rf.up.model, model.test, type = "prob")[,2]
rf.up.prediction <- prediction(rf.up.pred,model.test$Manipulater)
rf.up.perf <- performance(rf.up.prediction, "tpr","fpr")

plot(rf.up.perf,main="ROC Curve for Up Sample Random Forest",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```




```
#AUC for the ROC plot
performance(rf.up.prediction, "auc")

## An object of class "performance"
## Slot "x.name":
## [1] "None"
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.7493687
##
##
## Slot "alpha.values":
## list()
```

Extracting all the rules from the trees built using random forest

Random Forest with down sampling - First Approach

To incorporate down-sampling (sample the majority class to make their frequencies closer to the minority class.), random forest can use an downsampling strategy

The below code chunk sets some of the control parameters

```
objControl <- trainControl(method='boot', number = 1,
                           returnResamp='final',
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           classProbs = TRUE,
                           sampling="down")
```

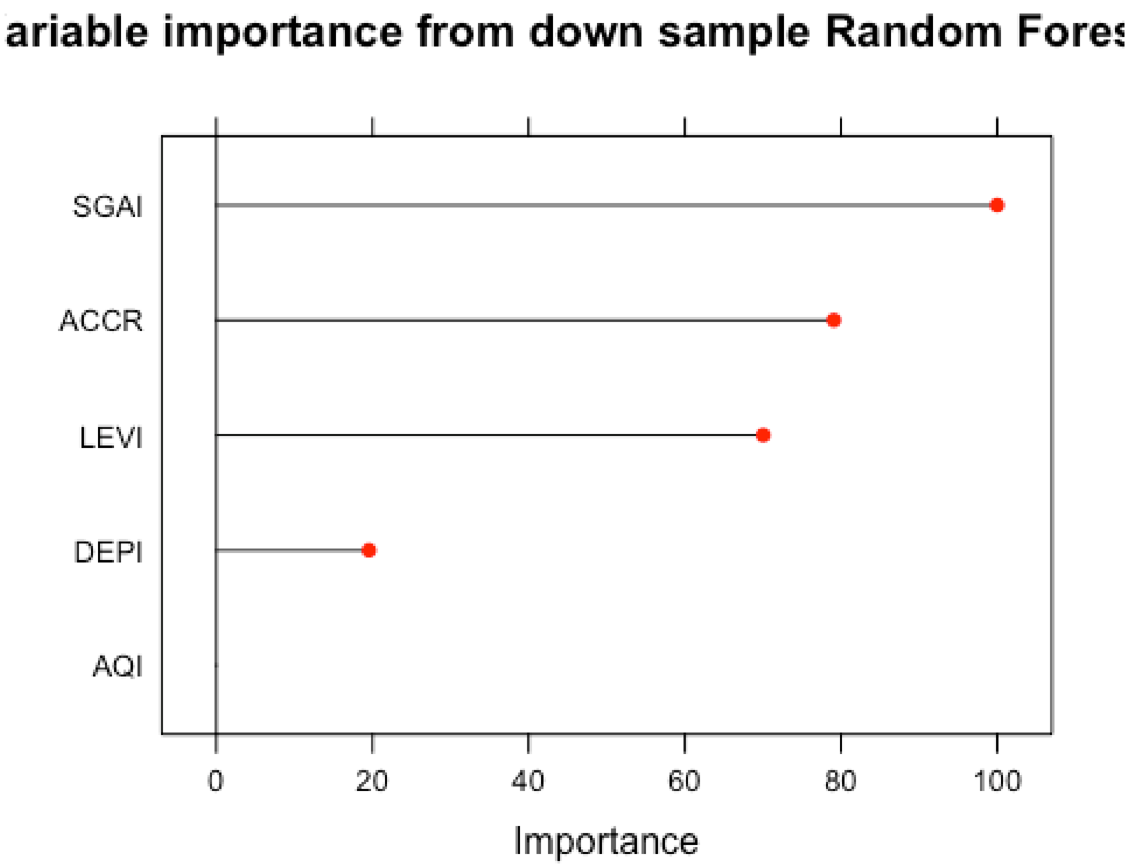
After setting the control parameters, the model is run

```
set.seed(4121)

rf.down1.model <- train(model.train[,1:5], model.train[,6],
                  method='rf',
                  trControl=objControl,
                  metric = "ROC",
                  prox=TRUE,allowParallel=TRUE)
```

Confusion Matrix for down sampling RF on train set

```
#rf.down1.model$finalModel #rf.down1.model$results
print(rf.down1.model)

## Random Forest
##
## 868 samples
##   5 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Bootstrapped (1 reps)
## Summary of sample sizes: 868
## Addtional sampling using down-sampling
##
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens      Spec
##   2     0.8309375  0.790625  0.6
##   3     0.8282812  0.759375  0.7
##   5     0.8275000  0.821875  0.7
##
## ROC was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
confusionMatrix.train(rf.down1.model)

## Bootstrapped (1 reps) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##             Reference
## Prediction    No  Yes
##        No   76.7  1.2
##        Yes  20.3  1.8
##
##  Accuracy (average) : 0.7848

plot(varImp(rf.down1.model), main = "Variable importance from down sample
Random Forest", col = 2, lwd = 2)
```



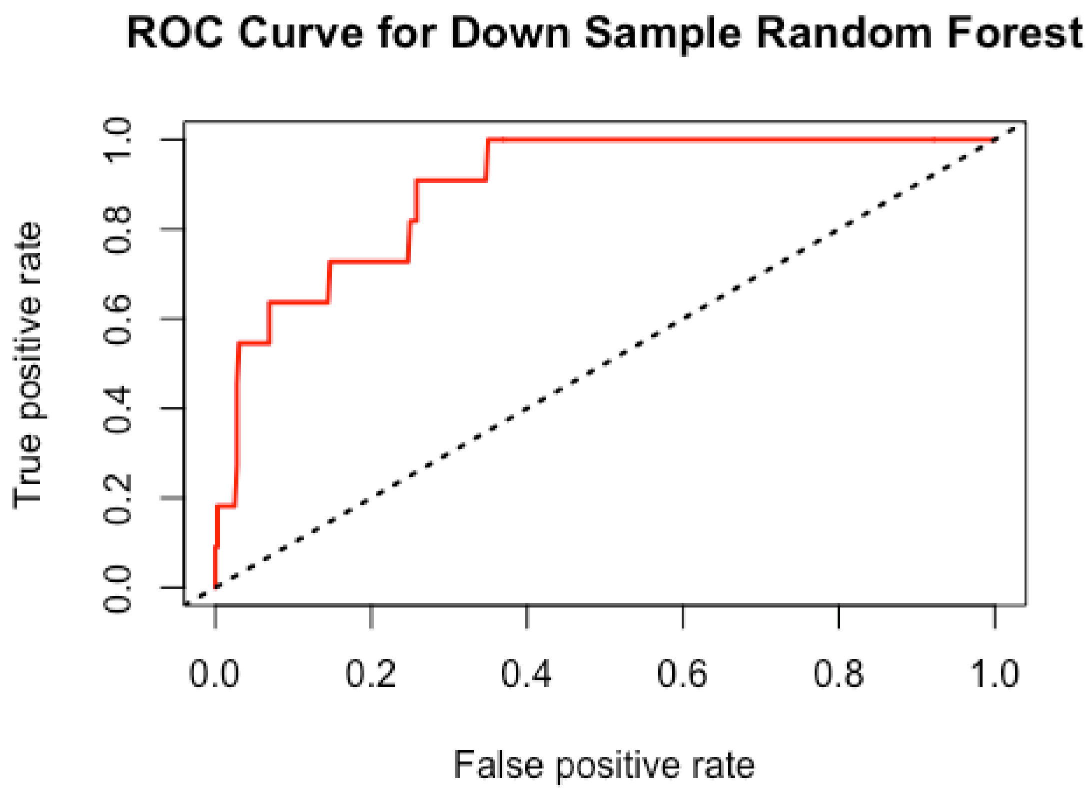
Confusion Matrix for down sampling RF on test set

```
caretPredictedClass <- predict(rf.down1.model, model.test, type = "raw")
confusionMatrix(caretPredictedClass,model.test$Manipulater)

## Confusion Matrix and Statistics
##
##             Reference
```

```
## Prediction  No Yes
##        No  282   3
##        Yes  78   8
##
##                Accuracy : 0.7817
##                  95% CI : (0.7361, 0.8227)
##     No Information Rate : 0.9704
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1186
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.78333
##             Specificity : 0.72727
##          Pos Pred Value : 0.98947
##          Neg Pred Value : 0.09302
##              Prevalence : 0.97035
##          Detection Rate : 0.76011
##    Detection Prevalence : 0.76819
##       Balanced Accuracy : 0.75530
##
##        'Positive' Class : No
##
```

ROC plot for down sample random forest on test set

```
rf.down1.pred <- predict(rf.down1.model, model.test, type = "prob")[,2]
rf.down1.prediction <- prediction(rf.down1.pred,model.test$Manipulater)
rf.down1.perf <- performance(rf.down1.prediction, "tpr","fpr")

plot(rf.down1.perf,main="ROC Curve for Down Sample Random
Forest",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```




```
#AUC for the ROC plot
performance(rf.down1.prediction, "auc")
```

```
## An object of class "performance"
## Slot "x.name":
## [1] "None"
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.892298
##
##
## Slot "alpha.values":
## list()
```

Random Forest with down sampling - Second Approach

To incorporate down-sampling (sample the majority class to make their frequencies closer to the rarest class.), random forest can take a random sample of size c*nmin, where c is the number of classes and nmin is the number of samples in the minority class.

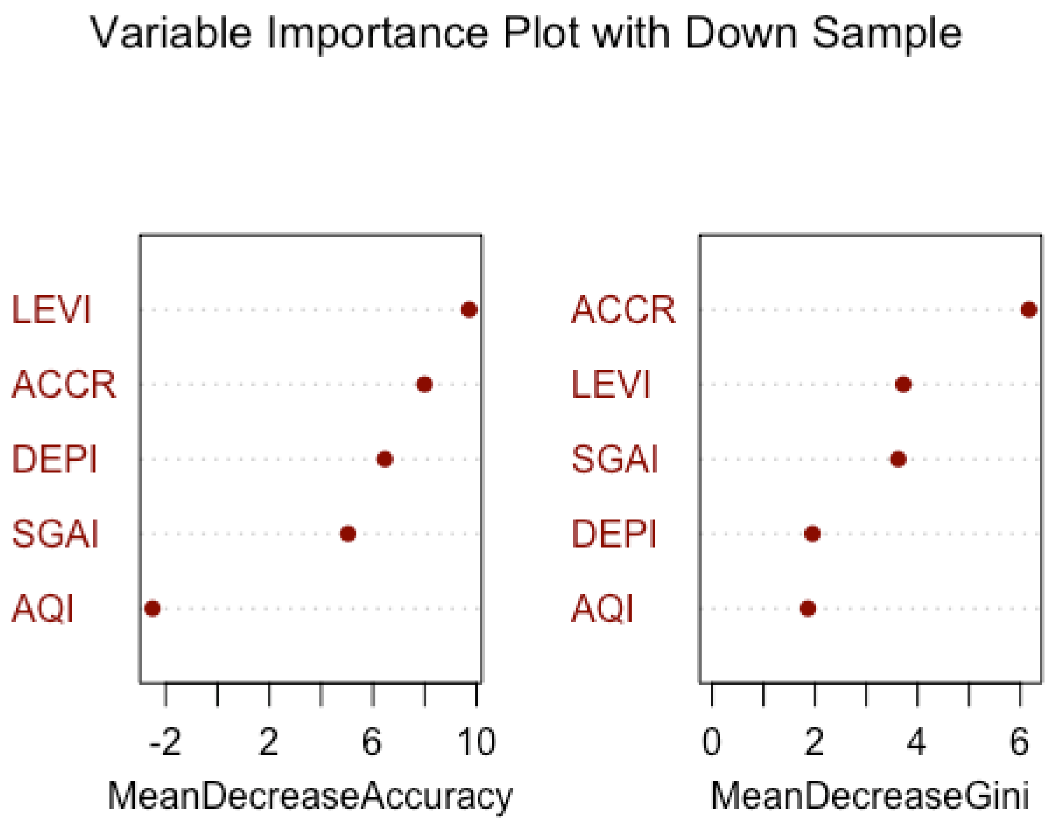
THIS IMPLEMENTATION IS WITHOUT CARET PACKAGE

```
nmin <- sum(model.train$Manipulater == "Yes") #total minority cases
set.seed(4121)
rf.down2.model <- randomForest(Manipulater ~ .,
                         data=model.train, importance=TRUE, mtry = 2,
                         #if strata is not defined RF does bootstrap sample
                         strata = model.train$Manipulater,
                         #selecting nmin cases from positive and negative
class
                         sampsize = rep(nmin,2),
                         #cutoff: 'winning' class for an observation is the
one
                         #with the maximum ratio of proportion of votes to
cutoff.
                         cutoff = c(1/2, 1/2),ntree=1024,  nodesize = 10,
                         keep.forest = TRUE)#, xtest = model.test[,-12])
```

Variable importance and Confusion matrix on downsample random forest on train set

```
#To plot the error rate.
#plot(rf.down2.model, main = "Error rate vs. number of trees (RF with
downsample", type = "l", lwd = 3)

#To know the legends, type rf.down2.model to get the confusion matrix and
#see the error

print(rf.down2.model)

##
## Call:
##  randomForest(formula = Manipulater ~ ., data = model.train, importance =
TRUE,      mtry = 2, strata = model.train$Manipulater, sampsize = rep(nmin,
2), cutoff = c(1/2, 1/2), ntree = 1024, nodesize = 10,      keep.forest =
TRUE)
##                Type of random forest: classification
##                      Number of trees: 1024
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 21.54%
## Confusion matrix:
##      No Yes class.error
## No  663 177   0.2107143
## Yes  10  18   0.3571429
```

```r
varImpPlot(rf.down2.model, main = "Variable Importance Plot with Down
Sample", pch = 16, col = 'darkred')
```



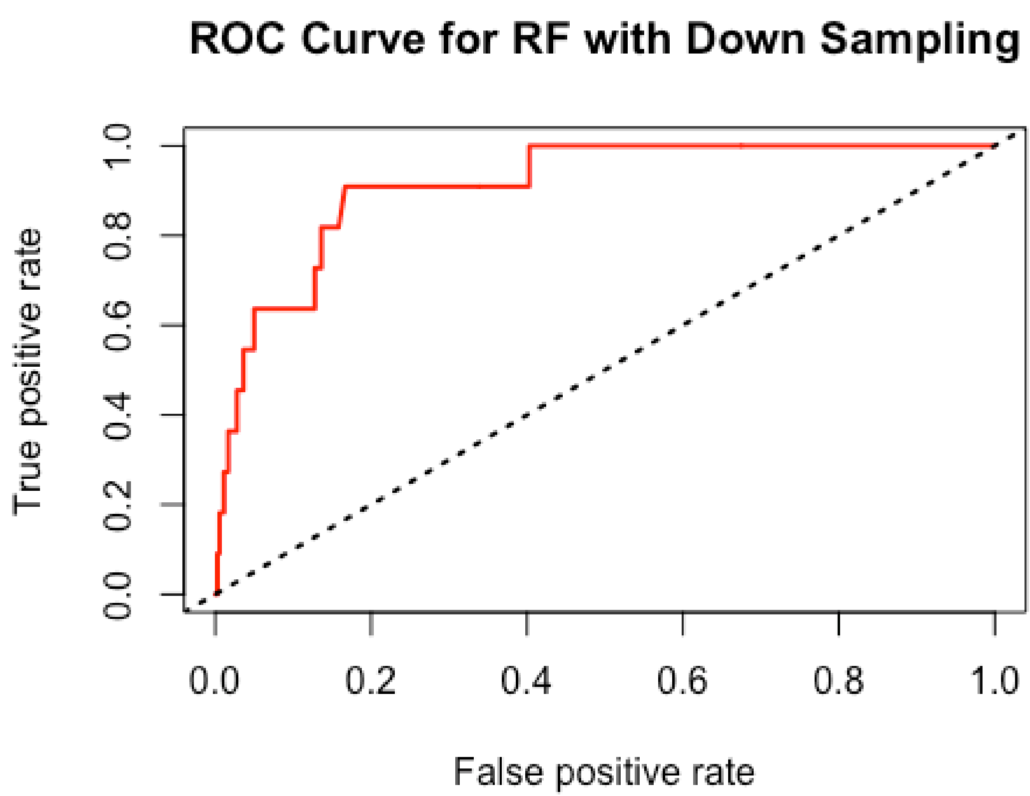
Variable importance and Confusion matrix on downsample random forest on test set

```r
testPredictedClass <- predict(rf.down2.model, model.test, type = "response")
confusionMatrix(testPredictedClass,model.test$Manipulater)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  275   1
##        Yes  85  10
##
##                Accuracy : 0.7682
##                  95% CI : (0.7219, 0.8102)
##     No Information Rate : 0.9704
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1431
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7639
```

```
##               Specificity : 0.9091
##            Pos Pred Value : 0.9964
##            Neg Pred Value : 0.1053
##                Prevalence : 0.9704
##            Detection Rate : 0.7412
##      Detection Prevalence : 0.7439
##         Balanced Accuracy : 0.8365
##
##          'Positive' Class : No
##
```

ROC plot for Random Forest with downsampling on test set

```
rf.down2.pred <- predict(rf.down2.model, model.test, type = "prob")[,2]
rf.down2.prediction <- prediction(rf.down2.pred,model.test$Manipulater)
rf.down2.perf <- performance(rf.down2.prediction, "tpr","fpr")

plot(rf.down2.perf,main="ROC Curve for RF with Down Sampling",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```




```
#AUC for the ROC plot
performance(rf.down2.prediction, "auc")
```

```
## An object of class "performance"
## Slot "x.name":
## [1] "None"
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.9109848
##
##
## Slot "alpha.values":
## list()
```

Random Forest with SMOTE

Synthetic minority oversampling technique (SMOTE) blends under-sampling of the majority class with a special form of over-sampling the minority class. SMOTE oversamples the rare event by using bootstrapping and k-nearest neighbor to synthetically create additional observations of that event.

