Insurance_X_Classification_Exercise.R

puj83

Mon May 11 15:09:25 2020

```
# Some notes before starting:
#   * Read all the way through the instructions.
# * Models must be built using Python, R, or SAS.
# * New features can be created.
# * Users cannot add or supplement with external data.
# * While simple techniques may develop adequate models, success in this
exercise typically involves feature engineering and model tuning.
# * Throughout your code, please use comments to document your thought
process as you move through exploratory data analysis, feature engineering,
model tuning, etc.
# * Please review your submission against the submission expectations.
#
#
# Step 1 - Clean and prepare your data:
#   There are several entries where values have been deleted to simulate
dirty data. Please clean the data with whatever method(s) you believe is
best/most suitable. Note that some of the missing values are truly blank
(unknown answers).  Success in this exercise typically involves feature
engineering and avoiding data leakage.
#
# Step 2 - Build your models:
#   Please use two different machine learning/statistical algorithms to
develop a total of two models. Please include comments that document choices
you make (such as those for feature engineering and for model tuning).
#
# Step 3 - Generate predictions:
#   Create predictions on the data in test.csv using each of your trained
models.  The predictions should be the class probabilities for belonging to
the positive class (labeled '1').
#
# Be sure to output a prediction for each of the rows in the test dataset
(10K rows).  Save the results of each of your models in a separate CSV file.
Title the two files 'results1.csv' and 'results2.csv'.  A result file should
each have a single column representing the output from one model (no header
label or index column is needed).
#
# Step 4 - Compare your modeling approaches:
#   Please prepare a relatively short write-up comparing the pros and cons of
the two algorithms you used (PDF preferred). As part of the write-up, please
identify which algorithm you think will perform the best. For the best
performing model, are there choices you made in the context of the exercise
that might be different in a business context? How would explain to a
```

```r
business partner the concept that one model is better than the other?
#
#   Step 5 - Submit your work:
#   Your submission should consist of all the code used for exploratory data
analysis, cleaning, prepping, and modeling (text, html, or pdf preferred),
the two result files (.csv format - each containing 10,000 decimal
probabilities), and your write-up comparing the pros and cons of the two
modeling techniques used (text, html, or pdf preferred). Note: The results
files should not include the original data, only the probabilities.
#
# Your work will be scored on techniques used (appropriateness and
complexity), evaluation of the two techniques compared in the write-up, model
performance on the data hold out  - measured by AUC, and your overall
code/comments.  The threshold for passing model performance is set high,
expecting that model tuning and feature engineering will be used.  The best
score of the two models submitted will be used.
#
# Please do not submit the original data back to us.

# install.packages("pillar")
# install.packages("dplyr")
# install.packages("tibble")
# install.packages("pdflatex")
# install.packages("ggpubr")
# install.packages("neuralnet")
# install.packages("ada")

library(pillar)
library(dplyr)

## Warning: package 'dplyr' was built under R version 3.5.3

##
## Attaching package: 'dplyr'

## The following object is masked from 'package:pillar':
##
##     dim_desc

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(ggplot2)
# library(pdflatex)
library(reshape2)
library(ggcorrplot)
```

```
## Warning: package 'ggcorrplot' was built under R version 3.5.3

library(randomForest)

## Warning: package 'randomForest' was built under R version 3.5.3

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine

library(factoextra)

## Warning: package 'factoextra' was built under R version 3.5.3

## Welcome! Want to learn more? See two factoextra-related books at
https://goo.gl/ve3WBa

library(ggpubr)
library(neuralnet)

## Warning: package 'neuralnet' was built under R version 3.5.3

##
## Attaching package: 'neuralnet'

## The following object is masked from 'package:dplyr':
##
##     compute

library(ada)

## Warning: package 'ada' was built under R version 3.5.3

## Loading required package: rpart

#############################################################################
##########################################
# Importing the data
#############################################################################
##########################################

SF_Train<-read.csv(file =
"C:/Users/puj83/OneDrive/CV/Cases/InsuranceX/exercise_04_train.csv", header =
```