The below code chunk sets some of the control parameters

```
objControl <- trainControl(method='cv', number = 5,
                           returnResamp='final',
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           classProbs = TRUE,
                           sampling="smote")
```

After setting the control parameters, the model is run

```
set.seed(4121)

rf.smote.model <- train(model.train[,1:5], model.train[,6],
                  method='rf',
                  trControl=objControl,
                  metric = "ROC",
                  prox=TRUE,allowParallel=TRUE)
```

Confusion Matrix for RF on train set

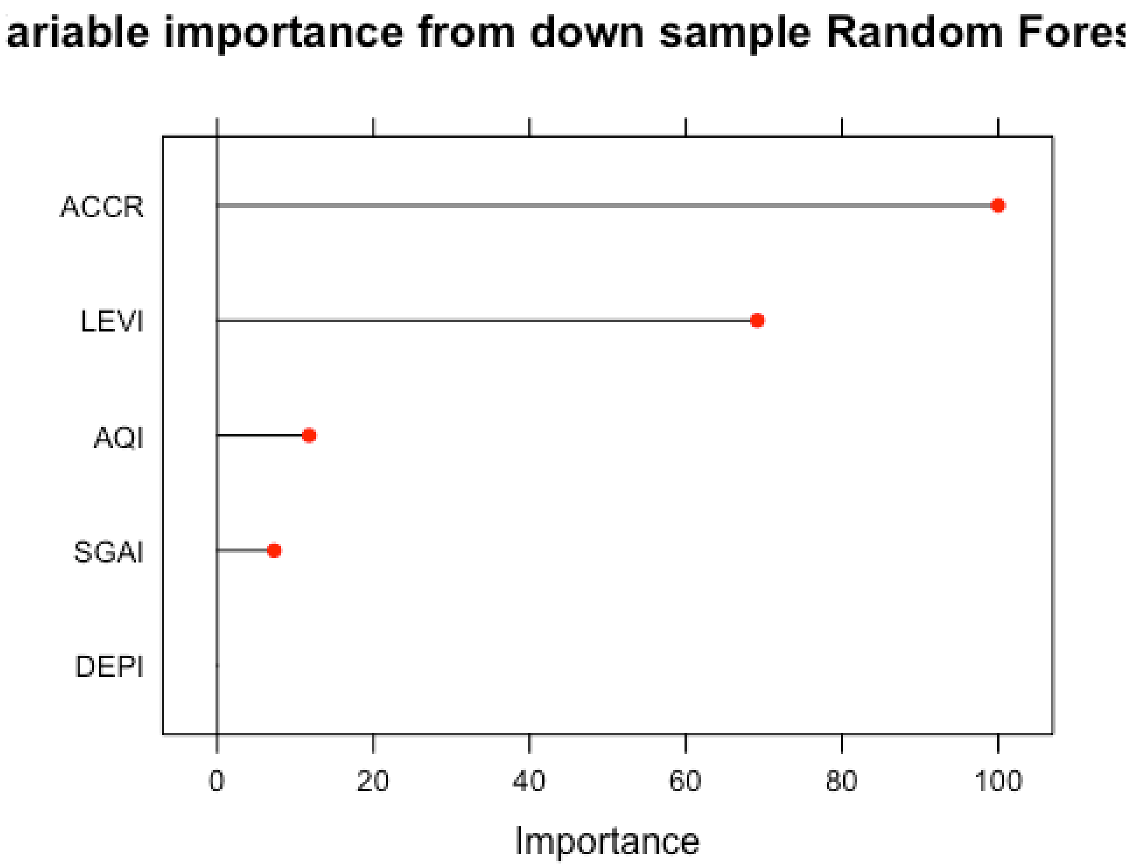
```
#rf.smote.model$finalModel #rf.smote.model$results
print(rf.smote.model)
```

```
## Random Forest
##
## 868 samples
##   5 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 694, 694, 695, 695, 694
## Addtional sampling using SMOTE
##
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##   2     0.7754762  0.8440476  0.3733333
##   3     0.7565476  0.8345238  0.4000000
##   5     0.7198413  0.8285714  0.3266667
##
## ROC was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
confusionMatrix.train(rf.smote.model)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction   No  Yes
##        No  81.7  2.1
##        Yes 15.1  1.2
##
##  Accuracy (average) : 0.8283
```

```
plot(varImp(rf.smote.model), main = "Variable importance from down sample
Random Forest", col = 2, lwd = 2)
```



Confusion Matrix for RF on test set

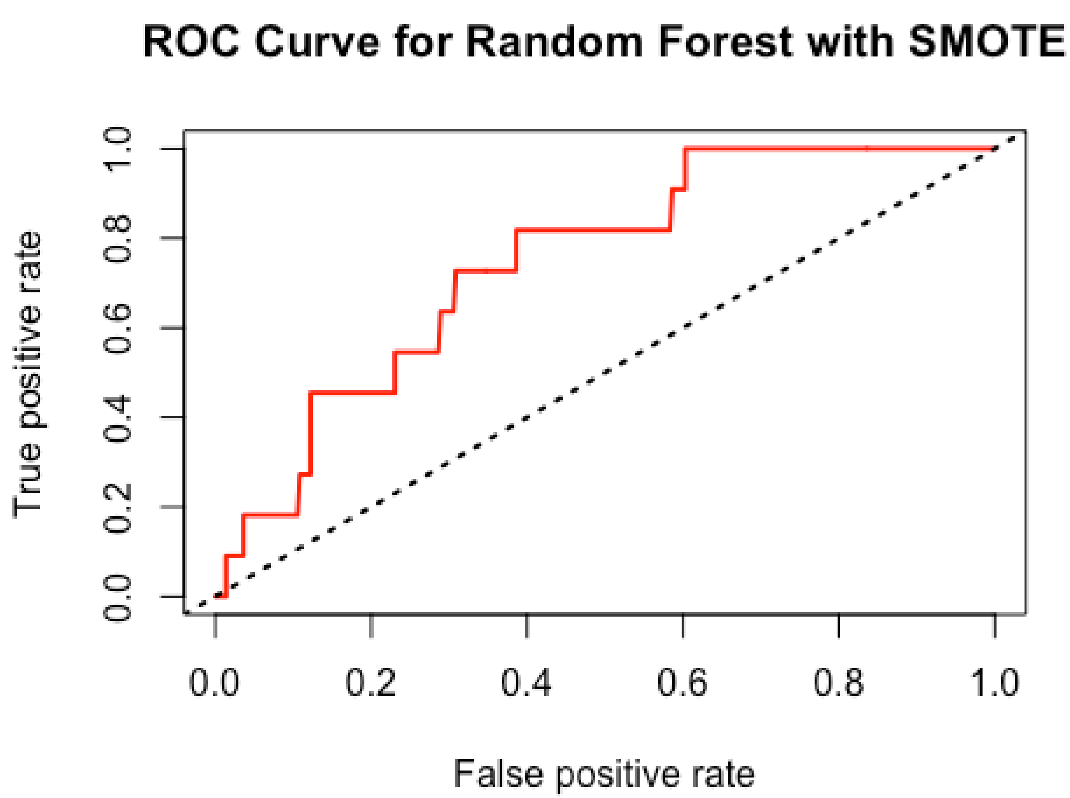
```
caretPredictedClass <- predict(rf.smote.model, model.test, type = "raw")
confusionMatrix(caretPredictedClass,model.test$Manipulater)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  316   7
##        Yes  44   4
##
##                Accuracy : 0.8625
##                  95% CI : (0.8232, 0.8959)
##     No Information Rate : 0.9704
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0918
##  Mcnemar's Test P-Value : 4.631e-07
##
##             Sensitivity : 0.87778
##             Specificity : 0.36364
##          Pos Pred Value : 0.97833
```

```
##          Neg Pred Value : 0.08333
##              Prevalence : 0.97035
##          Detection Rate : 0.85175
##    Detection Prevalence : 0.87062
##       Balanced Accuracy : 0.62071
##
##        'Positive' Class : No
##
```

ROC plot for random forest on test set

```
rf.smote.pred <- predict(rf.smote.model, model.test, type = "prob")[,2]
rf.smote.prediction <- prediction(rf.smote.pred,model.test$Manipulater)
rf.smote.perf <- performance(rf.smote.prediction, "tpr","fpr")

plot(rf.smote.perf,main="ROC Curve for Random Forest with SMOTE",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```




```
#AUC for the ROC plot
performance(rf.smote.prediction, "auc")

## An object of class "performance"
## Slot "x.name":
## [1] "None"
```

```
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.7454545
##
##
## Slot "alpha.values":
## list()
```

Random Forest with Edited Neareast Neighbour

Edited Nearest Neighbor for multiclass imbalanced problems. It removes examples whose class label differs from the class of at least half of its k nearest neighbors. All the existing classes can be under-sampled with this technique. Alternatively a subset of classes to under-sample can be provided by the user.

The below code chunk sets some of the control parameters

After setting the control parameters, the model is run

Confusion Matrix for ENN RF on train set

Confusion Matrix for ENN RF on test set

ROC plot for down sample random forest on test set

Boosting

Boosting is an ensemble technique which tries to create a strong classifier from several weak classifier. The model buidling through boosting is sequential. 1. The first model is build based on the random sample on N observations picked from original dataset (with replacement). Equal weight is assigned to each observation. These weights decide the probability of observations which will be picked up in the training set. 2. In the second step, all the original dataset is passed through the model. For regressor model, the observations whose predicted value differs the most from the actual value is defined to be most in error. 3. The sampling probabilities of the observations which are most in error, is adjusted such that their chance of getting picked up for the second model is higher. 4. As the model buidling progresses, in each of the sequence of models, the pattern which are more difficult are picked up. Different models are better in different part of the observation space. 5.

Rgeressors are combined using weighted median. Models which are more confident about their predictions are weighted more heavily.

Boosting algorithms in R

Adaboost is one of the ways to boost the performance of decision trees on binary classification problems. The decision trees with just one level will mostly be a weak learner. These weak learners will achieve an accuracy just above random chance on a classification problem.

Adaboost is also referred to as discrete AdaBoost as it is used for classification rather than regression. The algorithms in boosting are:

1. Adaboost classification trees: ***adaboost()*** Required Package is **fastAdaboost**
2. Adaboost.M1: ***AdaBoost.M1()*** Required Package is **adabag, plyr**
3. Boosted Classification Trees: ***ada()*** Required Package is **adabag, plyr**
4. Boosted Generalized Additive Model: ***gamBoost()*** Required Package is **mboost, plyr**
5. Boosted Generalized Linear Model: ***glmboost()*** Required Package is **mboost, plyr**
6. Boosted Linear Model: ***Bstlm()*** Required Package is **bst, plyr**
7. Boosted Logistic Regression: ***LogitBoost()*** Required Package is **caTools**
8. Boosted Smoothing Spline: ***bstSm()*** Required Package is **bst, plyr**
9. Boosted Tree: ***blackboost()*** Required Package is **party, mboost, plyr**
10. Boosted Tree: ***bstTree()*** Required Package is **bst, plyr**
11. C5.0: ***C5.0()*** Required Package is **C50, plyr**
12. Cost Sensitive C5.0: ***C5.0Cost()*** Required Package is **C50, plyr**
13. Cubist: ***glmboost()*** Required Package is **cubist**
14. DeepBoost: ***deepboost()*** Required Package is **deepboost**
15. eXtreme Gradient Boosting: ***xgbLinear()*** Required Package is **xgboost**
16. eXtreme Gradient Boosting: ***xgbTree()*** Required Package is **xgboost, plyr**
17. Stochastic Gradient Boosting: ***gbm()*** Required Package is **gbm, plyr**

Boosting with adaboost (normal CV)

The below code chunk sets some of the control parameters for adaboost

```
objControl <- trainControl(method='cv', number = 5,
                           returnResamp='all',
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           classProbs = TRUE)#, p = 0.70) #in case method =
#"LGOCV"

search.grid <- expand.grid(mfinal = (1:10)*10, maxdepth = c(1:4),
                    coeflearn = c("Breiman", "Freund", "Zhu"))
```

After setting the control paramters, the model is run

```
set.seed(4121)

ada.model <- train(model.train[,1:5], model.train[,6],
                   method='AdaBoost.M1',
                   trControl=objControl,
                   tuneGrid = search.grid,
                   metric = "ROC",
                   prox=TRUE,allowParallel=TRUE)

## Loading required package: adabag

## Loading required package: rpart

## Loading required package: mlbench

## Loading required package: plyr

##
## Attaching package: 'plyr'

## The following object is masked from 'package:DMwR':
##
##     join
```

Confusion Matrix for adaboost on train set

```
#ada.model$finalModel #ada.model$results
print(ada.model)

## AdaBoost.M1
##
## 868 samples
##   5 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 694, 694, 695, 695, 694
## Resampling results across tuning parameters:
##
##   coeflearn  maxdepth  mfinal  ROC        Sens       Spec
##   Breiman    1          10     0.7398214  1.0000000  0.04000000
##   Breiman    1          20     0.7671230  1.0000000  0.04000000
##   Breiman    1          30     0.7653770  1.0000000  0.04000000
##   Breiman    1          40     0.7841468  0.9988095  0.04000000
##   Breiman    1          50     0.7782937  1.0000000  0.04000000
##   Breiman    1          60     0.7801984  1.0000000  0.04000000
##   Breiman    1          70     0.7914484  1.0000000  0.04000000
##   Breiman    1          80     0.8036310  1.0000000  0.04000000
##   Breiman    1          90     0.8037103  1.0000000  0.04000000
```

```
## Breiman 1 100 0.8041071 1.0000000 0.04000000
## Breiman 2 10 0.7781548 0.9976190 0.04000000
## Breiman 2 20 0.8036111 0.9976190 0.08000000
## Breiman 2 30 0.7946627 0.9976190 0.08000000
## Breiman 2 40 0.7933532 0.9964286 0.08000000
## Breiman 2 50 0.8022817 0.9976190 0.08000000
## Breiman 2 60 0.8048413 0.9976190 0.08000000
## Breiman 2 70 0.7995635 0.9976190 0.08000000
## Breiman 2 80 0.7996032 0.9976190 0.08000000
## Breiman 2 90 0.7852381 0.9976190 0.08000000
## Breiman 2 100 0.7941270 0.9988095 0.08000000
## Breiman 3 10 0.7831548 0.9952381 0.07333333
## Breiman 3 20 0.7701786 0.9952381 0.07333333
## Breiman 3 30 0.7709921 0.9964286 0.07333333
## Breiman 3 40 0.7814286 0.9964286 0.11333333
## Breiman 3 50 0.7594444 0.9964286 0.11333333
## Breiman 3 60 0.7659524 0.9976190 0.11333333
## Breiman 3 70 0.7565873 0.9976190 0.11333333
## Breiman 3 80 0.7742857 0.9964286 0.11333333
## Breiman 3 90 0.7750397 0.9988095 0.11333333
## Breiman 3 100 0.7825397 0.9988095 0.11333333
## Breiman 4 10 0.8188492 0.9928571 0.00000000
## Breiman 4 20 0.7895238 0.9952381 0.04000000
## Breiman 4 30 0.7856944 0.9964286 0.00000000
## Breiman 4 40 0.7739683 0.9976190 0.04000000
## Breiman 4 50 0.7761905 0.9964286 0.04000000
## Breiman 4 60 0.7808333 0.9964286 0.04000000
## Breiman 4 70 0.7751190 0.9964286 0.04000000
## Breiman 4 80 0.7743651 0.9964286 0.04000000
## Breiman 4 90 0.7811111 0.9964286 0.04000000
## Breiman 4 100 0.7925794 0.9964286 0.04000000
## Freund 1 10 0.7560913 0.9988095 0.07333333
## Freund 1 20 0.7905754 0.9988095 0.11333333
## Freund 1 30 0.8171032 0.9976190 0.11333333
## Freund 1 40 0.8153373 0.9964286 0.07333333
## Freund 1 50 0.8115675 0.9976190 0.07333333
## Freund 1 60 0.8153770 0.9952381 0.07333333
## Freund 1 70 0.8110516 0.9976190 0.11333333
## Freund 1 80 0.8129167 0.9988095 0.07333333
## Freund 1 90 0.8060913 1.0000000 0.07333333
## Freund 1 100 0.8115278 0.9976190 0.07333333
## Freund 2 10 0.7292659 0.9928571 0.15333333
## Freund 2 20 0.7131151 0.9964286 0.15333333
## Freund 2 30 0.7343056 0.9964286 0.11333333
## Freund 2 40 0.7211905 0.9976190 0.11333333
## Freund 2 50 0.7443651 0.9976190 0.11333333
## Freund 2 60 0.7403175 0.9976190 0.11333333
## Freund 2 70 0.7240476 0.9964286 0.11333333
## Freund 2 80 0.7342460 0.9952381 0.11333333
## Freund 2 90 0.7537698 0.9988095 0.11333333
```

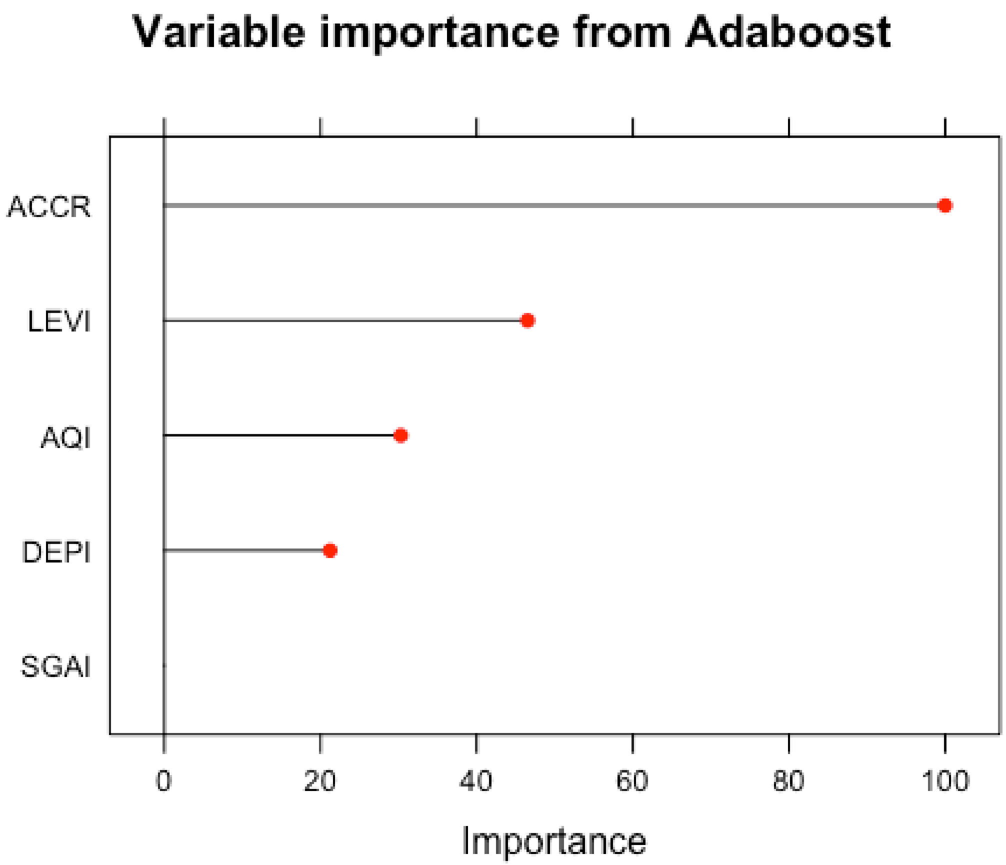
```
## Freund 2 100 0.7512302 0.9976190 0.11333333
## Freund 3 10 0.7551389 0.9904762 0.18666667
## Freund 3 20 0.7517262 0.9940476 0.12000000
## Freund 3 30 0.7400397 0.9952381 0.04000000
## Freund 3 40 0.7479365 0.9976190 0.04000000
## Freund 3 50 0.7488492 0.9976190 0.04000000
## Freund 3 60 0.7349603 0.9964286 0.08000000
## Freund 3 70 0.7552381 0.9952381 0.08000000
## Freund 3 80 0.7554365 0.9964286 0.08000000
## Freund 3 90 0.7689683 0.9964286 0.08000000
## Freund 3 100 0.7649206 0.9964286 0.08000000
## Freund 4 10 0.7097024 0.9928571 0.07333333
## Freund 4 20 0.7595437 0.9940476 0.04000000
## Freund 4 30 0.7547222 0.9964286 0.04000000
## Freund 4 40 0.7446032 0.9940476 0.04000000
## Freund 4 50 0.7680556 0.9940476 0.04000000
## Freund 4 60 0.7445635 0.9928571 0.04000000
## Freund 4 70 0.7459127 0.9952381 0.04000000
## Freund 4 80 0.7496825 0.9940476 0.08000000
## Freund 4 90 0.7511905 0.9940476 0.08000000
## Freund 4 100 0.7473413 0.9964286 0.08000000
## Zhu 1 10 0.7661310 0.9988095 0.00000000
## Zhu 1 20 0.8068056 1.0000000 0.00000000
## Zhu 1 30 0.8010119 0.9988095 0.04000000
## Zhu 1 40 0.8072421 0.9976190 0.04000000
## Zhu 1 50 0.8034722 0.9988095 0.04000000
## Zhu 1 60 0.8112500 0.9988095 0.04000000
## Zhu 1 70 0.8091865 0.9976190 0.08000000
## Zhu 1 80 0.8259325 1.0000000 0.08000000
## Zhu 1 90 0.8289286 0.9952381 0.04000000
## Zhu 1 100 0.8295238 0.9964286 0.04000000
## Zhu 2 10 0.7541071 0.9940476 0.08000000
## Zhu 2 20 0.7580952 0.9916667 0.04000000
## Zhu 2 30 0.7646230 0.9916667 0.04000000
## Zhu 2 40 0.7850595 0.9916667 0.04000000
## Zhu 2 50 0.7734325 0.9928571 0.08000000
## Zhu 2 60 0.7822817 0.9928571 0.04000000
## Zhu 2 70 0.7628968 0.9940476 0.04000000
## Zhu 2 80 0.7704762 0.9952381 0.04000000
## Zhu 2 90 0.7678175 0.9952381 0.07333333
## Zhu 2 100 0.7782540 0.9940476 0.07333333
## Zhu 3 10 0.7898810 0.9916667 0.19333333
## Zhu 3 20 0.8089484 0.9904762 0.15333333
## Zhu 3 30 0.8078175 0.9916667 0.11333333
## Zhu 3 40 0.8285714 0.9928571 0.07333333
## Zhu 3 50 0.8134524 0.9940476 0.11333333
## Zhu 3 60 0.8027381 0.9940476 0.07333333
## Zhu 3 70 0.8135317 0.9904762 0.11333333
## Zhu 3 80 0.8203175 0.9916667 0.11333333
## Zhu 3 90 0.8032143 0.9940476 0.11333333
```

```
##   Zhu        3         100     0.8052778  0.9952381  0.11333333
##   Zhu        4          10     0.7496429  0.9928571  0.00000000
##   Zhu        4          20     0.7848413  0.9916667  0.16000000
##   Zhu        4          30     0.7597421  0.9928571  0.12000000
##   Zhu        4          40     0.7871230  0.9916667  0.08000000
##   Zhu        4          50     0.8053968  0.9928571  0.08000000
##   Zhu        4          60     0.7821429  0.9928571  0.04000000
##   Zhu        4          70     0.7785714  0.9928571  0.04000000
##   Zhu        4          80     0.8016667  0.9940476  0.08000000
##   Zhu        4          90     0.7911111  0.9952381  0.04000000
##   Zhu        4         100     0.7867460  0.9952381  0.04000000
## 
## ROC was used to select the optimal model using  the largest value.
## The final values used for the model were mfinal = 100, maxdepth = 1
##  and coeflearn = Zhu.
```

```
confusionMatrix.train(ada.model)
```

```
## Cross-Validated (5 fold) Confusion Matrix 
## 
## (entries are percentual average cell counts across resamples)
##  
##           Reference
## Prediction   No  Yes
##        No  96.4  3.1
##        Yes  0.3  0.1
##                             
##  Accuracy (average) : 0.9654
```

```
plot(varImp(ada.model), main = "Variable importance from Adaboost", col = 2,
lwd = 2)
```



Confusion Matrix for adaboost on test set

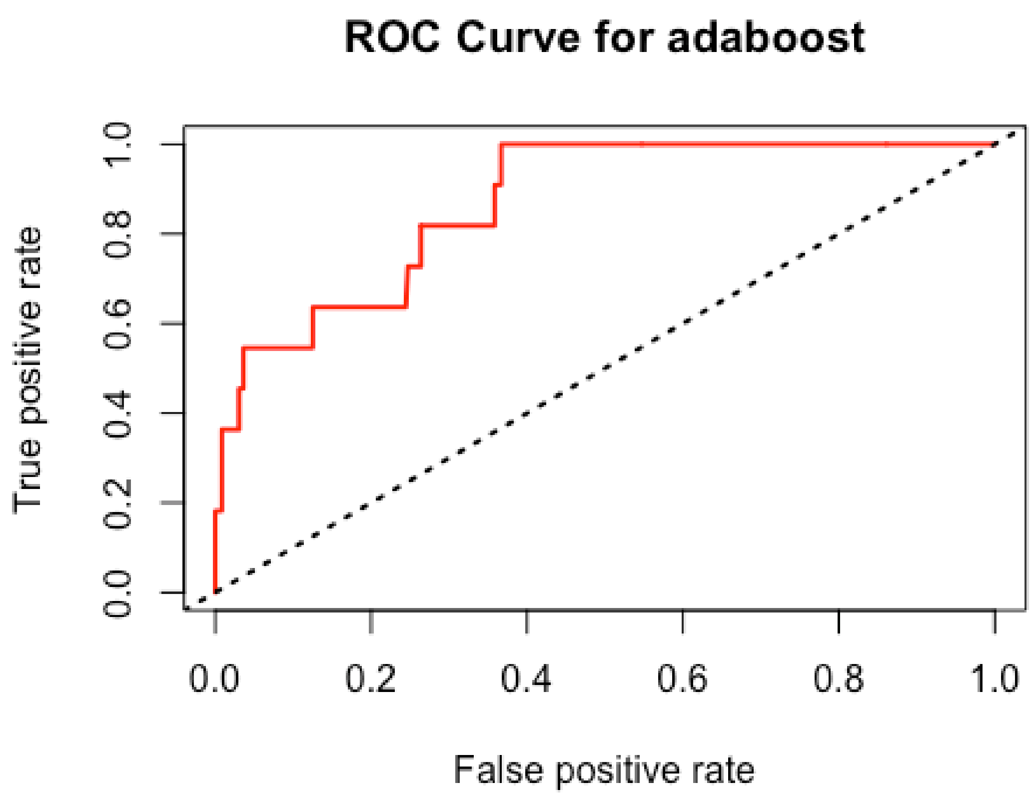
```
caretPredictedClass <- predict(ada.model, model.test, type = "raw")
confusionMatrix(caretPredictedClass,model.test$Manipulater)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  357   8
##        Yes   3   3
##
##                Accuracy : 0.9704
##                  95% CI : (0.9476, 0.9851)
##     No Information Rate : 0.9704
##     P-Value [Acc > NIR] : 0.5793
##
##                   Kappa : 0.3391
##  Mcnemar's Test P-Value : 0.2278
##
##             Sensitivity : 0.9917
##             Specificity : 0.2727
##          Pos Pred Value : 0.9781
```

```
##          Neg Pred Value : 0.5000
##              Prevalence : 0.9704
##          Detection Rate : 0.9623
##    Detection Prevalence : 0.9838
##       Balanced Accuracy : 0.6322
##
##        'Positive' Class : No
##
```

ROC plot for adaboost on test set

```r
ada.pred <- predict(ada.model, model.test, type = "prob")[,2]
ada.prediction <- prediction(ada.pred,model.test$Manipulater)
ada.perf <- performance(ada.prediction, "tpr","fpr")

plot(ada.perf,main="ROC Curve for adaboost",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```




```r
#AUC for the ROC plot
performance(ada.prediction, "auc")

## An object of class "performance"
## Slot "x.name":
## [1] "None"
```

```
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.8688131
##
##
## Slot "alpha.values":
## list()
```

Visulaizing the rules coming out of ada boost. We can loop and print all the trees which was built using boosting. For simplicity, we are printing just one of the trees

To retrieve the understand any model specific attribute, we have to call the **$finalmodel** of the train object created using caret package. This is a generic way to use functions which are model specific. Here **get_tree()** is a function of **fastadaboost** package which cannot be used unless the the object returned is not of adaboost class.

```
#listTreesAda(ada.model$finalModel,3) #this is a function with rattle package
#get_tree(ada.model$finalModel,2)
```

Boosting with adaboost (upsample)

The below code chunk sets some of the control parameters for adaboost

```
objControl <- trainControl(method='cv', number = 5,
                           returnResamp='all',
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           classProbs = TRUE,
                           sampling = "up")#, p = 0.70) #in case method =
#"LGOCV"

search.grid <- expand.grid(mfinal = (1:10)*10, maxdepth = c(1,4),
                    coeflearn = c("Breiman", "Freund", "Zhu"))
```

After setting the control paramters, the model is run

```
set.seed(4121)

ada.up.model <- train(model.train[,1:5], model.train[,6],
                  method='AdaBoost.M1',
                  trControl=objControl,
```

```
                  tuneGrid = search.grid,
                  metric = "ROC",
                  prox=TRUE,allowParallel=TRUE)
```

Confusion Matrix for adaboost on train set

```
#ada.up.model$finalModel #ada.up.model$results
print(ada.up.model)
```