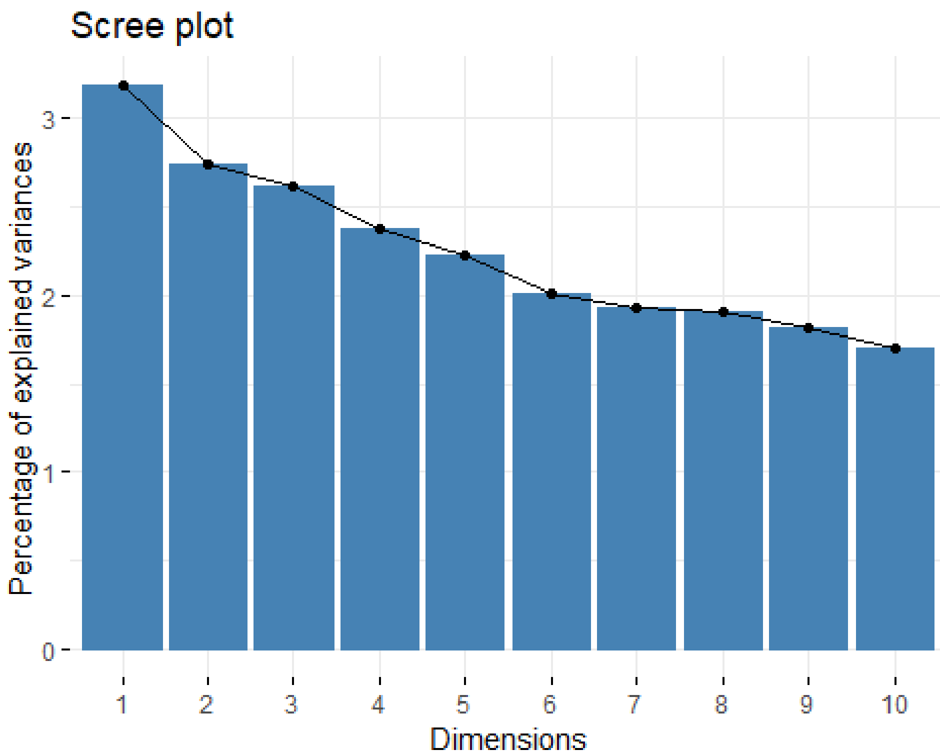
```r
T, sep = ",")
SF_Test<-read.csv(file =
"C:/Users/puj83/OneDrive/CV/Cases/InsuranceX/exercise_04_test.csv", header =
T, sep = ",")

names(SF_Train)

##   [1] "x0"  "x1"  "x2"  "x3"  "x4"  "x5"  "x6"  "x7"  "x8"  "x9"  "x10"
##  [12] "x11" "x12" "x13" "x14" "x15" "x16" "x17" "x18" "x19" "x20" "x21"
##  [23] "x22" "x23" "x24" "x25" "x26" "x27" "x28" "x29" "x30" "x31" "x32"
##  [34] "x33" "x34" "x35" "x36" "x37" "x38" "x39" "x40" "x41" "x42" "x43"
##  [45] "x44" "x45" "x46" "x47" "x48" "x49" "x50" "x51" "x52" "x53" "x54"
##  [56] "x55" "x56" "x57" "x58" "x59" "x60" "x61" "x62" "x63" "x64" "x65"
##  [67] "x66" "x67" "x68" "x69" "x70" "x71" "x72" "x73" "x74" "x75" "x76"
##  [78] "x77" "x78" "x79" "x80" "x81" "x82" "x83" "x84" "x85" "x86" "x87"
##  [89] "x88" "x89" "x90" "x91" "x92" "x93" "x94" "x95" "x96" "x97" "x98"
## [100] "x99" "y"

names(SF_Test)

##   [1] "x0"  "x1"  "x2"  "x3"  "x4"  "x5"  "x6"  "x7"  "x8"  "x9"  "x10"
##  [12] "x11" "x12" "x13" "x14" "x15" "x16" "x17" "x18" "x19" "x20" "x21"
##  [23] "x22" "x23" "x24" "x25" "x26" "x27" "x28" "x29" "x30" "x31" "x32"
##  [34] "x33" "x34" "x35" "x36" "x37" "x38" "x39" "x40" "x41" "x42" "x43"
##  [45] "x44" "x45" "x46" "x47" "x48" "x49" "x50" "x51" "x52" "x53" "x54"
##  [56] "x55" "x56" "x57" "x58" "x59" "x60" "x61" "x62" "x63" "x64" "x65"
##  [67] "x66" "x67" "x68" "x69" "x70" "x71" "x72" "x73" "x74" "x75" "x76"
##  [78] "x77" "x78" "x79" "x80" "x81" "x82" "x83" "x84" "x85" "x86" "x87"
##  [89] "x88" "x89" "x90" "x91" "x92" "x93" "x94" "x95" "x96" "x97" "x98"
## [100] "x99"

# str(SF_Train)

# Below are factors in SF_Train

# $ x34: Factor w/ 11 levels "","bmw","chevrolet",..: 2 10 2 10 10 10 2 9 9 4
...
# $ x35: Factor w/ 9 levels "","fri","friday",..: 5 9 6 8 9 8 9 6 8 9 ...
# $ x41: Factor w/ 37814 levels "","$0.03 ","$0.09 ",..: 21448 27346 24405
28719 1817 22615 22255 4083 1533 28524 ...
# $ x45: Factor w/ 10 levels "","-0.01%","-0.02%",..: 2 6 6 7 2 6 7 7 6 6 ...
# $ x68: Factor w/ 13 levels "","Apr","Aug",..: 13 7 7 2 3 3 7 7 3 10 ...
# $ x93: Factor w/ 4 levels "","america","asia",..: 3 3 3 3 3 3 2 3 3 3 ...

# str(SF_Test)

# Below are factors in SF_Test

# $ x34: Factor w/ 11 levels "","bmw","chevrolet",..: 2 10 2 10 10 10 2 9 9 4
...
```

```
# $ x35: Factor w/ 9 levels "","fri","friday",..: 5 9 6 8 9 8 9 6 8 9 ...
# $ x41: Factor w/ 37814 levels "","$0.03 ","$0.09 ",..: 21448 27346 24405
28719 1817 22615 22255 4083 1533 28524 ...
# $ x45: Factor w/ 10 levels "","-0.01%","-0.02%",..: 2 6 6 7 2 6 7 7 6 6 ...
# $ x68: Factor w/ 13 levels "","Apr","Aug",..: 13 7 7 2 3 3 7 7 3 10 ...
# $ x93: Factor w/ 4 levels "","america","asia",..: 3 3 3 3 3 3 2 3 3 3 ...

# Only Numerics

SF_Train_Correlation_Var <- subset(SF_Train, select = -c(x34, x35, x41, x45,
x68, x93, y))
SF_Train_Correlation_Var_Copy<-SF_Train_Correlation_Var
SF_Train_Correlation_Var1<- subset(SF_Train, select = -c(x34, x35, x41, x45,
x68, x93))

SF_Validation <- subset(SF_Test, select = -c(x34, x35, x41, x45, x68, x93))
SF_Validation_Copy<-SF_Validation

set.seed(123)

#############################################################################
###########################################
# Basic Feature Engineering
#############################################################################
###########################################

#########################
# Training Set
#########################

# Checking missing values

sum(is.na(SF_Train_Correlation_Var))/prod(dim(SF_Train_Correlation_Var))

## [1] 0.0002005319

SF_Train_Correlation_Var %>% summarize_all(funs(sum(is.na(.)) / length(.)))

## Warning: funs() is soft deprecated as of dplyr 0.8.0
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once per session.
```

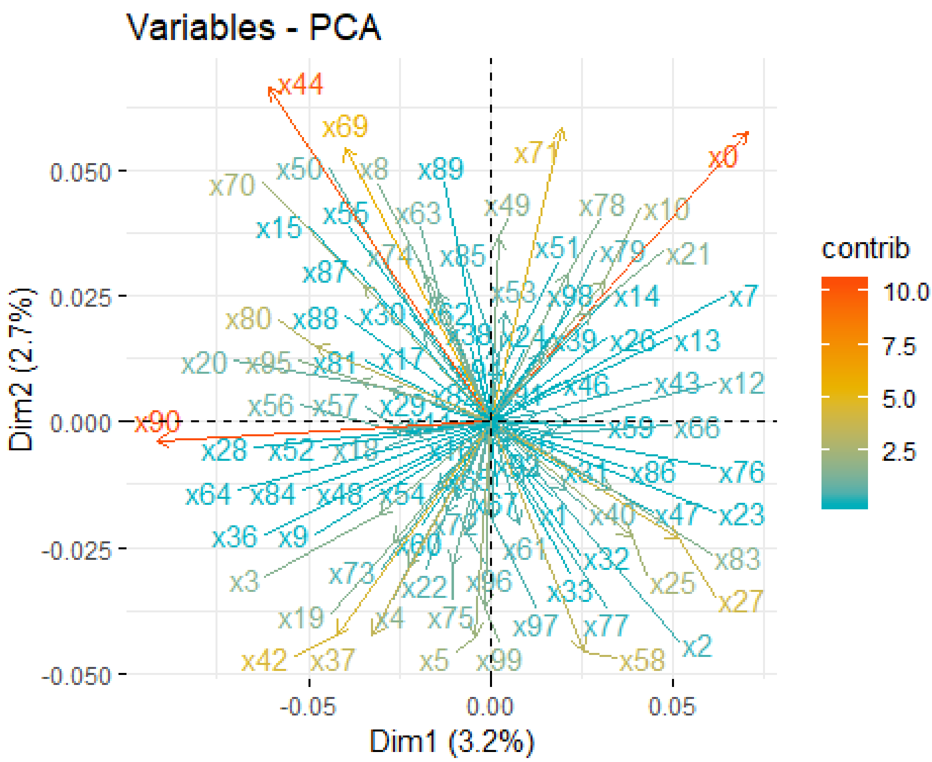
```
##          x0       x1       x2       x3    x4       x5       x6    x7    x8
## 1 0.000275 0.000275 0.000175 0.000225 2e-04 0.000275 0.000175 3e-04 1e-04
##      x9     x10     x11   x12      x13     x14     x15      x16     x17
## 1 2e-04 0.00025 0.00015 3e-04 0.000275 7.5e-05 0.00015 0.000175 0.00025
##         x18      x19      x20      x21      x22     x23   x24   x25
## 1 0.000325 0.000175 0.000125 0.000375 0.000175 0.00015 3e-04 2e-04
##         x26   x27      x28      x29      x30      x31     x32      x33
## 1 0.000225 1e-04 0.000225 0.000125 0.000125 0.000175 0.00015 0.000225
##        x36      x37      x38   x39   x40      x42   x43   x44     x46
## 1 0.00015 0.000125 0.000125 2e-04 2e-04 0.000325 5e-05 1e-04 0.00025
##         x47   x48     x49      x50      x51     x52      x53     x54   x55
## 1 0.000125 2e-04 7.5e-05 0.000175 0.000275 0.00025 0.000125 0.00015 4e-04
##         x56      x57   x58   x59      x60     x61   x62      x63     x64
## 1 0.000275 0.000175 2e-04 2e-04 0.000275 0.00015 3e-04 0.000325 0.00015
##     x65      x66     x67      x69   x70      x71     x72   x73      x74
## 1 3e-04 0.000225 0.00015 0.000275 1e-04 0.000125 0.00025 2e-04 0.000175
##        x75      x76      x77      x78      x79      x80   x81      x82
## 1 0.00025 0.000275 0.000175 0.000225 0.000175 0.000175 1e-04 0.000175
##         x83     x84   x85      x86      x87   x88      x89      x90
## 1 0.000125 7.5e-05 3e-04 0.000225 0.000175 1e-04 0.000275 0.000125
##         x91   x92     x94   x95      x96      x97      x98     x99
## 1 0.000125 2e-04 0.00025 2e-04 0.000375 0.000225 0.000125 0.00025

# Mean imputation

SF_Train_Correlation_Var[] <- lapply(SF_Train_Correlation_Var, function(x) {
  x[is.na(x)] <- mean(x, na.rm = TRUE)
  x
})

# Standardizing/normalizing data

SF_Train_Correlation_Var<-apply(SF_Train_Correlation_Var, MARGIN = 2, FUN =
function(X) (X - min(X))/diff(range(X)))
SF_Train_Correlation_Var<-as.data.frame(SF_Train_Correlation_Var)

#########################
# Validation Set
#########################

# Checking missing values

sum(is.na(SF_Validation))/prod(dim(SF_Validation))

## [1] 0.0002042553

SF_Validation %>% summarize_all(funs(sum(is.na(.)) / length(.)))

##      x0    x1    x2    x3 x4 x5    x6    x7    x8    x9   x10   x11 x12
## 1 3e-04 1e-04 2e-04 4e-04  0  0 4e-04 1e-04 3e-04 1e-04 1e-04 3e-04   0
```

```
##      x13   x14   x15   x16   x17   x18   x19   x20 x21 x22   x23   x24
## 1 6e-04 2e-04 3e-04 2e-04 3e-04 2e-04 2e-04 2e-04   0   0 3e-04 4e-04
##      x25   x26   x27   x28   x29 x30   x31   x32   x33   x36   x37   x38
## 1 3e-04 2e-04 5e-04 3e-04 1e-04   0 3e-04 1e-04 3e-04 3e-04 1e-04 1e-04
##      x39 x40   x42 x43   x44   x46   x47   x48   x49 x50   x51   x52   x53
## 1 2e-04   0 2e-04   0 1e-04 1e-04 1e-04 6e-04 3e-04   0 2e-04 1e-04 1e-04
##    x54   x55   x56   x57   x58   x59   x60   x61   x62   x63 x64   x65
## 1   0 1e-04 1e-04 3e-04 2e-04 3e-04 3e-04 3e-04 4e-04 2e-04   0 2e-04
##      x66   x67   x69   x70   x71 x72   x73   x74   x75 x76   x77   x78
## 1 2e-04 4e-04 3e-04 2e-04 1e-04   0 5e-04 3e-04 2e-04   0 5e-04 1e-04
##      x79   x80   x81   x82   x83 x84   x85   x86   x87   x88   x89   x90
## 1 3e-04 2e-04 2e-04 3e-04 1e-04   0 3e-04 3e-04 2e-04 2e-04 2e-04 3e-04
##    x91 x92   x94   x95   x96   x97   x98   x99
## 1   0   0 2e-04 1e-04 2e-04 4e-04 2e-04 5e-04

# Mean imputation

SF_Validation[] <- lapply(SF_Validation, function(x) {
  x[is.na(x)] <- mean(x, na.rm = TRUE)
  x
})

# Standardizing/normalizing data

SF_Validation<-apply(SF_Validation, MARGIN = 2, FUN = function(X) (X -
min(X))/diff(range(X)))
SF_Validation<-as.data.frame(SF_Validation)

#############################################################################
###########################################
# PCA Analysis / Exploratory Data Analysis (Feature Extraction)
#############################################################################
###########################################

# PCA on Training Data to determine potential predictors to test on
validation set

# Arguments for princomp():
# x: a numeric matrix or data frame
# cor: a logical value. If TRUE, the data will be centered and scaled before
the analysis
# scores: a logical value. If TRUE, the coordinates on each principal
component are calculated

res.pca<-princomp(SF_Train_Correlation_Var, cor = FALSE, scores = TRUE)

#Visualize eigenvalues (scree plot). Show the percentage of variances
explained by each principal component.
```

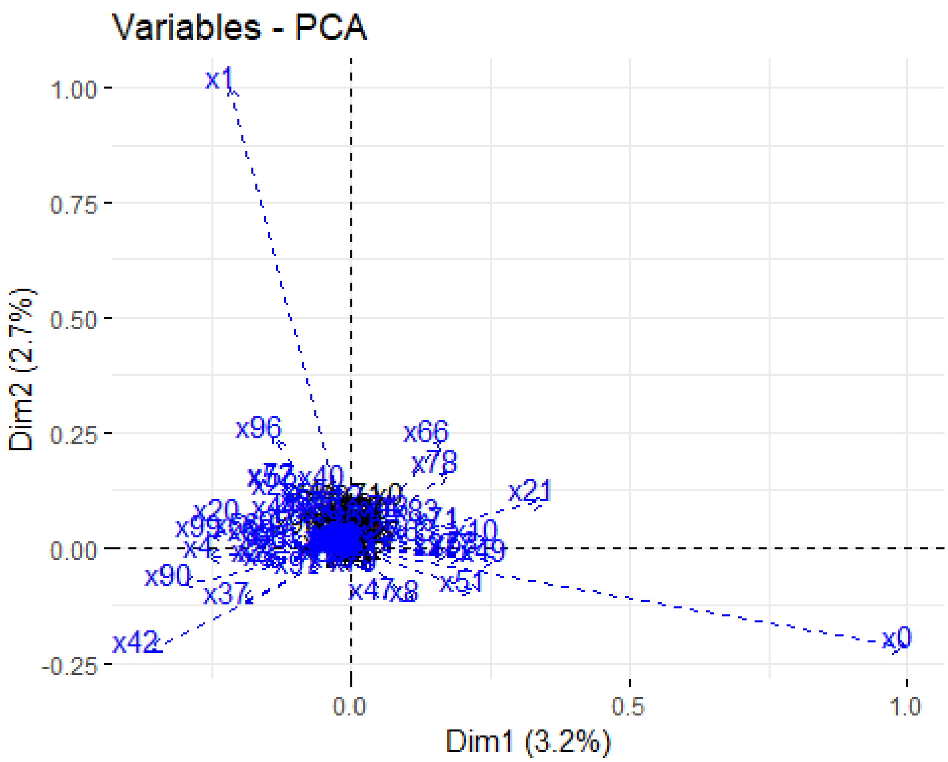
```r
fviz_eig(res.pca)
```




```r
# Graph of variables.
# Positive correlated variables point to the same side of the plot.
# Negative correlated variables point to opposite sides of the graph.

fviz_pca_var(res.pca,
             col.var = "contrib", # Color by contributions to the PC
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE     # Avoid text overlapping
)
```




```r
# PCA Results

eig.val <- get_eigenvalue(res.pca)

# Results for Variables
res.var <- get_pca_var(res.pca)
# res.var$coord          # Coordinates
# res.var$contrib        # Contributions to the PCs
# res.var$cos2           # Quality of representation

quanti.sup <- SF_Train_Correlation_Var
# head(quanti.sup)

# Predict coordinates and compute cos2
quanti.coord <- cor(quanti.sup, res.pca$x)
quanti.cos2 <- quanti.coord^2
# Graph of variables including supplementary variables
p <- fviz_pca_var(res.pca)
fviz_add(p, quanti.coord, color ="blue", geom="arrow")
```