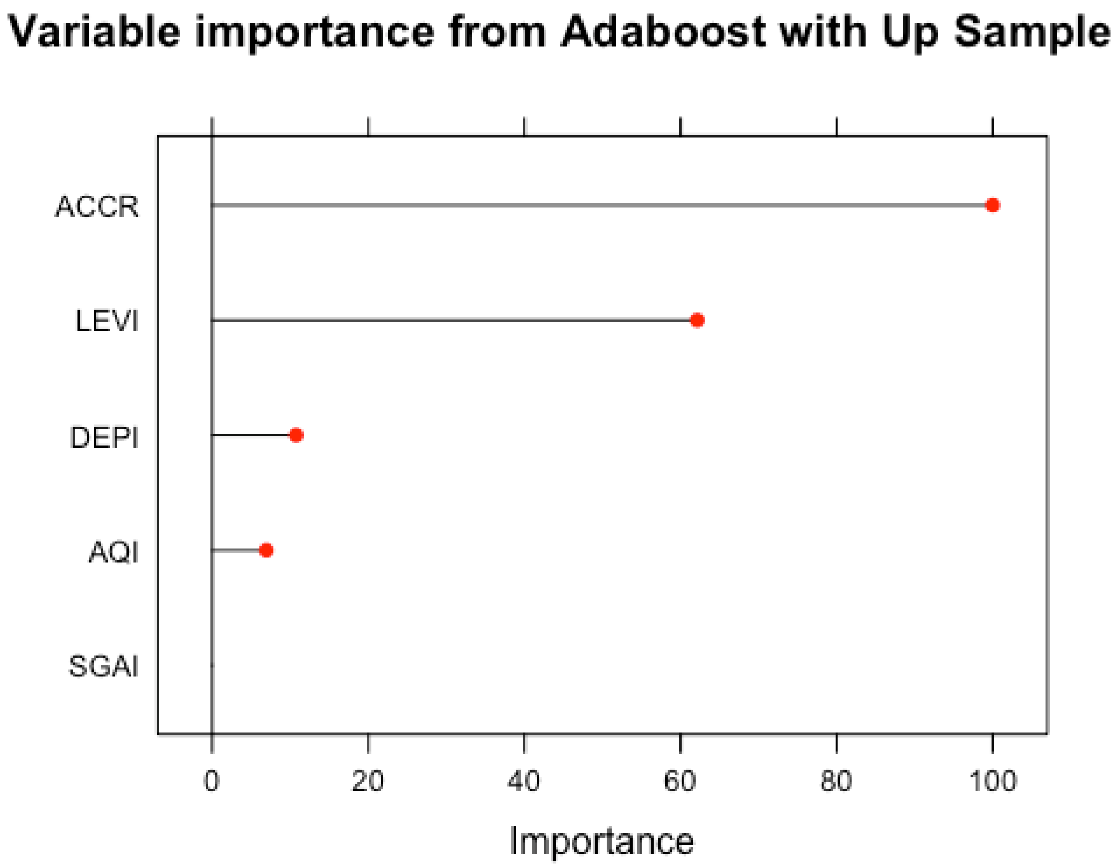
```
## AdaBoost.M1
##
## 868 samples
##   5 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 694, 694, 695, 695, 694
## Addtional sampling using up-sampling
##
## Resampling results across tuning parameters:
##
##   coeflearn  maxdepth  mfinal  ROC        Sens       Spec
##   Breiman    1          10     0.7424008  0.7773810  0.45333333
##   Breiman    1          20     0.7681548  0.8619048  0.48000000
##   Breiman    1          30     0.7830754  0.8940476  0.44000000
##   Breiman    1          40     0.7871032  0.8964286  0.47333333
##   Breiman    1          50     0.8073611  0.8892857  0.47333333
##   Breiman    1          60     0.8104365  0.8702381  0.54666667
##   Breiman    1          70     0.8132937  0.8797619  0.51333333
##   Breiman    1          80     0.8150000  0.8809524  0.44000000
##   Breiman    1          90     0.8240278  0.8857143  0.44000000
##   Breiman    1         100     0.8311111  0.9035714  0.44000000
##   Breiman    4          10     0.7859722  0.9642857  0.32666667
##   Breiman    4          20     0.7650000  0.9809524  0.10666667
##   Breiman    4          30     0.7853770  0.9904762  0.08000000
##   Breiman    4          40     0.7494444  0.9940476  0.08000000
##   Breiman    4          50     0.7717659  0.9928571  0.08000000
##   Breiman    4          60     0.7665079  0.9928571  0.08000000
##   Breiman    4          70     0.7631151  0.9940476  0.08000000
##   Breiman    4          80     0.7758333  0.9952381  0.08000000
##   Breiman    4          90     0.7771429  0.9952381  0.08000000
##   Breiman    4         100     0.7691270  0.9952381  0.08000000
##   Freund     1          10     0.7486508  0.8440476  0.44000000
##   Freund     1          20     0.7921032  0.8928571  0.43333333
##   Freund     1          30     0.8125198  0.8892857  0.48000000
##   Freund     1          40     0.8156548  0.9071429  0.47333333
##   Freund     1          50     0.8273214  0.9023810  0.47333333
##   Freund     1          60     0.8142262  0.9059524  0.44000000
##   Freund     1          70     0.8269643  0.9035714  0.40666667
```

```
##   Freund     1          80     0.8295238  0.9214286  0.44000000
##   Freund     1          90     0.8238294  0.9250000  0.40000000
##   Freund     1         100     0.8304960  0.9238095  0.40666667
##   Freund     4          10     0.7640079  0.9845238  0.11333333
##   Freund     4          20     0.7307143  0.9880952  0.12000000
##   Freund     4          30     0.7142857  0.9928571  0.04000000
##   Freund     4          40     0.7260317  0.9940476  0.00000000
##   Freund     4          50     0.7340079  0.9940476  0.00000000
##   Freund     4          60     0.7467857  0.9964286  0.00000000
##   Freund     4          70     0.7433333  0.9952381  0.00000000
##   Freund     4          80     0.7294444  0.9964286  0.00000000
##   Freund     4          90     0.7433333  0.9964286  0.00000000
##   Freund     4         100     0.7433333  0.9976190  0.00000000
##   Zhu        1          10     0.7429563  0.7797619  0.42000000
##   Zhu        1          20     0.8029762  0.8773810  0.51333333
##   Zhu        1          30     0.8269444  0.8833333  0.34000000
##   Zhu        1          40     0.8317659  0.9083333  0.45333333
##   Zhu        1          50     0.8386706  0.9023810  0.45333333
##   Zhu        1          60     0.8312103  0.9095238  0.38000000
##   Zhu        1          70     0.8355754  0.9190476  0.37333333
##   Zhu        1          80     0.8343849  0.9190476  0.36666667
##   Zhu        1          90     0.8310317  0.9226190  0.34666667
##   Zhu        1         100     0.8304365  0.9238095  0.33333333
##   Zhu        4          10     0.7410317  0.9845238  0.08000000
##   Zhu        4          20     0.7800000  0.9904762  0.04000000
##   Zhu        4          30     0.7726389  0.9928571  0.08000000
##   Zhu        4          40     0.7898016  0.9940476  0.11333333
##   Zhu        4          50     0.7755556  0.9940476  0.11333333
##   Zhu        4          60     0.7697619  0.9940476  0.07333333
##   Zhu        4          70     0.7874603  0.9952381  0.07333333
##   Zhu        4          80     0.8019444  0.9952381  0.07333333
##   Zhu        4          90     0.8022619  0.9964286  0.07333333
##   Zhu        4         100     0.7954365  0.9952381  0.04000000
##
## ROC was used to select the optimal model using  the largest value.
## The final values used for the model were mfinal = 50, maxdepth = 1
##  and coeflearn = Zhu.
```

```
confusionMatrix.train(ada.up.model)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction   No  Yes
##        No  87.3  1.8
##        Yes  9.4  1.4
##
##  Accuracy (average) : 0.8871
```

```
plot(varImp(ada.up.model), main = "Variable importance from Adaboost with Up
Sample", col = 2, lwd = 2)
```



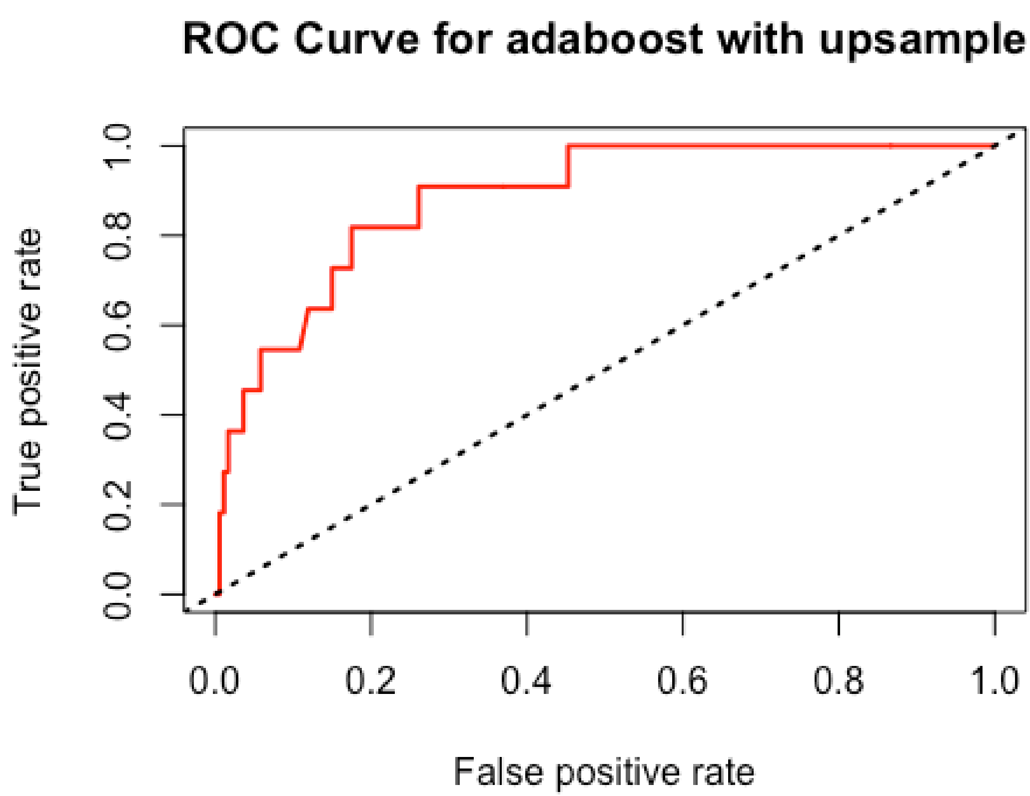
Confusion Matrix for adaboost on test set

```
caretPredictedClass <- predict(ada.up.model, model.test, type = "raw")
confusionMatrix(caretPredictedClass,model.test$Manipulater)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  324   5
##        Yes  36   6
##
##                Accuracy : 0.8895
##                  95% CI : (0.8531, 0.9195)
##     No Information Rate : 0.9704
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1883
##  Mcnemar's Test P-Value : 2.797e-06
##
##             Sensitivity : 0.9000
```

```
##               Specificity : 0.5455
##            Pos Pred Value : 0.9848
##            Neg Pred Value : 0.1429
##                Prevalence : 0.9704
##            Detection Rate : 0.8733
##      Detection Prevalence : 0.8868
##         Balanced Accuracy : 0.7227
##
##          'Positive' Class : No
##
```

ROC plot for adaboost on test set

```
ada.pred <- predict(ada.up.model, model.test, type = "prob")[,2]
ada.prediction <- prediction(ada.pred,model.test$Manipulater)
ada.perf <- performance(ada.prediction, "tpr","fpr")

plot(ada.perf,main="ROC Curve for adaboost with upsample",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```




```
#AUC for the ROC plot
performance(ada.prediction, "auc")
```

```
## An object of class "performance"
## Slot "x.name":
## [1] "None"
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.8830808
##
##
## Slot "alpha.values":
## list()
```

Boosting with adaboost (down sample)

The below code chunk sets some of the control parameters for adaboost

```
objControl <- trainControl(method='cv', number = 5,
                           returnResamp='all',
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           classProbs = TRUE,
                           sampling = "down")#, p = 0.70) #in case method =
#"LGOCV"

search.grid <- expand.grid(mfinal = (1:10)*10, maxdepth = c(1,4),
                    coeflearn = c("Breiman", "Freund", "Zhu"))
```

After setting the control paramters, the model is run

```
set.seed(4121)

ada.down.model <- train(model.train[,1:5], model.train[,6],
                  method='AdaBoost.M1',
                  trControl=objControl,
                  tuneGrid = search.grid,
                  metric = "ROC",
                  prox=TRUE,allowParallel=TRUE)
```

Confusion Matrix for adaboost on train set

```
#ada.down.model$finalModel #ada.down.model$results
print(ada.down.model)
```

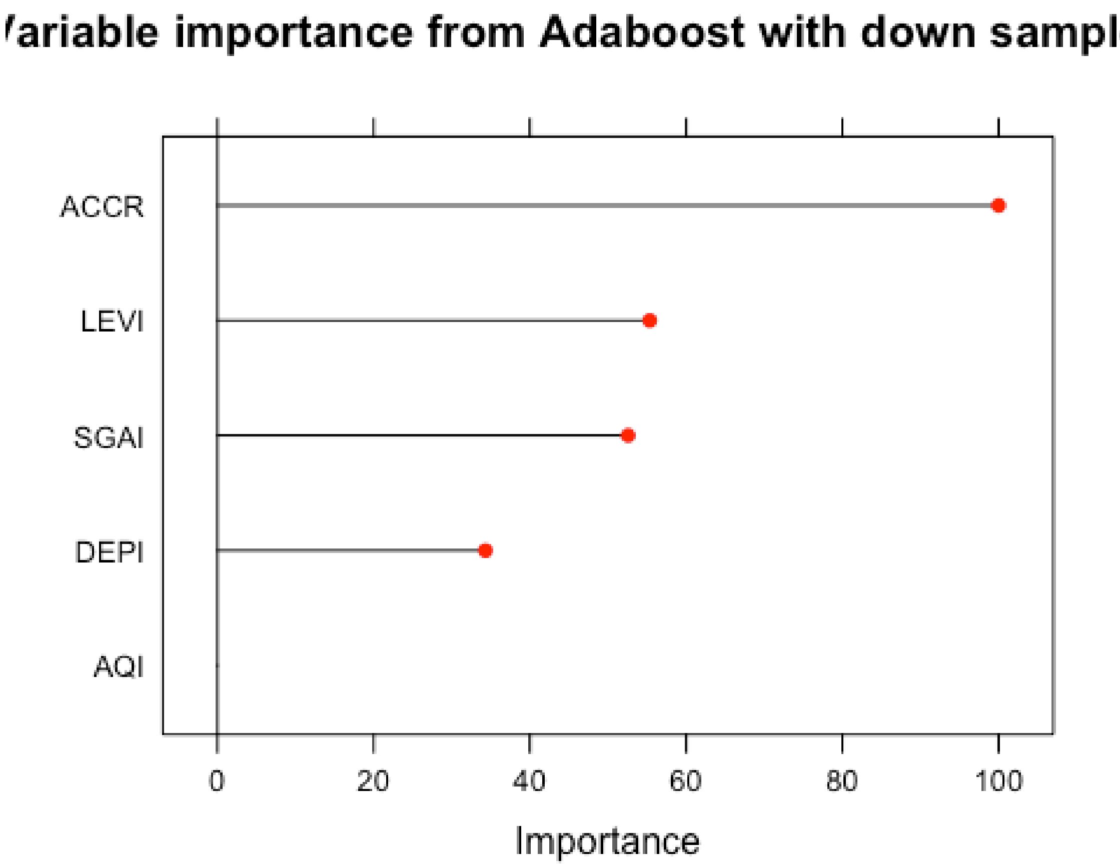
```
## AdaBoost.M1
##
## 868 samples
##   5 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 694, 694, 695, 695, 694
## Addtional sampling using down-sampling
##
## Resampling results across tuning parameters:
##
##   coeflearn  maxdepth  mfinal  ROC        Sens       Spec
##   Breiman    1          10     0.6637302  0.7428571  0.4600000
##   Breiman    1          20     0.7517262  0.6809524  0.6466667
##   Breiman    1          30     0.7637103  0.7035714  0.6866667
##   Breiman    1          40     0.7766468  0.7285714  0.6933333
##   Breiman    1          50     0.7862897  0.7226190  0.7266667
##   Breiman    1          60     0.7950198  0.7119048  0.7266667
##   Breiman    1          70     0.8111310  0.7166667  0.7266667
##   Breiman    1          80     0.8055357  0.7047619  0.6866667
##   Breiman    1          90     0.8035516  0.7071429  0.7266667
##   Breiman    1         100     0.8049802  0.7142857  0.7266667
##   Breiman    4          10     0.7152579  0.6630952  0.5866667
##   Breiman    4          20     0.7137897  0.6726190  0.6200000
##   Breiman    4          30     0.7113095  0.6666667  0.6266667
##   Breiman    4          40     0.7100000  0.6619048  0.6266667
##   Breiman    4          50     0.7190079  0.6726190  0.6266667
##   Breiman    4          60     0.7240079  0.6702381  0.6266667
##   Breiman    4          70     0.7206746  0.6738095  0.6266667
##   Breiman    4          80     0.7140079  0.6738095  0.6266667
##   Breiman    4          90     0.7174603  0.6785714  0.6266667
##   Breiman    4         100     0.7055952  0.6750000  0.5866667
##   Freund     1          10     0.7517460  0.7369048  0.6800000
##   Freund     1          20     0.7357738  0.7095238  0.6533333
##   Freund     1          30     0.7486706  0.7023810  0.6466667
##   Freund     1          40     0.7544643  0.7202381  0.6866667
##   Freund     1          50     0.7305357  0.7107143  0.6866667
##   Freund     1          60     0.7229365  0.7011905  0.6866667
##   Freund     1          70     0.7344048  0.7071429  0.6466667
##   Freund     1          80     0.7320238  0.7226190  0.6866667
##   Freund     1          90     0.7315873  0.7142857  0.6866667
##   Freund     1         100     0.7346032  0.7309524  0.6866667
##   Freund     4          10     0.7218849  0.6726190  0.6866667
##   Freund     4          20     0.7469841  0.6535714  0.7266667
##   Freund     4          30     0.7349206  0.6821429  0.6933333
##   Freund     4          40     0.7491270  0.6702381  0.6600000
##   Freund     4          50     0.7450397  0.6773810  0.6533333
##   Freund     4          60     0.7404365  0.6785714  0.6533333
```

```
##   Freund     4          70     0.7453968  0.6726190  0.6200000
##   Freund     4          80     0.7607937  0.6761905  0.7333333
##   Freund     4          90     0.7596825  0.6785714  0.7000000
##   Freund     4         100     0.7594048  0.6797619  0.7000000
##   Zhu        1          10     0.7325595  0.7797619  0.4866667
##   Zhu        1          20     0.7503968  0.7642857  0.5533333
##   Zhu        1          30     0.7474405  0.7261905  0.5533333
##   Zhu        1          40     0.7803968  0.7547619  0.6266667
##   Zhu        1          50     0.7689087  0.7428571  0.5533333
##   Zhu        1          60     0.7661310  0.7571429  0.5533333
##   Zhu        1          70     0.7575992  0.7321429  0.5533333
##   Zhu        1          80     0.7558333  0.7511905  0.4866667
##   Zhu        1          90     0.7531746  0.7535714  0.4866667
##   Zhu        1         100     0.7559524  0.7511905  0.4866667
##   Zhu        4          10     0.7341667  0.6702381  0.5933333
##   Zhu        4          20     0.7180556  0.6714286  0.5200000
##   Zhu        4          30     0.7307143  0.6750000  0.5866667
##   Zhu        4          40     0.7415873  0.6773810  0.5466667
##   Zhu        4          50     0.7350794  0.6797619  0.5466667
##   Zhu        4          60     0.7513095  0.6821429  0.5866667
##   Zhu        4          70     0.7598413  0.6809524  0.6266667
##   Zhu        4          80     0.7600397  0.6904762  0.6266667
##   Zhu        4          90     0.7626587  0.6785714  0.6266667
##   Zhu        4         100     0.7632937  0.6892857  0.6266667
##
## ROC was used to select the optimal model using  the largest value.
## The final values used for the model were mfinal = 70, maxdepth = 1
##  and coeflearn = Breiman.
```

```
confusionMatrix.train(ada.down.model)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction   No  Yes
##        No  69.4  0.9
##        Yes 27.4  2.3
##
##  Accuracy (average) : 0.7166
```

```
plot(varImp(ada.down.model), main = "Variable importance from Adaboost with
down sample", col = 2, lwd = 2)
```



Confusion Matrix for adaboost on test set

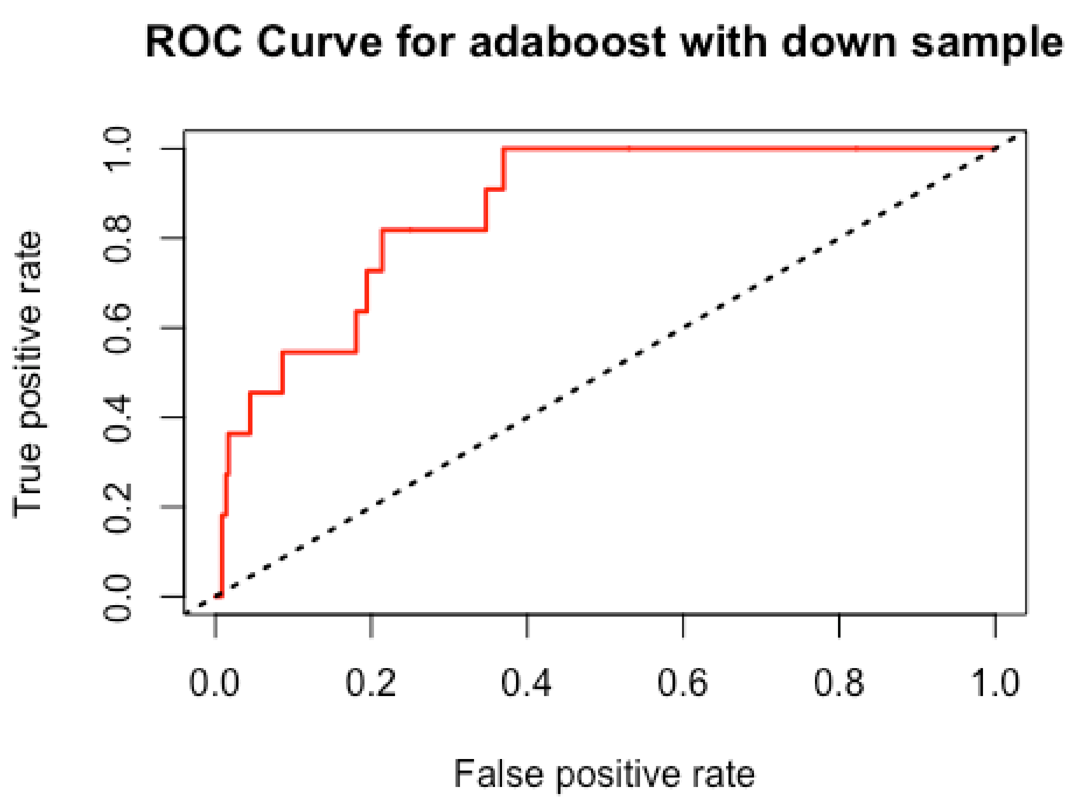
```
caretPredictedClass <- predict(ada.down.model, model.test, type = "raw")
confusionMatrix(caretPredictedClass,model.test$Manipulater)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  276   2
##        Yes  84   9
##
##                Accuracy : 0.7682
##                  95% CI : (0.7219, 0.8102)
##     No Information Rate : 0.9704
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1268
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.76667
##             Specificity : 0.81818
##          Pos Pred Value : 0.99281
```

```
##          Neg Pred Value : 0.09677
##              Prevalence : 0.97035
##          Detection Rate : 0.74394
##    Detection Prevalence : 0.74933
##       Balanced Accuracy : 0.79242
##
##        'Positive' Class : No
##
```

ROC plot for adaboost on test set

```
ada.pred <- predict(ada.down.model, model.test, type = "prob")[,2]
ada.prediction <- prediction(ada.pred,model.test$Manipulater)
ada.perf <- performance(ada.prediction, "tpr","fpr")

plot(ada.perf,main="ROC Curve for adaboost with down sample",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```




```
#AUC for the ROC plot
performance(ada.prediction, "auc")

## An object of class "performance"
## Slot "x.name":
## [1] "None"
```

```
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.8651515
##
##
## Slot "alpha.values":
## list()
```

Boosting with adaboost (SMOTE)

The below code chunk sets some of the control parameters for adaboost

```
objControl <- trainControl(method='cv', number = 5,
                           returnResamp='all',
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           classProbs = TRUE,
                           sampling = "smote")#, p = 0.70) #in case method =
#"LGOCV"

search.grid <- expand.grid(mfinal = (1:10)*10, maxdepth = c(1,4),
                    coeflearn = c("Breiman", "Freund", "Zhu"))
```

After setting the control paramters, the model is run

```
set.seed(4121)

ada.smote.model <- train(model.train[,1:5], model.train[,6],
                  method='AdaBoost.M1',
                  trControl=objControl,
                  tuneGrid = search.grid,
                  metric = "ROC",
                  prox=TRUE,allowParallel=TRUE)
```

Confusion Matrix for adaboost on train set

```
#ada.smote.model$finalModel #ada.smote.model$results
print(ada.smote.model)

## AdaBoost.M1
##
```

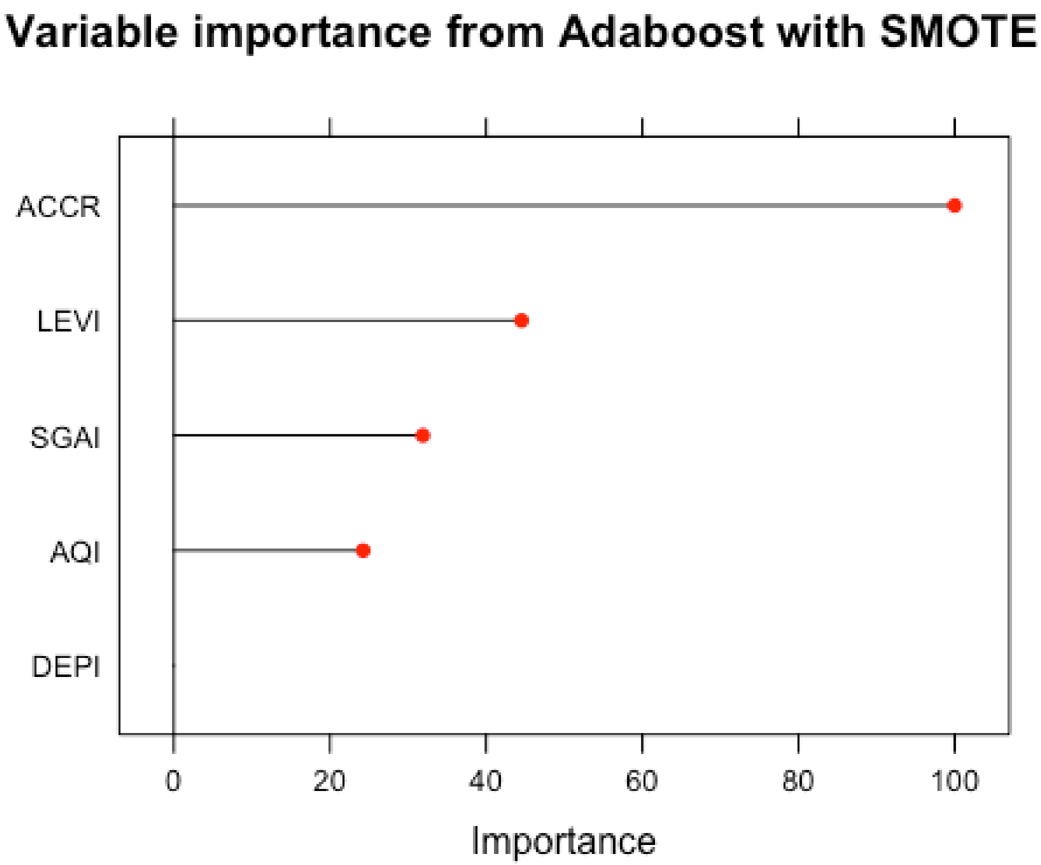
```
## 868 samples
##   5 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 694, 694, 695, 695, 694
## Addtional sampling using SMOTE
##
## Resampling results across tuning parameters:
##
##   coeflearn  maxdepth  mfinal  ROC        Sens       Spec
##   Breiman    1          10     0.6778373  0.8309524  0.1800000
##   Breiman    1          20     0.6997222  0.8630952  0.2933333
##   Breiman    1          30     0.7099206  0.8345238  0.4333333
##   Breiman    1          40     0.7137897  0.8428571  0.4733333
##   Breiman    1          50     0.7251190  0.8428571  0.5466667
##   Breiman    1          60     0.7078770  0.8345238  0.4666667
##   Breiman    1          70     0.7107738  0.8404762  0.3666667
##   Breiman    1          80     0.7137302  0.8440476  0.3666667
##   Breiman    1          90     0.7183333  0.8392857  0.4400000
##   Breiman    1         100     0.7109524  0.8345238  0.3933333
##   Breiman    4          10     0.7352976  0.8333333  0.5466667
##   Breiman    4          20     0.7168651  0.8250000  0.3933333
##   Breiman    4          30     0.7277976  0.8333333  0.4266667
##   Breiman    4          40     0.7169444  0.8321429  0.4333333
##   Breiman    4          50     0.7156349  0.8357143  0.4266667
##   Breiman    4          60     0.7167063  0.8333333  0.4266667
##   Breiman    4          70     0.7073016  0.8428571  0.4333333
##   Breiman    4          80     0.7074603  0.8488095  0.4333333
##   Breiman    4          90     0.7040079  0.8452381  0.4000000
##   Breiman    4         100     0.7043254  0.8523810  0.4266667
##   Freund     1          10     0.7111310  0.8535714  0.1466667
##   Freund     1          20     0.7645437  0.8392857  0.4400000
##   Freund     1          30     0.7633333  0.8345238  0.4066667
##   Freund     1          40     0.7717063  0.8440476  0.4133333
##   Freund     1          50     0.7737897  0.8488095  0.4800000
##   Freund     1          60     0.7667460  0.8547619  0.4133333
##   Freund     1          70     0.7803770  0.8476190  0.4066667
##   Freund     1          80     0.7693452  0.8452381  0.4466667
##   Freund     1          90     0.7738690  0.8488095  0.4800000
##   Freund     1         100     0.7840476  0.8452381  0.4466667
##   Freund     4          10     0.6876190  0.8130952  0.4933333
##   Freund     4          20     0.6890476  0.8178571  0.4000000
##   Freund     4          30     0.6813889  0.8166667  0.4000000
##   Freund     4          40     0.6799603  0.8178571  0.4066667
##   Freund     4          50     0.6597619  0.8154762  0.4000000
##   Freund     4          60     0.6605159  0.8178571  0.3266667
##   Freund     4          70     0.6546825  0.8297619  0.3666667
##   Freund     4          80     0.6575794  0.8273810  0.4066667
```

```
##   Freund     4         90     0.6600000  0.8392857  0.4066667
##   Freund     4        100     0.6570238  0.8380952  0.4066667
##   Zhu        1         10     0.6970040  0.8119048  0.4400000
##   Zhu        1         20     0.6982738  0.8190476  0.4066667
##   Zhu        1         30     0.7344048  0.8261905  0.4533333
##   Zhu        1         40     0.7393849  0.8452381  0.4533333
##   Zhu        1         50     0.7592063  0.8250000  0.5533333
##   Zhu        1         60     0.7596429  0.8333333  0.5200000
##   Zhu        1         70     0.7656349  0.8273810  0.5200000
##   Zhu        1         80     0.7622619  0.8261905  0.5533333
##   Zhu        1         90     0.7612103  0.8333333  0.5200000
##   Zhu        1        100     0.7614683  0.8333333  0.5533333
##   Zhu        4         10     0.6713492  0.8202381  0.4333333
##   Zhu        4         20     0.6626190  0.8333333  0.4000000
##   Zhu        4         30     0.6821429  0.8309524  0.3933333
##   Zhu        4         40     0.7145635  0.8404762  0.3933333
##   Zhu        4         50     0.7238095  0.8404762  0.3266667
##   Zhu        4         60     0.7354762  0.8511905  0.3266667
##   Zhu        4         70     0.7357143  0.8476190  0.3266667
##   Zhu        4         80     0.7423810  0.8357143  0.3266667
##   Zhu        4         90     0.7305952  0.8440476  0.3266667
##   Zhu        4        100     0.7332540  0.8392857  0.3266667
##
## ROC was used to select the optimal model using  the largest value.
## The final values used for the model were mfinal = 100, maxdepth = 1
##  and coeflearn = Freund.
```

```
confusionMatrix.train(ada.smote.model)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##             Reference
## Prediction    No  Yes
##        No   81.8  1.8
##        Yes  15.0  1.4
##
##  Accuracy (average) : 0.8318
```

```
plot(varImp(ada.smote.model), main = "Variable importance from Adaboost with
SMOTE", col = 2, lwd = 2)
```



Confusion Matrix for adaboost on test set

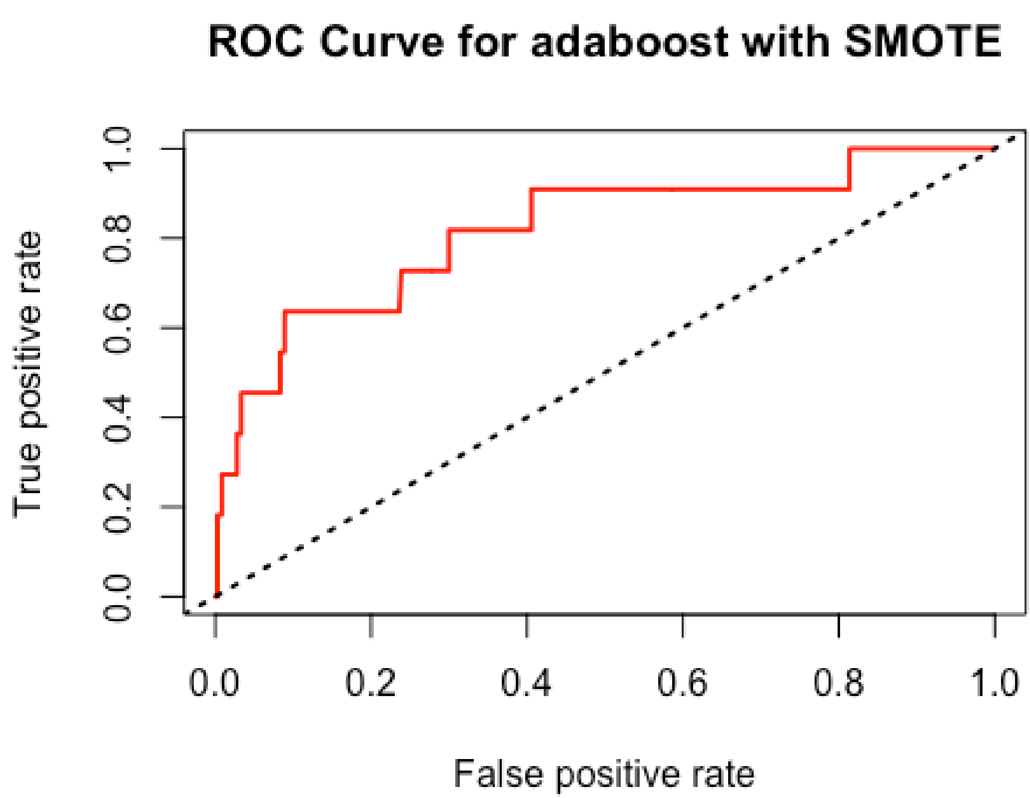
```
caretPredictedClass <- predict(ada.smote.model, model.test, type = "raw")
confusionMatrix(caretPredictedClass,model.test$Manipulater)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  284   4
##        Yes  76   7
##
##                Accuracy : 0.7844
##                  95% CI : (0.739, 0.8251)
##     No Information Rate : 0.9704
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1019
##  Mcnemar's Test P-Value : 2.054e-15
##
##             Sensitivity : 0.78889
##             Specificity : 0.63636
##          Pos Pred Value : 0.98611
```

```
##          Neg Pred Value : 0.08434
##              Prevalence : 0.97035
##          Detection Rate : 0.76550
##    Detection Prevalence : 0.77628
##       Balanced Accuracy : 0.71263
##
##        'Positive' Class : No
##
```

ROC plot for adaboost on test set

```
ada.pred <- predict(ada.smote.model, model.test, type = "prob")[,2]
ada.prediction <- prediction(ada.pred,model.test$Manipulater)
ada.perf <- performance(ada.prediction, "tpr","fpr")

plot(ada.perf,main="ROC Curve for adaboost with SMOTE",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```