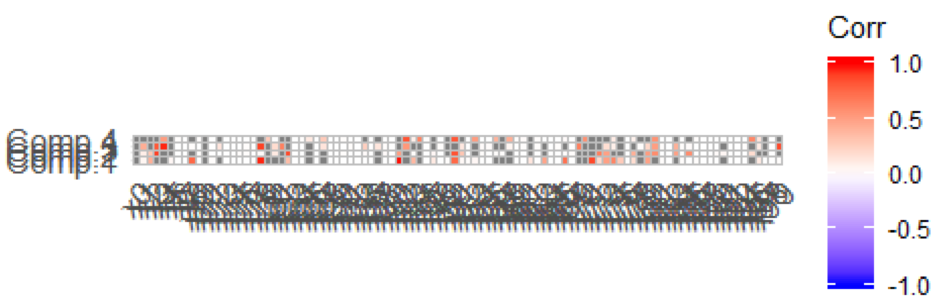



```
# Here we'll show how to calculate the PCA results for variables:
coordinates, cos2 and contributions:

# var.coord = loadings * the component standard deviations
# var.cos2 = var.coord^2
# var.contrib. The contribution of a variable to a given principal component
is (in percentage) : (var.cos2 * 100) / (total cos2 of the component)

var_coord_func <- function(loadings, comp.sdev){
  loadings*comp.sdev
}
# Compute Coordinates
#::::::::::::::::::::::::::::::::::::::::
loadings <- res.pca$loadings
sdev <- res.pca$sdev
var.coord <- t(apply(loadings, 1, var_coord_func, sdev))
head(var.coord[, 1:4])

##           Comp.1      Comp.2      Comp.3      Comp.4
## x0  0.070306870  0.05756889  0.01891283  0.01870020
## x1  0.004697459 -0.00569136 -0.01225523  0.02875406
## x2  0.013299706 -0.01057674 -0.03422464 -0.02174533
## x3 -0.030518395 -0.01833448  0.01697233 -0.01968608
## x4 -0.022797489 -0.02874476  0.01846526  0.01283098
## x5 -0.004354739 -0.04265499 -0.06070574  0.04409827
```

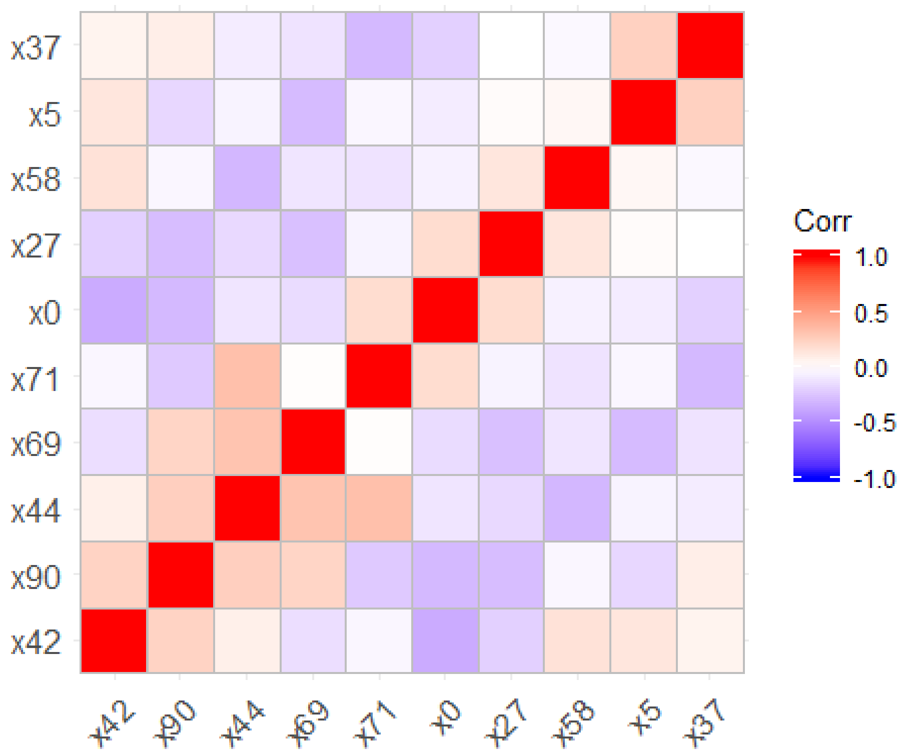
```r
# Compute Cos2
#::::::::::::::::::::::::::::::::::::::::
var.cos2 <- var.coord^2
head(var.cos2[, 1:4])
```

```
##          Comp.1       Comp.2       Comp.3       Comp.4
## x0 4.943056e-03 3.314177e-03 0.0003576952 0.0003496975
## x1 2.206612e-05 3.239157e-05 0.0001501907 0.0008267957
## x2 1.768822e-04 1.118675e-04 0.0011713262 0.0004728594
## x3 9.313724e-04 3.361530e-04 0.0002880600 0.0003875416
## x4 5.197255e-04 8.262611e-04 0.0003409657 0.0001646342
## x5 1.896375e-05 1.819448e-03 0.0036851870 0.0019446578
```

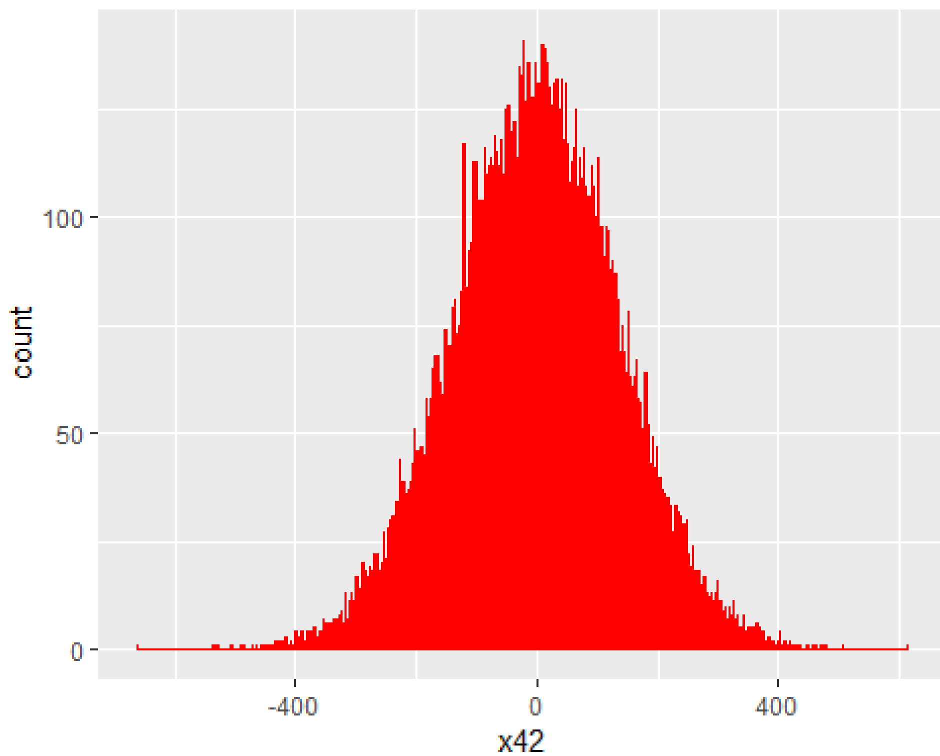
```r
# Compute contributions
#::::::::::::::::::::::::::::::::::::::::
comp.cos2 <- apply(var.cos2, 2, sum)
contrib <- function(var.cos2, comp.cos2){var.cos2*100/comp.cos2}
var.contrib <- t(apply(var.cos2,1, contrib, comp.cos2))
Contributions<-var.contrib[, 1:4]
ggcorrplot(Contributions)
```