```
#AUC for the ROC plot
performance(ada.prediction, "auc")

## An object of class "performance"
## Slot "x.name":
## [1] "None"
```

```
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.817803
##
##
## Slot "alpha.values":
## list()
```

Boosting with xgboost (normal)

The below code chunk sets some of the control parameters for adaboost

```
objControl <- trainControl(method='cv', number = 5,
                           returnResamp='final',
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           classProbs = TRUE)
```

1. Refer to know about the fine tuning parameters.
2. This can also be referred to know about the parameter fine tuning.

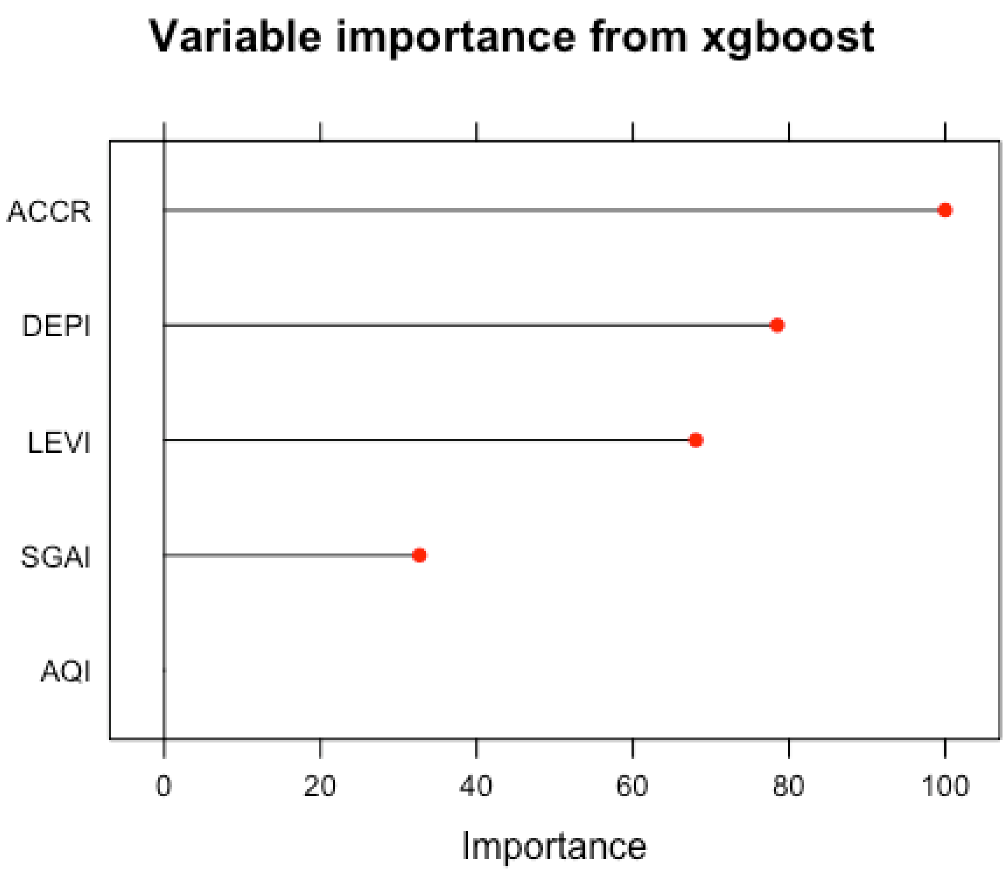
```
search.grid <- expand.grid(nrounds = (5:20)*10, max_depth = c(1:5)*2,
                    eta = c(0.01,0.03,0.09,0.3,0.5),
                    gamma = c(0:3)/100,
                    colsample_bytree = c(2:5)/10,
                    min_child_weight = c(1:6))
                    #subsample = c(0.5:1))
```

After setting the control paramters, the model is run

```
set.seed(4121)

xg.model <- train(model.train[,1:5], model.train[,6],
                  method='xgbTree',
                  trControl=objControl,
                  tuneGrid = search.grid,
                  metric = "ROC",
                  prox=TRUE,allowParallel=TRUE)

## Loading required package: xgboost
```

Confusion Matrix for xgboost on train set

```r
#print(xg.model)
confusionMatrix.train(xg.model)

## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction   No  Yes
##        No  96.8  3.1
##        Yes  0.0  0.1
##
##  Accuracy (average) : 0.9689

plot(varImp(xg.model), main = "Variable importance from xgboost", col = 2,
lwd = 2)
```



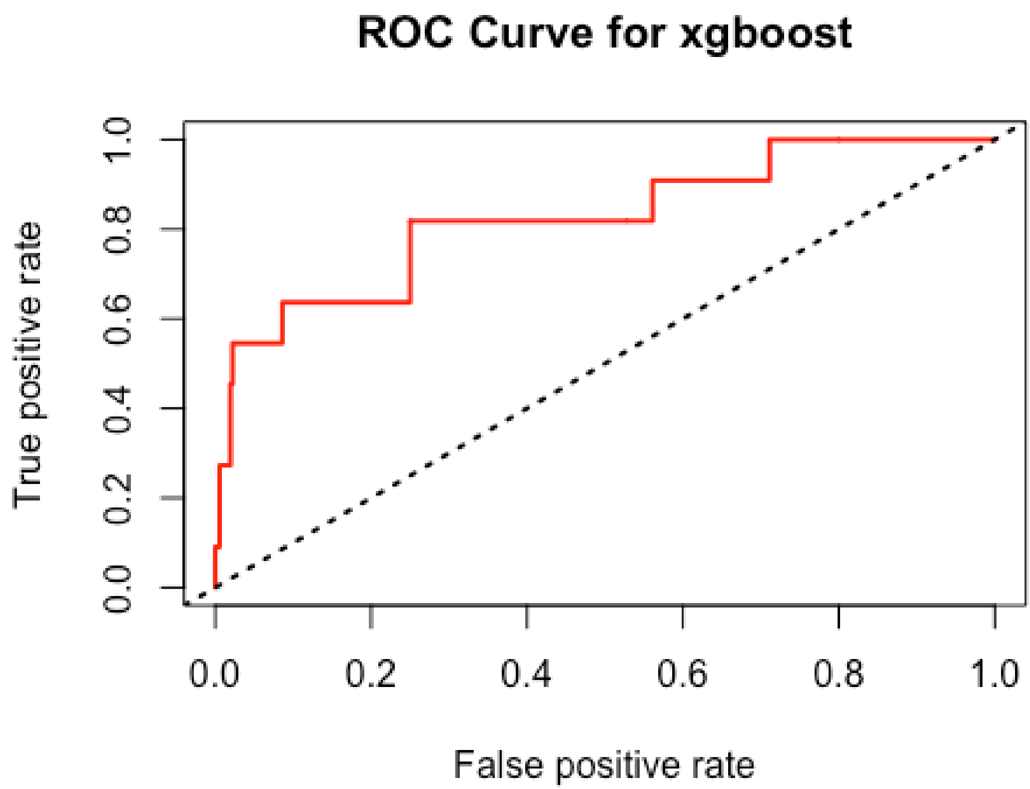
Confusion Matrix for xgboost on test set

```r
caretPredictedClass <- predict(xg.model, model.test[1:5], type = "raw")
confusionMatrix(caretPredictedClass,model.test$Manipulater)

## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction  No Yes
##        No  360  11
##        Yes   0   0
##
##                Accuracy : 0.9704
##                  95% CI : (0.9476, 0.9851)
##     No Information Rate : 0.9704
##     P-Value [Acc > NIR] : 0.579276
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : 0.002569
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9704
##          Neg Pred Value :    NaN
##              Prevalence : 0.9704
##          Detection Rate : 0.9704
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : No
##
```

ROC plot for xgboost on test set

```
xg.pred <- predict(xg.model, model.test[1:5], type = "prob")[,2]
xg.prediction <- prediction(xg.pred,model.test$Manipulater)
xg.perf <- performance(xg.prediction, "tpr","fpr")

plot(xg.perf,main="ROC Curve for xgboost",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```




```
#AUC for the ROC plot
performance(xg.prediction, "auc")

## An object of class "performance"
## Slot "x.name":
## [1] "None"
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.8244949
##
##
## Slot "alpha.values":
## list()
```

Boosting with xgboost (up sample)

The below code chunk sets some of the control parameters for adaboost

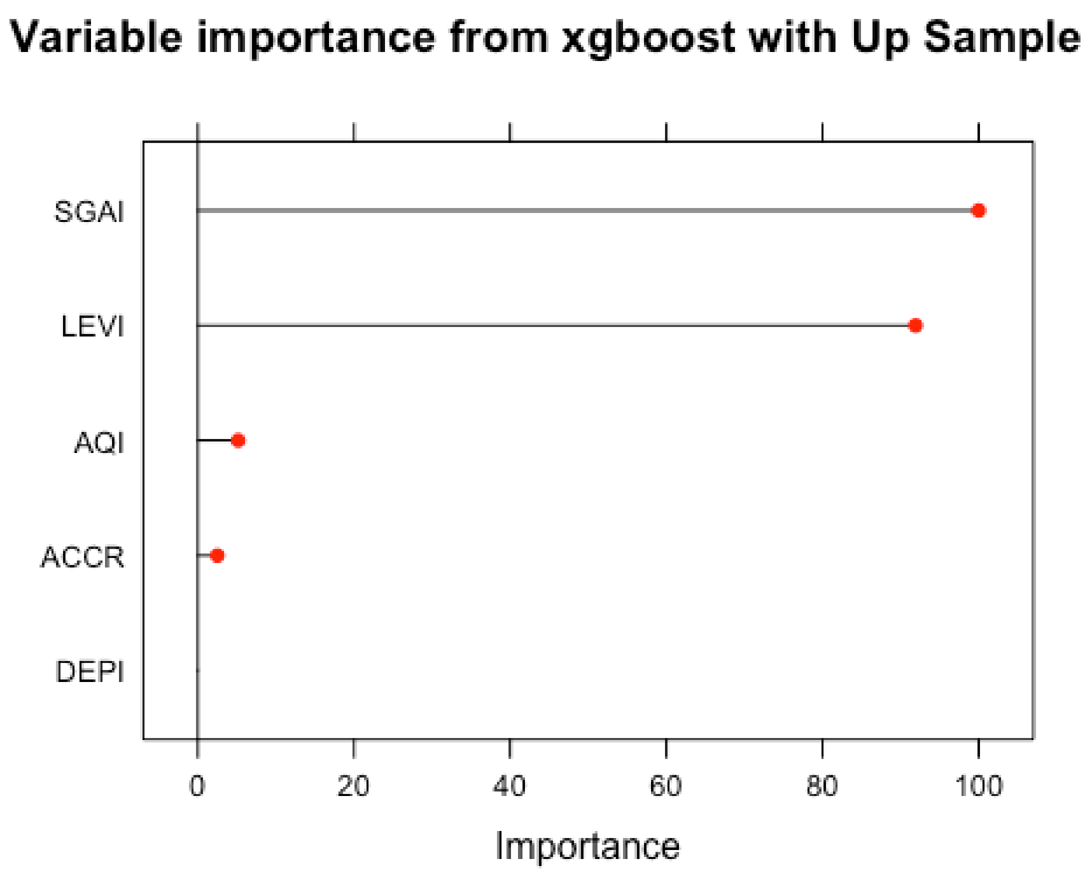
```
objControl <- trainControl(method='cv', number = 5,
                           returnResamp='final',
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           classProbs = TRUE, sampling = "up")

search.grid <- expand.grid(nrounds = (5:20)*10, max_depth = c(1:5)*2,
                    eta = c(0.01,0.03,0.09,0.3,0.5),
                    gamma = c(0:3)/100,
                    colsample_bytree = c(2:5)/10,
                    min_child_weight = c(1:6))
                    #subsample = c(0.5:1))
```

After setting the control paramters, the model is run

```
set.seed(4121)

xg.up.model <- train(model.train[,1:5], model.train[,6],
                  method='xgbTree',
                  trControl=objControl,
                  tuneGrid = search.grid,
                  metric = "ROC",
                  prox=TRUE,allowParallel=TRUE)
```

Confusion Matrix for xgboost on train set

```
#print(xg.up.model)
confusionMatrix.train(xg.up.model)

## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction   No  Yes
##        No  95.6  2.5
##        Yes  1.2  0.7
##
##  Accuracy (average) : 0.9631

plot(varImp(xg.up.model), main = "Variable importance from xgboost with Up
Sample", col = 2, lwd = 2)
```



Confusion Matrix for xgboost on test set

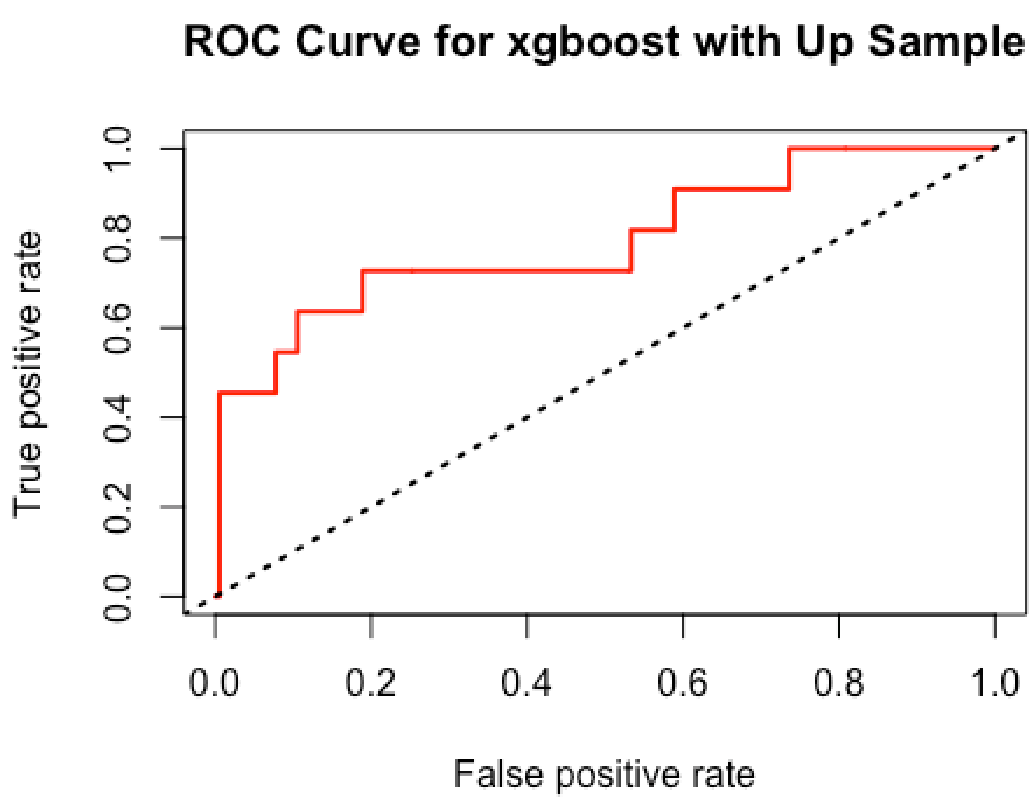
```
caretPredictedClass <- predict(xg.up.model, model.test[1:5], type = "raw")
confusionMatrix(caretPredictedClass,model.test$Manipulater)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   No Yes
##        No   358   8
##        Yes    2   3
##
##                  Accuracy : 0.973
##                    95% CI : (0.951, 0.987)
##       No Information Rate : 0.9704
##       P-Value [Acc > NIR] : 0.4581
##
##                     Kappa : 0.3632
##    Mcnemar's Test P-Value : 0.1138
##
##               Sensitivity : 0.9944
##               Specificity : 0.2727
##            Pos Pred Value : 0.9781
```

```
##          Neg Pred Value : 0.6000
##              Prevalence : 0.9704
##          Detection Rate : 0.9650
##    Detection Prevalence : 0.9865
##       Balanced Accuracy : 0.6336
##
##        'Positive' Class : No
##
```

ROC plot for xgboost on test set

```
xg.pred <- predict(xg.up.model, model.test[1:5], type = "prob")[,2]
xg.prediction <- prediction(xg.pred,model.test$Manipulater)
xg.perf <- performance(xg.prediction, "tpr","fpr")

plot(xg.perf,main="ROC Curve for xgboost with Up Sample",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```




```
#AUC for the ROC plot
performance(xg.prediction, "auc")

## An object of class "performance"
## Slot "x.name":
## [1] "None"
```

```
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.794697
##
##
## Slot "alpha.values":
## list()
```

Boosting with xgboost (down sample)

The below code chunk sets some of the control parameters for adaboost

```
objControl <- trainControl(method='cv', number = 5,
                           returnResamp='final',
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           classProbs = TRUE, sampling = "down")

search.grid <- expand.grid(nrounds = (5:20)*10, max_depth = c(1:5)*2,
                    eta = c(0.01,0.03,0.09,0.3,0.5),
                    gamma = c(0:3)/100,
                    colsample_bytree = c(2:5)/10,
                    min_child_weight = c(1:6))
                    #subsample = c(0.5:1))
```

After setting the control paramters, the model is run

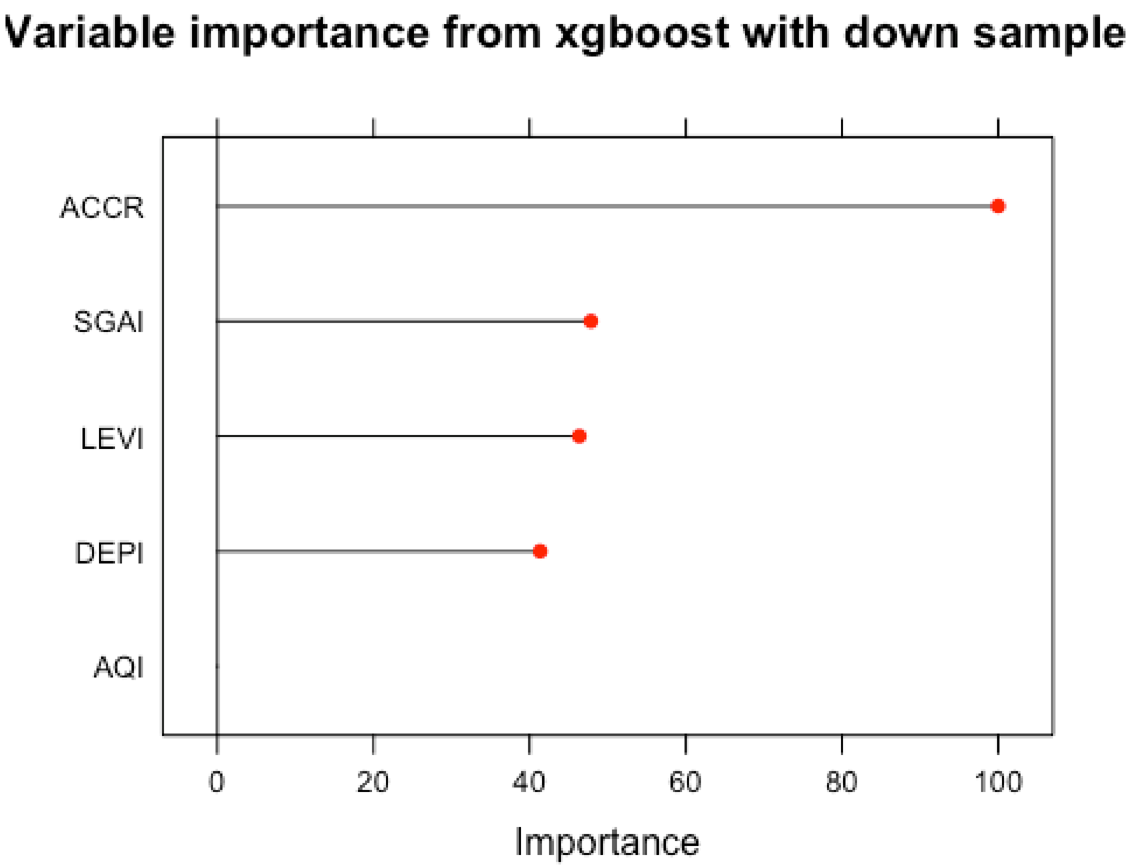
```
set.seed(4121)

xg.down.model <- train(model.train[,1:5], model.train[,6],
                  method='xgbTree',
                  trControl=objControl,
                  tuneGrid = search.grid,
                  metric = "ROC",
                  prox=TRUE,allowParallel=TRUE)
```

Confusion Matrix for xgboost on train set

```
#print(xg.down.model)
confusionMatrix.train(xg.down.model)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction   No  Yes
##        No  69.4  0.6
##        Yes 27.4  2.6
##
##  Accuracy (average) : 0.72
```

```
plot(varImp(xg.down.model), main = "Variable importance from xgboost with
down sample", col = 2, lwd = 2)
```



Confusion Matrix for xgboost on test set

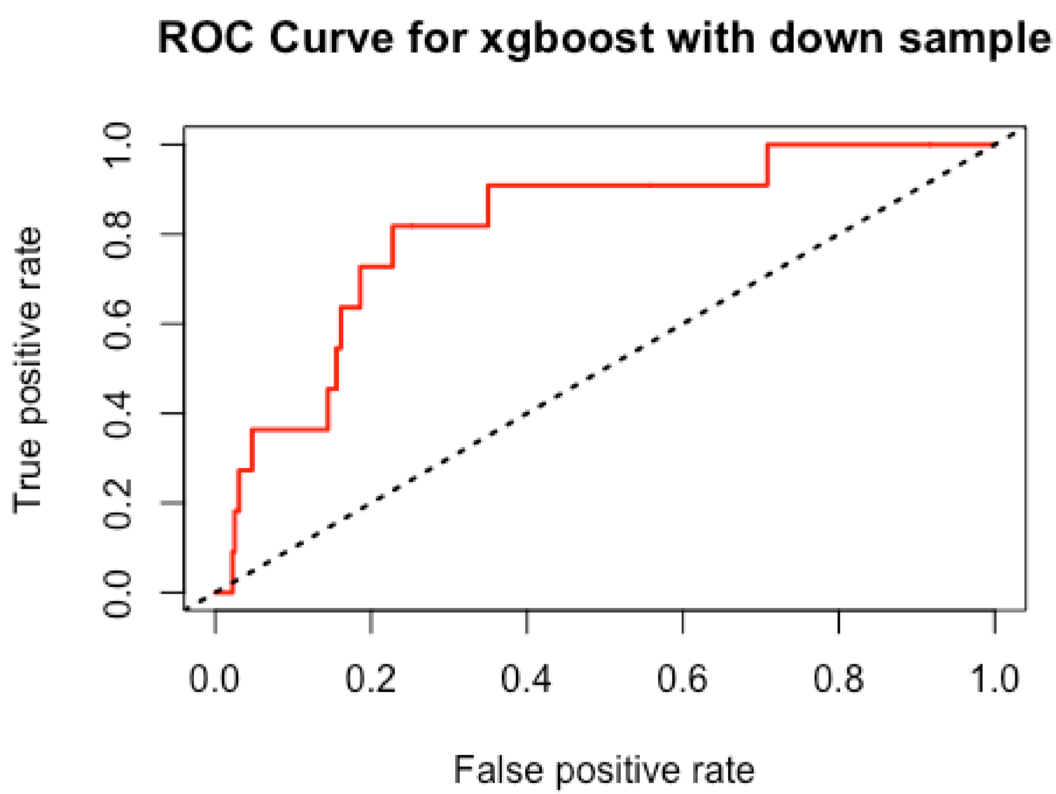
```
caretPredictedClass <- predict(xg.down.model, model.test[1:5], type = "raw")
confusionMatrix(caretPredictedClass,model.test$Manipulater)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  244   2
```

```
##        Yes 116   9
##
##                Accuracy : 0.6819
##                  95% CI : (0.6319, 0.7291)
##     No Information Rate : 0.9704
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0823
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6778
##             Specificity : 0.8182
##          Pos Pred Value : 0.9919
##          Neg Pred Value : 0.0720
##              Prevalence : 0.9704
##          Detection Rate : 0.6577
##    Detection Prevalence : 0.6631
##       Balanced Accuracy : 0.7480
##
##        'Positive' Class : No
##
```

ROC plot for xgboost on test set

```
xg.pred <- predict(xg.down.model, model.test[1:5], type = "prob")[,2]
xg.prediction <- prediction(xg.pred,model.test$Manipulater)
xg.perf <- performance(xg.prediction, "tpr","fpr")

plot(xg.perf,main="ROC Curve for xgboost with down sample",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```




```
#AUC for the ROC plot
performance(xg.prediction, "auc")

## An object of class "performance"
## Slot "x.name":
## [1] "None"
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.8128788
##
##
## Slot "alpha.values":
## list()
```

Boosting with xgboost (SMOTE)

The below code chunk sets some of the control parameters for adaboost

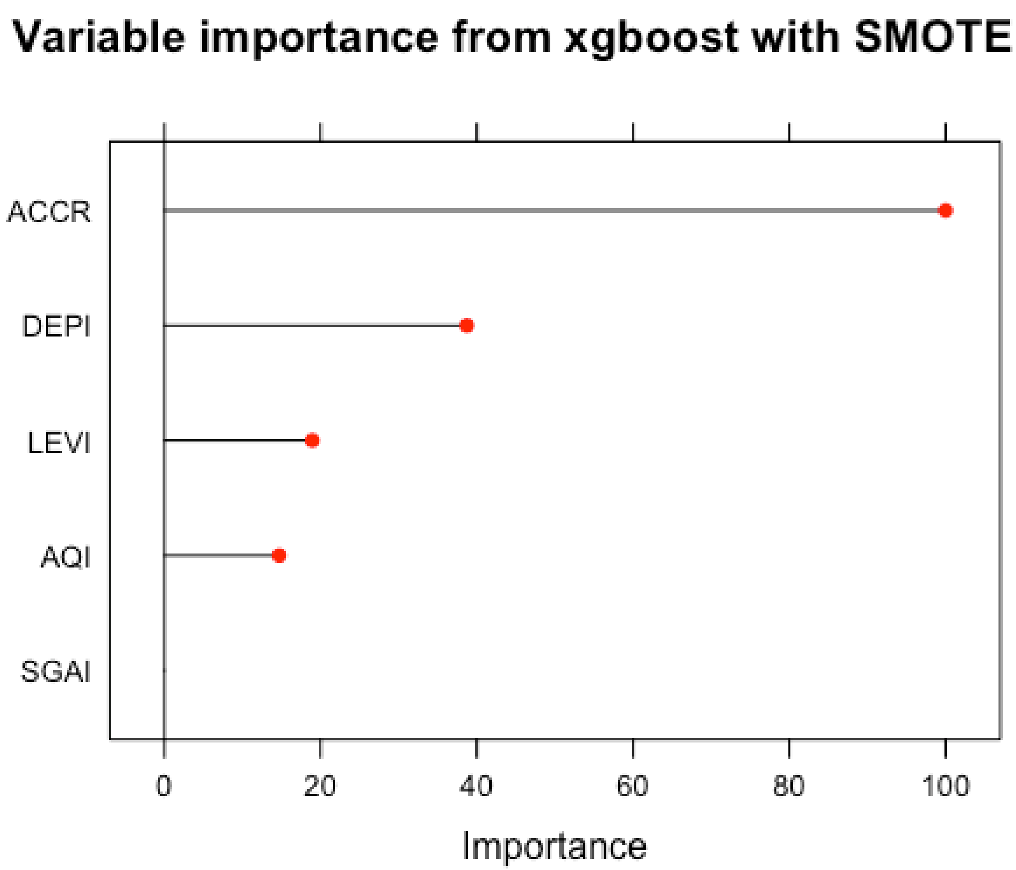
```
objControl <- trainControl(method='cv', number = 5,
                           returnResamp='final',
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           classProbs = TRUE, sampling = "smote")

search.grid <- expand.grid(nrounds = (5:20)*10, max_depth = c(1:5)*2,
                    eta = c(0.01,0.03,0.09,0.3,0.5),
                    gamma = c(0:3)/100,
                    colsample_bytree = c(2:5)/10,
                    min_child_weight = c(1:6))
                    #subsample = c(0.5:1))
```

After setting the control paramters, the model is run

```
set.seed(4121)

xg.smote.model <- train(model.train[,1:5], model.train[,6],
                  method='xgbTree',
                  trControl=objControl,
                  tuneGrid = search.grid,
                  metric = "ROC",
                  prox=TRUE,allowParallel=TRUE)
```

Confusion Matrix for xgboost on train set

```
#print(xg.smote.model)
confusionMatrix.train(xg.smote.model)

## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction   No  Yes
##        No  78.3  0.7
##        Yes 18.4  2.5
##
##  Accuracy (average) : 0.8088

plot(varImp(xg.smote.model), main = "Variable importance from xgboost with
SMOTE", col = 2, lwd = 2)
```



Confusion Matrix for xgboost on test set

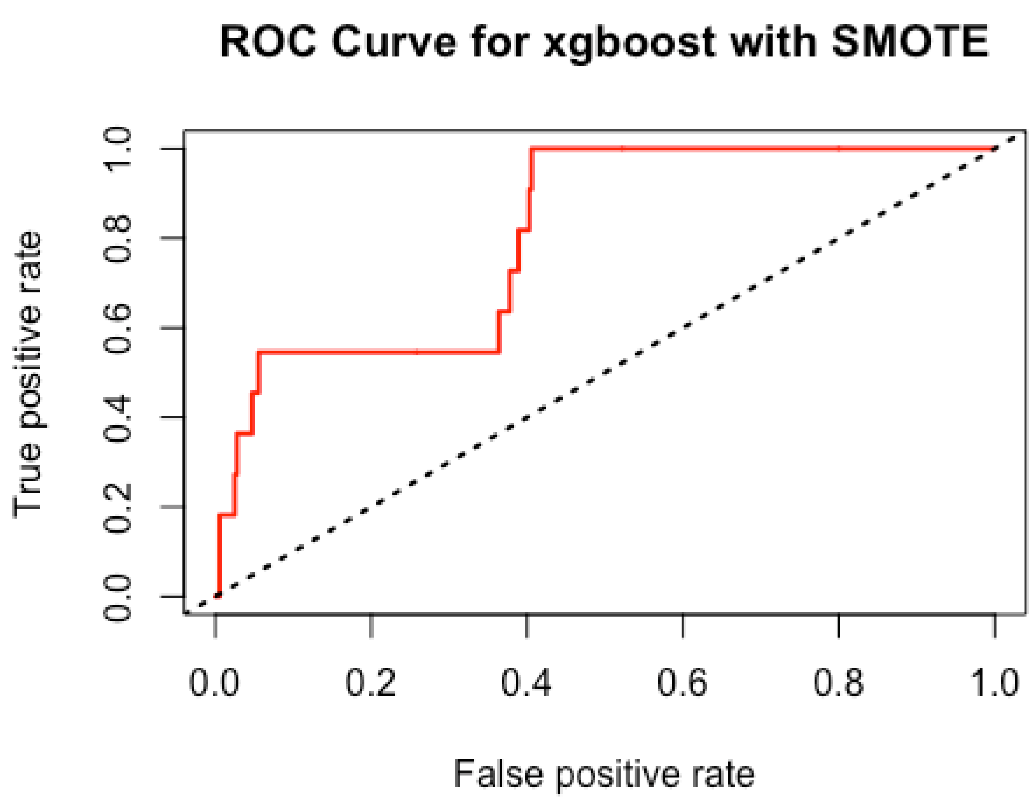
```
caretPredictedClass <- predict(xg.smote.model, model.test[1:5], type = "raw")
confusionMatrix(caretPredictedClass,model.test$Manipulater)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   No Yes
##        No   273   5
##        Yes   87   6
##
##                Accuracy : 0.752
##                  95% CI : (0.7048, 0.7951)
##     No Information Rate : 0.9704
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0658
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.75833
##             Specificity : 0.54545
##          Pos Pred Value : 0.98201
```

```
##          Neg Pred Value : 0.06452
##              Prevalence : 0.97035
##          Detection Rate : 0.73585
##    Detection Prevalence : 0.74933
##       Balanced Accuracy : 0.65189
##
##        'Positive' Class : No
##
```

ROC plot for xgboost on test set

```
xg.pred <- predict(xg.smote.model, model.test[1:5], type = "prob")[,2]
xg.prediction <- prediction(xg.pred,model.test$Manipulater)
xg.perf <- performance(xg.prediction, "tpr","fpr")

plot(xg.perf,main="ROC Curve for xgboost with SMOTE",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```




```
#AUC for the ROC plot
performance(xg.prediction, "auc")

## An object of class "performance"
## Slot "x.name":
## [1] "None"
```

```
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.8085859
##
##
## Slot "alpha.values":
## list()
```

Neural Network

Neural network implementation to find the manipulaters

The below code chunk sets some of the control parameters

```
objControl <- trainControl(method='cv', number = 5,
                           returnResamp='none',
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           classProbs = TRUE)
```

Using search grid to fine tune the neural network. **Size** fine tunes number of hidden units to tune and **decay** fine tunes weight decay

```
search.grid <- expand.grid(.decay = c(0.5, 0.1, 0.05), .size = c(2, 3,
4,5,6,7))
```

After setting the control paramters, the model is run. If we use **linout=TRUE** in **train()** the neural network builds a regression model. **linout=FALSE** will make **nnet** use a sigmodial function and all the predictions will be contrained between **[0,1]**