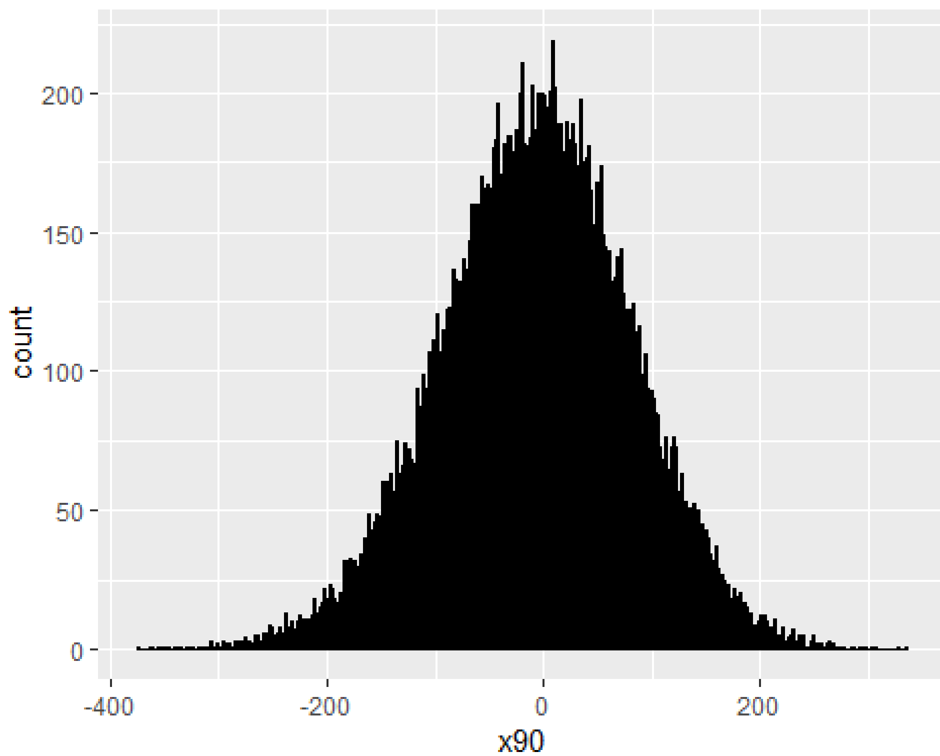

```r
SF_Train_Correlation_Vars_Corr <- subset(SF_Train_Correlation_Var, select =
c(x42, x90, x44, x69, x71, x0, x27, x58, x5, x37))
corr2<- round(cor(SF_Train_Correlation_Vars_Corr), 2)
ggcorrplot(corr2)
```



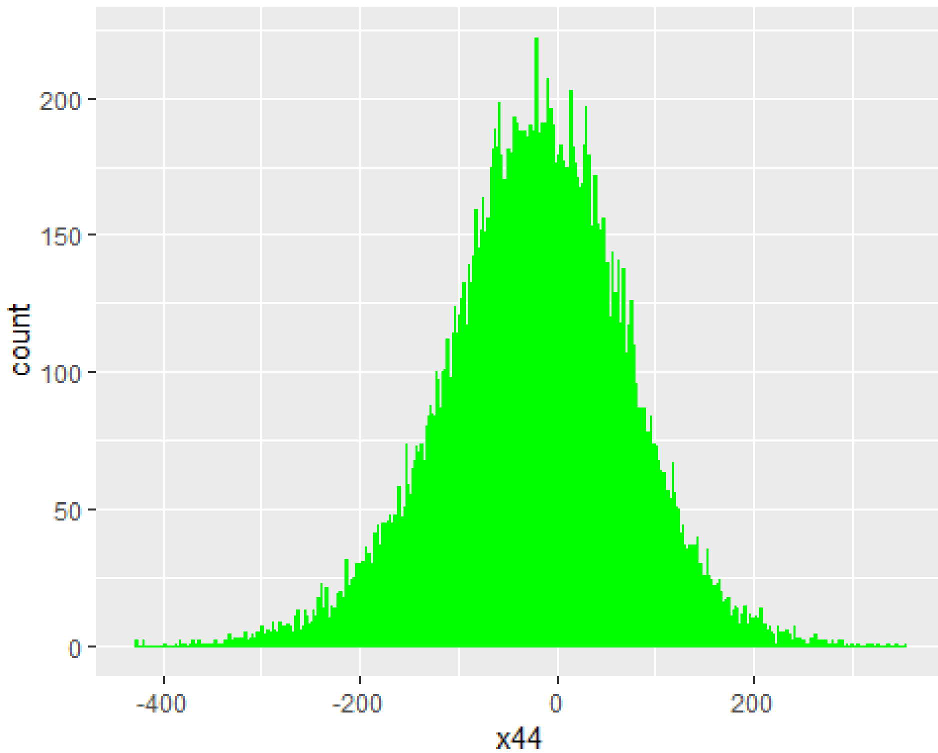

```
## set the seed to make your partition reproducible
set.seed(123)

# Histograms of variables

ggplot(SF_Train_Correlation_Var1, aes(x=x42)) +
  geom_histogram(binwidth=1, colour="red", fill="red")

## Warning: Removed 13 rows containing non-finite values (stat_bin).
```



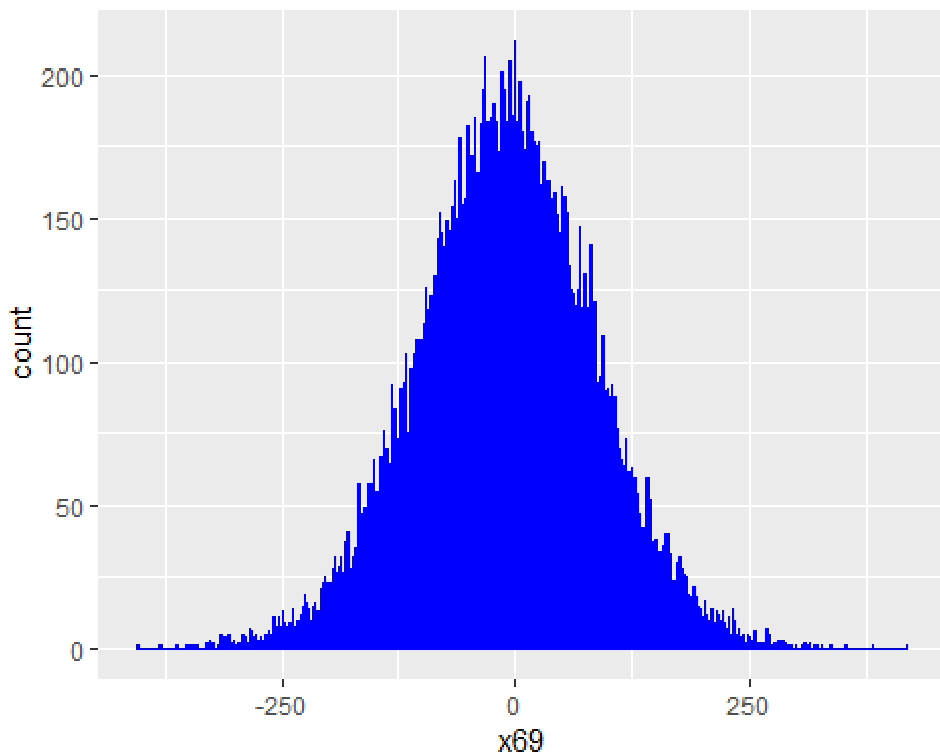

```
ggplot(SF_Train_Correlation_Var1, aes(x=x90)) +
  geom_histogram(binwidth=1, colour="black", fill="black")

## Warning: Removed 5 rows containing non-finite values (stat_bin).
```



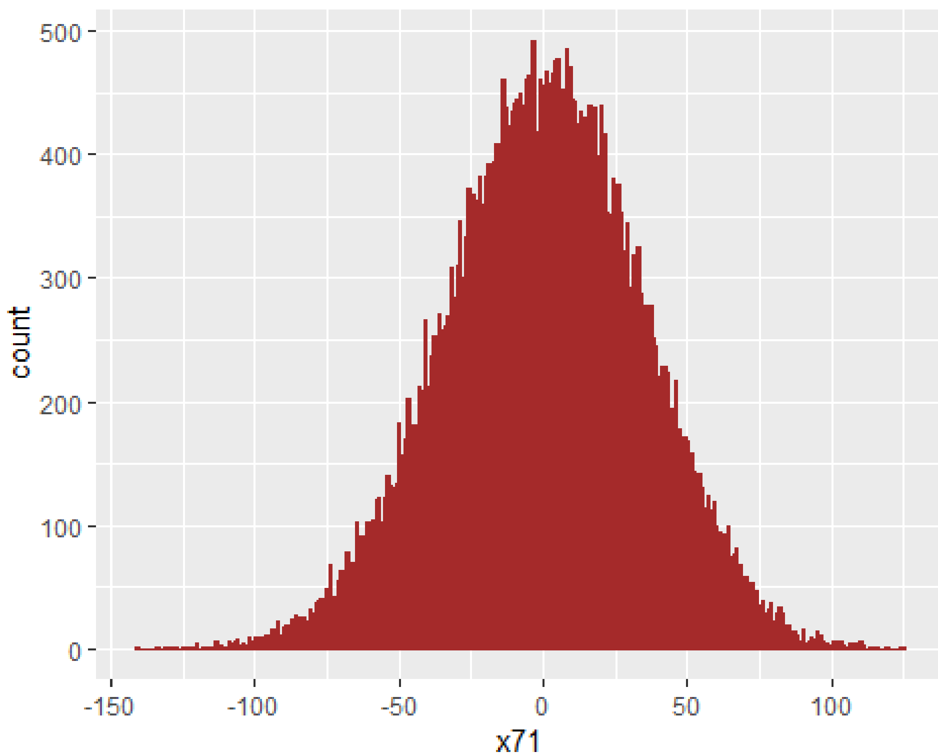

```
ggplot(SF_Train_Correlation_Var1, aes(x=x44)) +
  geom_histogram(binwidth=1, colour="green", fill="green")

## Warning: Removed 4 rows containing non-finite values (stat_bin).
```



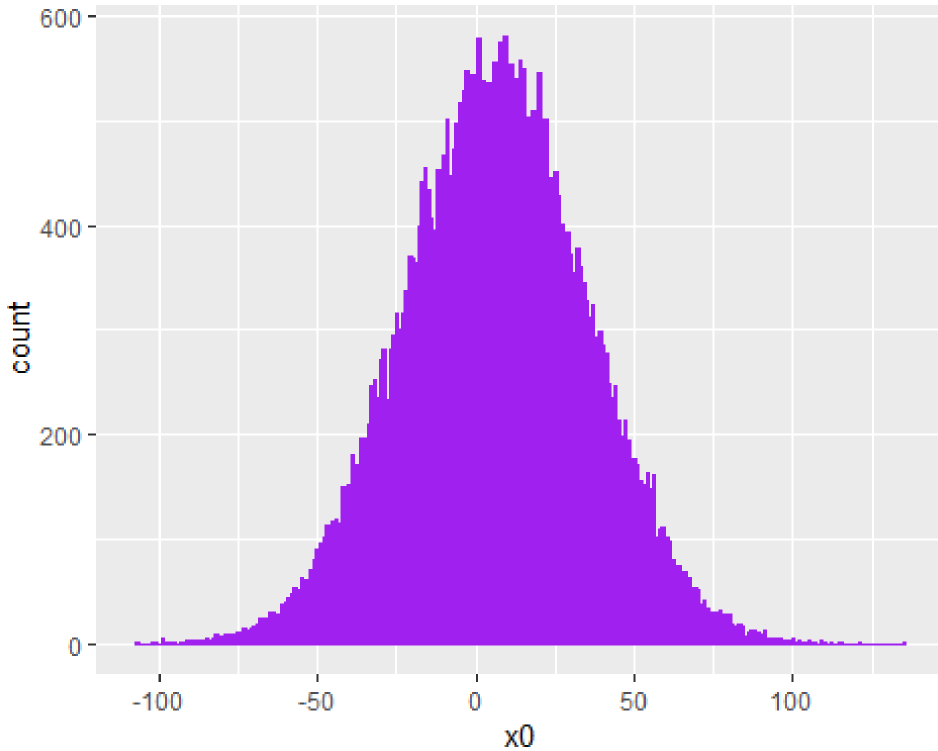

```
ggplot(SF_Train_Correlation_Var1, aes(x=x69)) +
  geom_histogram(binwidth=1, colour="blue", fill="blue")

## Warning: Removed 11 rows containing non-finite values (stat_bin).
```




```
ggplot(SF_Train_Correlation_Var1, aes(x=x71)) +
  geom_histogram(binwidth=1, colour="brown", fill="brown")

## Warning: Removed 5 rows containing non-finite values (stat_bin).
```




```r
ggplot(SF_Train_Correlation_Var1, aes(x=x0)) +
  geom_histogram(binwidth=1, colour="purple", fill="purple")

## Warning: Removed 11 rows containing non-finite values (stat_bin).
```




```
################################################################################
############################################
# Applying Machine Learning Algorithms via Basic Feature Engineering (Feature
Selection)
################################################################################
############################################

# Algorithm #1: Random Forest

SF_Train_Correlation_Var1$y<-
as.numeric(as.character(SF_Train_Correlation_Var1$y))
SF_Train_Correlation_Var1_y<-as.data.frame(SF_Train_Correlation_Var1$y)
SF_Train_Correlation_Var<-cbind(SF_Train_Correlation_Var,
SF_Train_Correlation_Var1_y)

names(SF_Train_Correlation_Var)[names(SF_Train_Correlation_Var) ==
"SF_Train_Correlation_Var1$y"] <- "y"
SF_Train_Correlation_Var$y<-as.factor(SF_Train_Correlation_Var$y)

## 75% of the sample size
smp_size <- floor(0.80 * nrow(SF_Train_Correlation_Var))

## set the seed to make your partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(SF_Train_Correlation_Var)), size = smp_size)
```

```
train <- SF_Train_Correlation_Var[train_ind, ]
test <- SF_Train_Correlation_Var[-train_ind, ]

# for reproducibility

set.seed(123)

#train

rf1<-randomForest(y~ x90 + x44 + x42 + x69 + x71 +x0 + x27 + x58+ x5 + x37,
                  data = train, ntree = 500,
                  mtry = 4, importance = TRUE, na.action = na.omit)

print(rf1)

##
## Call:
##  randomForest(formula = y ~ x90 + x44 + x42 + x69 + x71 + x0 +      x27 +
x58 + x5 + x37, data = train, ntree = 500, mtry = 4,      importance = TRUE,
na.action = na.omit)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 17.25%
## Confusion matrix:
##       0    1 class.error
## 0 24733  779  0.03053465
## 1  4742 1746  0.73088779

#test

rf2<-randomForest(y~ x90 + x44 + x42 + x69 + x71 +x0 + x27 + x58+ x5 + x37,
                  data = test, ntree = 500,
                  mtry = 4, importance = TRUE, na.action = na.omit)

print(rf2)

##
## Call:
##  randomForest(formula = y ~ x90 + x44 + x42 + x69 + x71 + x0 +      x27 +
x58 + x5 + x37, data = test, ntree = 500, mtry = 4,      importance = TRUE,
na.action = na.omit)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 18.57%
## Confusion matrix:
##      0   1 class.error
```

```
## 0 6193 175  0.02748116
## 1 1311 321  0.80330882

test$pred_randomForest<-predict(rf1, test)
test_rf_comparison<-test %>% select(y, pred_randomForest)

#Validation

SF_Validation$pred_randomForest<-predict(rf1, SF_Validation)

# Algorithm #2: Adaboost

train_var <- subset(train, select = c(x90, x44, x42, x69, x71, x0, x27, x58,
x5, x37))
ind_Attr1<-names(train_var)

test_var <- subset(test, select = c(x90, x44, x42, x69, x71, x0, x27, x58,
x5, x37))
ind_Attr2<-names(test_var)

# Build best ada boost model
ada1<-ada(x = train[,ind_Attr1],
            y = train$y,
            iter=20, loss="logistic",verbose=TRUE) # 20 Iterations

## FINAL: iter= 20  rate= 5.321952e-10
## FINAL: iter= 20  rate= 5.754619e-10
## FINAL: iter= 20  rate= 6.228893e-10
## FINAL: iter= 20  rate= 6.457764e-10
## FINAL: iter= 20  rate= 5.776881e-10
## FINAL: iter= 20  rate= 7.230121e-10
## FINAL: iter= 20  rate= 6.026166e-10
## FINAL: iter= 20  rate= 6.882537e-10
## FINAL: iter= 20  rate= 7.523619e-10
## FINAL: iter= 20  rate= 4.596955e-10
## FINAL: iter= 20  rate= 6.597402e-10
## FINAL: iter= 20  rate= 7.288775e-10
## FINAL: iter= 20  rate= 6.896165e-10
## FINAL: iter= 20  rate= 6.447544e-10
## FINAL: iter= 20  rate= 7.564216e-10
## FINAL: iter= 20  rate= 8.116626e-10
## FINAL: iter= 20  rate= 5.996096e-10
## FINAL: iter= 20  rate= 2.706232e-10
## FINAL: iter= 20  rate= 3.824858e-10
## FINAL: iter= 20  rate= 5.678407e-10

# Look at the model summary
summary(ada1)
```

```
## Call:
## ada(train[, ind_Attr1], y = train$y, loss = "logistic", iter = 20,
##     verbose = TRUE)
##
## Loss: logistic Method: discrete   Iteration: 20
##
## Training Results
##
## Accuracy: 0.806 Kappa: 0.096
```

```r
# Build best ada boost model
ada2<-ada(x = test[,ind_Attr2],
          y = test$y,
          iter=20, loss="logistic",verbose=TRUE) # 20 Iterations
```

```
## FINAL: iter= 20  rate= 8.784741e-09
## FINAL: iter= 20  rate= 9.343813e-09
## FINAL: iter= 20  rate= 9.585764e-09
## FINAL: iter= 20  rate= 1.011139e-08
## FINAL: iter= 20  rate= 9.547399e-09
## FINAL: iter= 20  rate= 1.036407e-08
## FINAL: iter= 20  rate= 1.221325e-08
## FINAL: iter= 20  rate= 1.139484e-08
## FINAL: iter= 20  rate= 1.18086e-08
## FINAL: iter= 20  rate= 6.414827e-09
## FINAL: iter= 20  rate= 1.212286e-08
## FINAL: iter= 20  rate= 6.872939e-09
## FINAL: iter= 20  rate= 9.732799e-09
## FINAL: iter= 20  rate= 1.025242e-08
## FINAL: iter= 20  rate= 6.871456e-09
## FINAL: iter= 20  rate= 7.980506e-09
## FINAL: iter= 20  rate= 8.629301e-09
## FINAL: iter= 20  rate= 7.707187e-09
## FINAL: iter= 20  rate= 5.370915e-09
## FINAL: iter= 20  rate= 1.272687e-08
```