```
set.seed(4121)

nn.objModel <- train(model.train[,1:5], model.train[,6],
                     method='nnet',
                     trControl=objControl,
                     metric = "ROC",
                     maxit = 1000,
                     tuneGrid = search.grid,
                     trace = FALSE,
                     linout = FALSE)
```

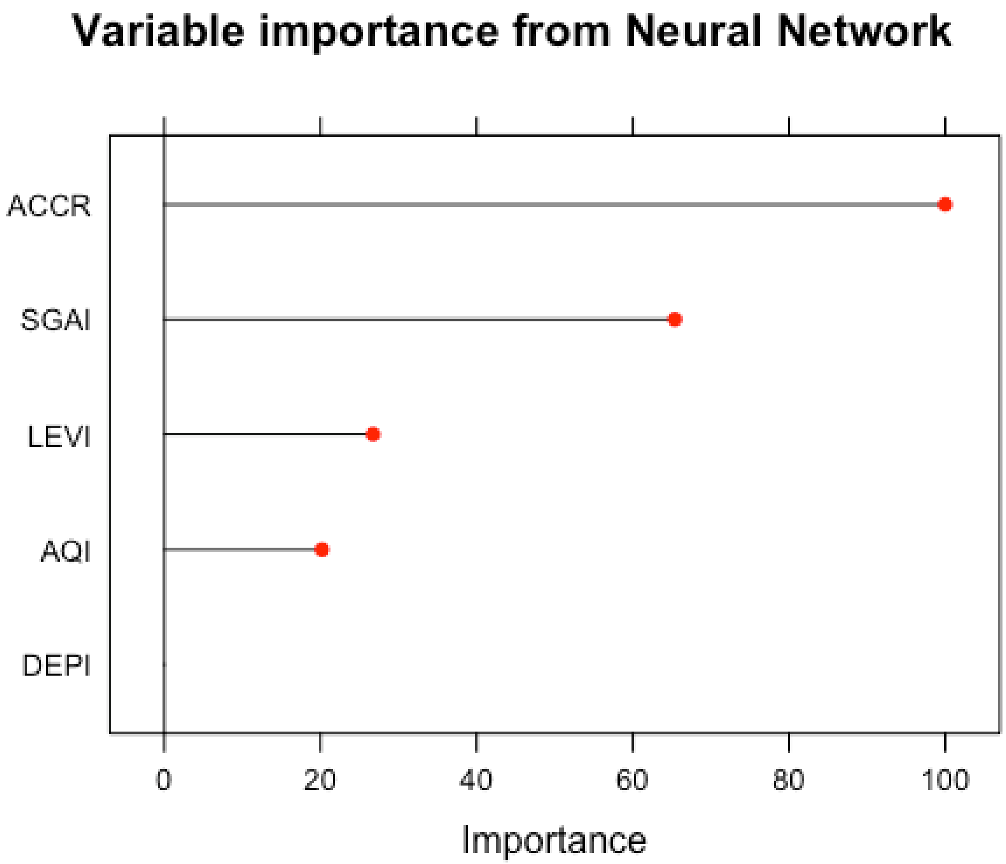
```
## Loading required package: nnet
```

Confusion Matrix for Neural Network on train set

```
#nn.objModel$finalModel #nn.objModel$results
print(nn.objModel)

## Neural Network
##
## 868 samples
##   5 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 694, 694, 695, 695, 694
## Resampling results across tuning parameters:
##
##   decay  size  ROC        Sens       Spec
##   0.05   2     0.6983730  0.9988095  0.12000000
##   0.05   3     0.7337302  0.9976190  0.16000000
##   0.05   4     0.7138889  0.9988095  0.18666667
##   0.05   5     0.7085317  0.9976190  0.16000000
##   0.05   6     0.7146032  0.9988095  0.12000000
##   0.05   7     0.6818651  0.9964286  0.03333333
##   0.10   2     0.7306746  0.9976190  0.12000000
##   0.10   3     0.6857937  0.9988095  0.00000000
##   0.10   4     0.7160714  0.9988095  0.04000000
##   0.10   5     0.7577381  0.9988095  0.08000000
##   0.10   6     0.7207143  0.9988095  0.00000000
##   0.10   7     0.7078571  0.9988095  0.08000000
##   0.50   2     0.6075000  1.0000000  0.00000000
##   0.50   3     0.5620635  1.0000000  0.00000000
##   0.50   4     0.5827778  1.0000000  0.00000000
##   0.50   5     0.5321429  1.0000000  0.00000000
##   0.50   6     0.5087302  1.0000000  0.00000000
##   0.50   7     0.6161508  1.0000000  0.00000000
##
## ROC was used to select the optimal model using  the largest value.
## The final values used for the model were size = 5 and decay = 0.1.

confusionMatrix.train(nn.objModel)

## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction   No  Yes
##        No  96.7  3.0
##        Yes  0.1  0.2
```

```
##
##  Accuracy (average) : 0.9689
```

```
plot(varImp(nn.objModel), main = "Variable importance from Neural Network",
col = 2, lwd = 2)
```



Confusion Matrix for Neural Network on test set

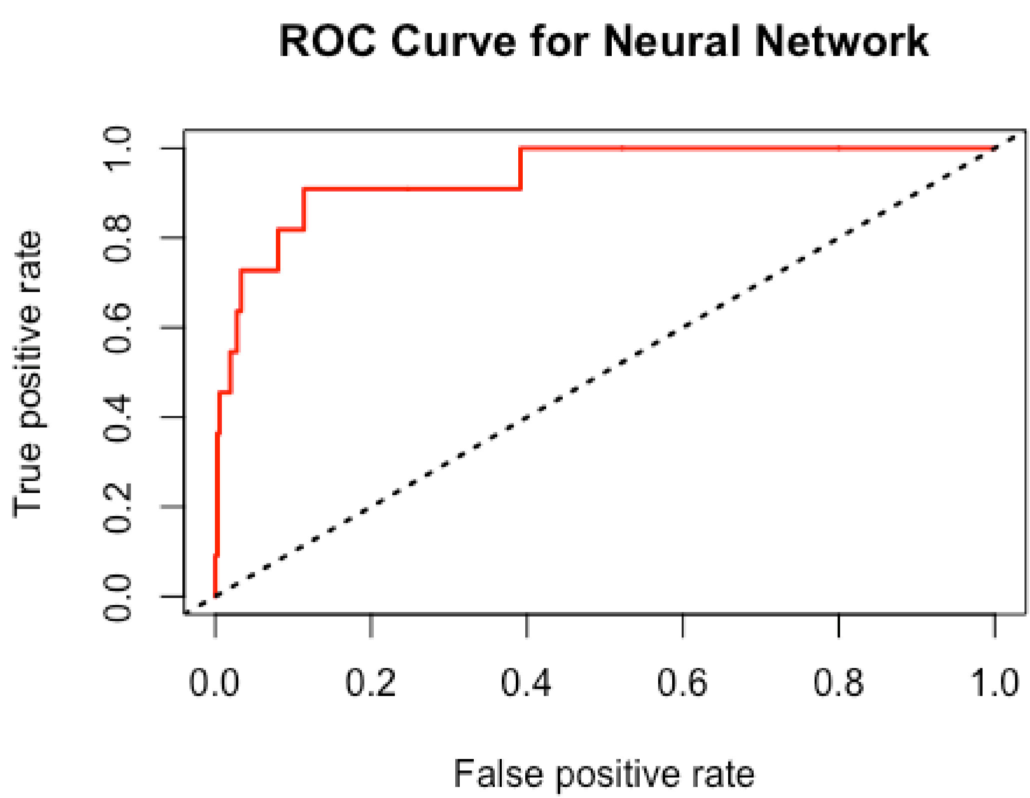
```
caretPredictedClass <- predict(nn.objModel, model.test, type = "raw")
confusionMatrix(caretPredictedClass,model.test$Manipulater)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  360  11
##        Yes   0   0
##
##                Accuracy : 0.9704
##                  95% CI : (0.9476, 0.9851)
##     No Information Rate : 0.9704
##     P-Value [Acc > NIR] : 0.579276
##
##                   Kappa : 0
```

```
##  Mcnemar's Test P-Value : 0.002569
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9704
##          Neg Pred Value :    NaN
##              Prevalence : 0.9704
##          Detection Rate : 0.9704
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : No
##
```

ROC plot for Neural Network on test set

```
nn.pred <- predict(nn.objModel, model.test, type = "prob")[,2]
nn.prediction <- prediction(nn.pred,model.test$Manipulater)
nn.perf <- performance(nn.prediction, "tpr","fpr")

plot(nn.perf,main="ROC Curve for Neural Network",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```



```r
#AUC for the ROC plot
performance(nn.prediction, "auc")
```

```
## An object of class "performance"
## Slot "x.name":
## [1] "None"
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.9381313
##
##
## Slot "alpha.values":
## list()
```

Logistic Regression

The variables DSRI and GMI causes fitted probability to be numerically 0 or 1. Using less number of variables in the logistic regression.

```r
lg.model.data <- as.data.frame(filter.data[,c(#"DSRI",
                                            #"GMI",
                                            "AQI",
                                            "SGI",
                                            "DEPI",
                                            "SGAI",
                                            "ACCR",
                                            "LEVI",
                                            "Manipulater"
)])
lg.train.data <- as.data.frame(data.train[,c(#"DSRI",
                                            #"GMI",
                                            "AQI",
                                            "SGI",
                                            "DEPI",
                                            "SGAI",
                                            "ACCR",
                                            "LEVI",
                                            "Manipulater"
)])
lg.test.data <- as.data.frame(data.test[,c(#"DSRI",
```

```
                                                       #"GMI",
                                                       "AQI",
                                                       "SGI",
                                                       "DEPI",
                                                       "SGAI",
                                                       "ACCR",
                                                       "LEVI",
                                                       "Manipulater"
)])
```

The below code chunk sets some of the control parameters

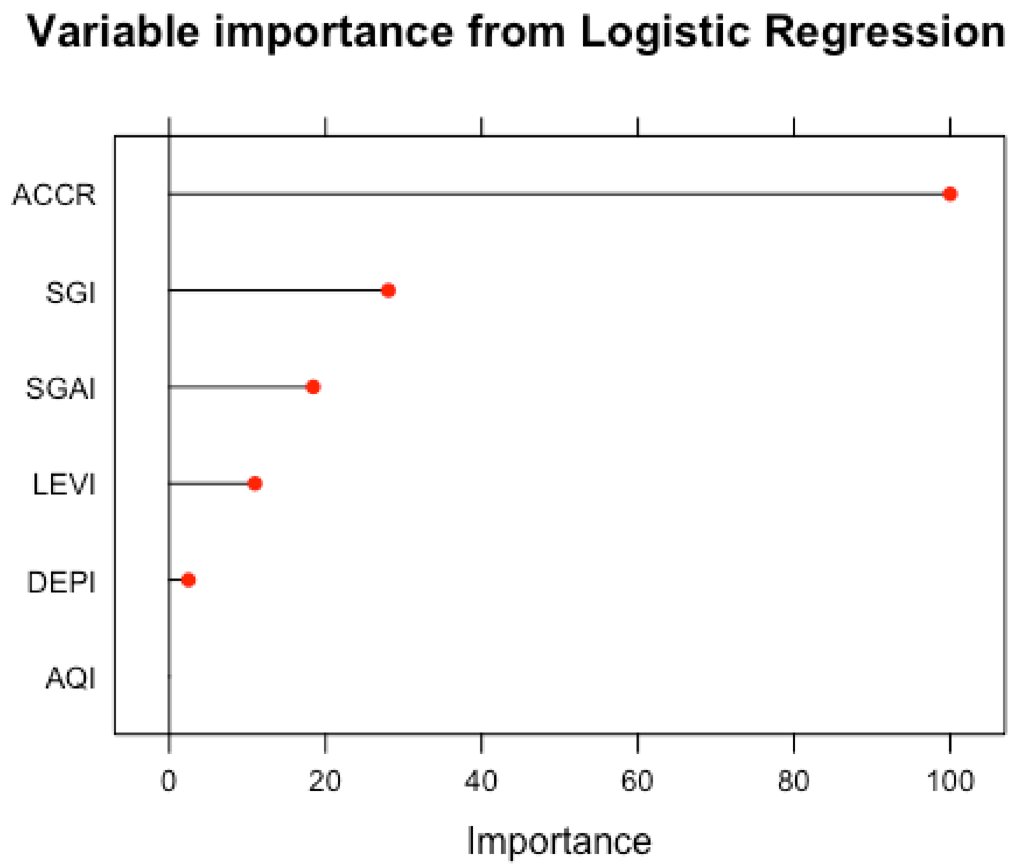
```
objControl <- trainControl(method='cv', number=5,
                           returnResamp='none',
                           summaryFunction = twoClassSummary,
                           savePredictions = TRUE,
                           classProbs = TRUE)
```

After setting the control paramters, the model is run

Confusion Matrix for logistic regression on train set

```
print(lg.objModel)

## Generalized Linear Model with Stepwise Feature Selection
##
## 868 samples
##   6 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 694, 694, 695, 695, 694
## Resampling results:
##
##   ROC        Sens       Spec
##   0.6880159  0.9964286  0.04
##
##

confusionMatrix.train(lg.objModel)

## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction   No  Yes
##        No  96.4  3.1
##        Yes  0.3  0.1
##
##  Accuracy (average) : 0.9654
```

```r
plot(varImp(lg.objModel), main = "Variable importance from Logistic
Regression", col = 2, lwd = 2)
```



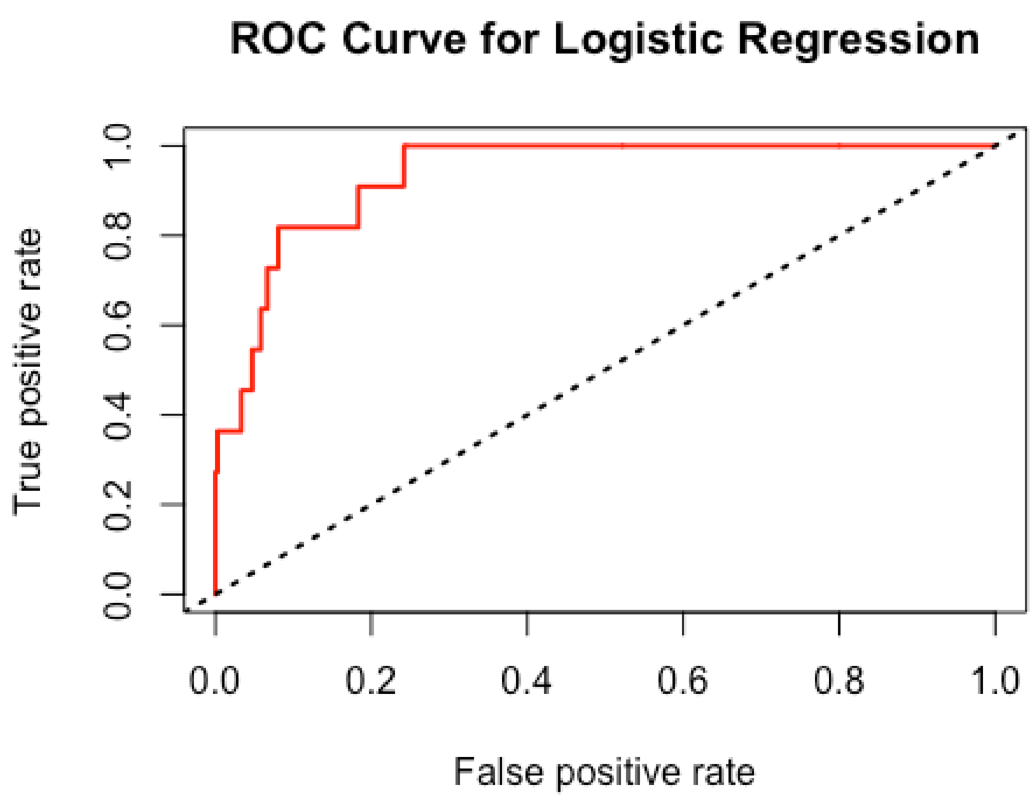
Confusion Matrix for logistic regression on test set

```r
caretPredictedClass <- predict(lg.objModel, lg.test.data, type = "raw")
confusionMatrix(caretPredictedClass,lg.test.data$Manipulater)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  360   9
##        Yes   0   2
##
##                Accuracy : 0.9757
##                  95% CI : (0.9545, 0.9888)
##     No Information Rate : 0.9704
##     P-Value [Acc > NIR] : 0.337237
##
##                   Kappa : 0.3013
##  Mcnemar's Test P-Value : 0.007661
##
##             Sensitivity : 1.0000
```

```
##               Specificity : 0.1818
##            Pos Pred Value : 0.9756
##            Neg Pred Value : 1.0000
##                Prevalence : 0.9704
##            Detection Rate : 0.9704
##      Detection Prevalence : 0.9946
##         Balanced Accuracy : 0.5909
##
##          'Positive' Class : No
##
```

ROC plot for logistic regression

```
lg.pred <- predict(lg.objModel, lg.test.data, type = "prob")[,2]
lg.prediction <- prediction(lg.pred,lg.test.data$Manipulater)
lg.perf <- performance(lg.prediction, "tpr","fpr")

plot(lg.perf,main="ROC Curve for Logistic Regression",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=3,col="black")
```




```
#AUC for the ROC plot
performance(lg.prediction, "auc")
```

```
## An object of class "performance"
## Slot "x.name":
## [1] "None"
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.935101
##
##
## Slot "alpha.values":
## list()
```

End of document