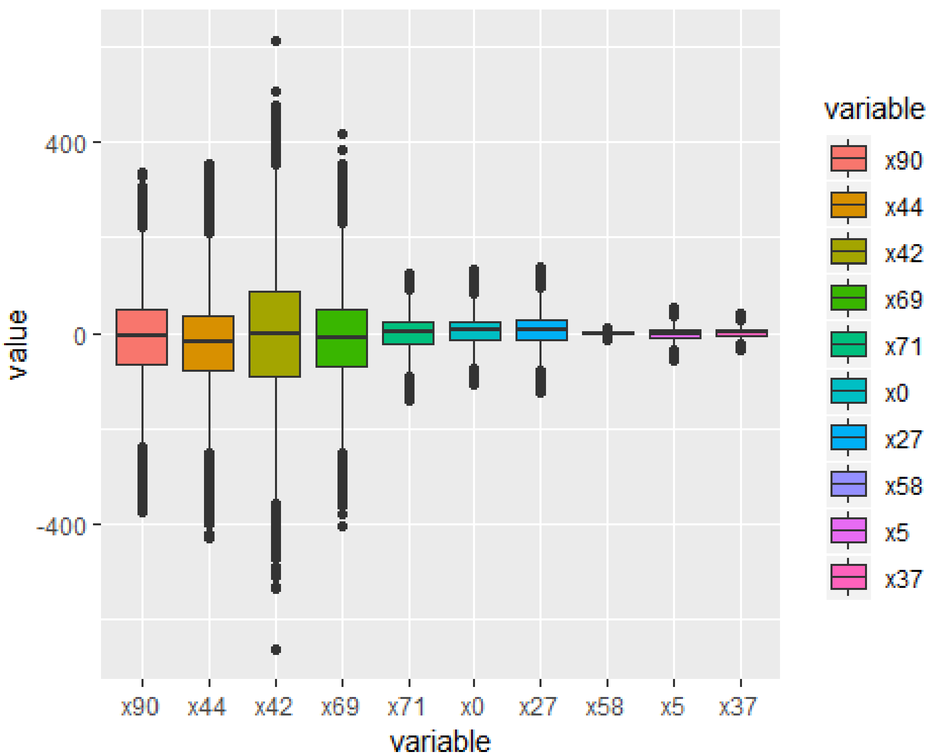
```r
# Look at the model summary
summary(ada2)
```

```
## Call:
## ada(test[, ind_Attr2], y = test$y, loss = "logistic", iter = 20,
##     verbose = TRUE)
##
## Loss: logistic Method: discrete   Iteration: 20
##
## Training Results
##
## Accuracy: 0.807 Kappa: 0.115
```

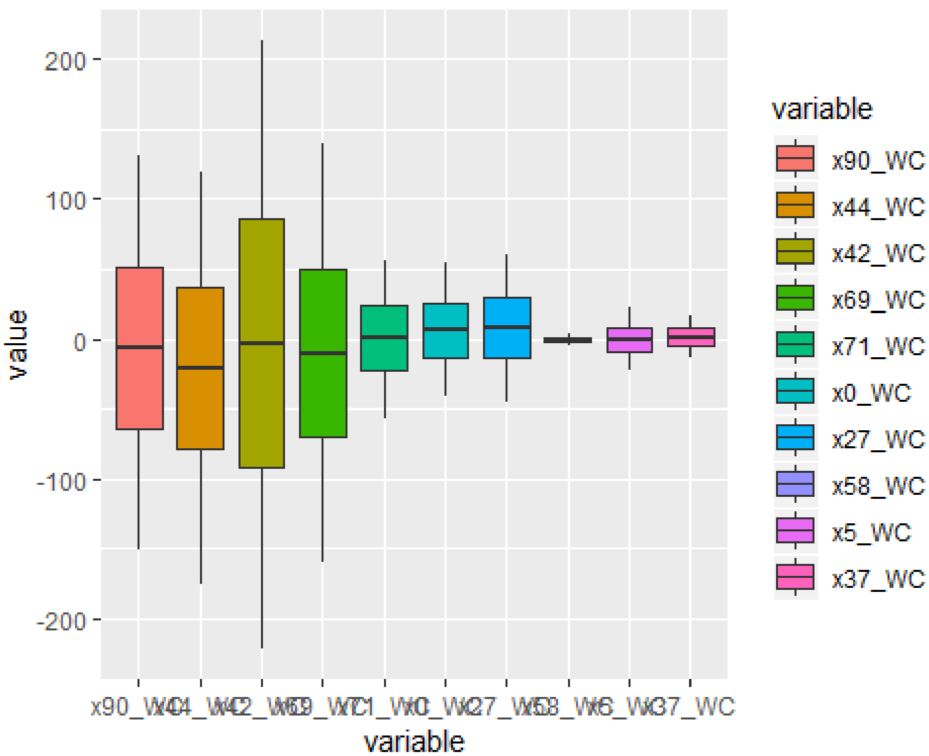
```r
# Predict on train data
pred_Train<-predict(ada1, train[,ind_Attr1])
```

```
# Build confusion matrix and find accuracy
cm_Train = table(train$y, pred_Train)
accu_Train= sum(diag(cm_Train))/sum(cm_Train)
rm(pred_Train, cm_Train)

# Predict on test data
pred_Test = predict(ada1, test[,ind_Attr2])

# Build confusion matrix and find accuracy
cm_Test = table(test$y, pred_Test)
accu_Test= sum(diag(cm_Test))/sum(cm_Test)
rm(pred_Test, cm_Test)

#Validation

SF_Validation$pred_ada<-predict(ada1, SF_Validation)

#############################################################################
###########################################
# More Feature Engineering to Improve Model
#############################################################################
###########################################

#########################
# Training Set
#########################

# Checking missing values

sum(is.na(SF_Train_Correlation_Var_Copy))/prod(dim(SF_Train_Correlation_Var_C
opy))

## [1] 0.0002005319

SF_Train_Correlation_Var_Copy %>% summarize_all(funs(sum(is.na(.)) /
length(.)))

##         x0       x1       x2       x3    x4       x5       x6    x7    x8
## 1 0.000275 0.000275 0.000175 0.000225 2e-04 0.000275 0.000175 3e-04 1e-04
##      x9     x10     x11   x12      x13     x14     x15      x16     x17
## 1 2e-04 0.00025 0.00015 3e-04 0.000275 7.5e-05 0.00015 0.000175 0.00025
##        x18      x19      x20      x21      x22     x23   x24   x25
## 1 0.000325 0.000175 0.000125 0.000375 0.000175 0.00015 3e-04 2e-04
##        x26   x27      x28      x29      x30      x31     x32      x33
## 1 0.000225 1e-04 0.000225 0.000125 0.000125 0.000175 0.00015 0.000225
##       x36      x37      x38   x39   x40      x42   x43   x44     x46
## 1 0.00015 0.000125 0.000125 2e-04 2e-04 0.000325 5e-05 1e-04 0.00025
##        x47   x48     x49      x50      x51     x52      x53     x54   x55
## 1 0.000125 2e-04 7.5e-05 0.000175 0.000275 0.00025 0.000125 0.00015 4e-04
```

```
##        x56      x57   x58   x59      x60     x61   x62      x63     x64
## 1 0.000275 0.000175 2e-04 2e-04 0.000275 0.00015 3e-04 0.000325 0.00015
##     x65      x66     x67      x69   x70      x71     x72   x73      x74
## 1 3e-04 0.000225 0.00015 0.000275 1e-04 0.000125 0.00025 2e-04 0.000175
##       x75      x76      x77      x78      x79      x80   x81      x82
## 1 0.00025 0.000275 0.000175 0.000225 0.000175 0.000175 1e-04 0.000175
##        x83     x84   x85      x86      x87   x88      x89      x90
## 1 0.000125 7.5e-05 3e-04 0.000225 0.000175 1e-04 0.000275 0.000125
##        x91   x92     x94   x95      x96      x97      x98     x99
## 1 0.000125 2e-04 0.00025 2e-04 0.000375 0.000225 0.000125 0.00025

# Mean imputation

SF_Train_Correlation_Var_Copy[] <- lapply(SF_Train_Correlation_Var_Copy,
function(x) {
  x[is.na(x)] <- mean(x, na.rm = TRUE)
  x
})

# Treating outliers by Winsorizing/Capping

# Winsorizing

fun <- function(x){
  quantiles <- quantile( x, c(.05, .95 ) )
  x[ x < quantiles[1] ] <- quantiles[1]
  x[ x > quantiles[2] ] <- quantiles[2]
  x
}

SF_Train_Correlation_Var_Copy_BP <- subset(SF_Train_Correlation_Var_Copy,
select = c(x90, x44, x42, x69, x71, x0, x27, x58, x5, x37))
ggplot(data = melt(SF_Train_Correlation_Var_Copy_BP), aes(x=variable,
y=value)) + geom_boxplot(aes(fill=variable))

## No id variables; using all as measure variables
```



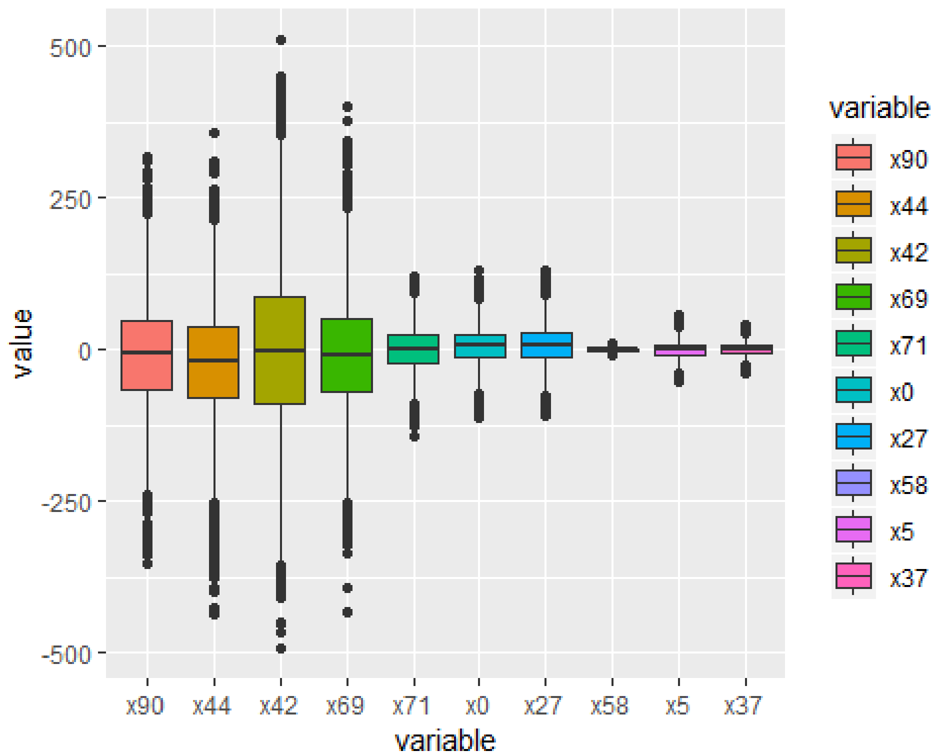

```
SF_Train_Correlation_Var_Copy$x90_WC<-fun(SF_Train_Correlation_Var_Copy$x90)
SF_Train_Correlation_Var_Copy$x44_WC<-fun(SF_Train_Correlation_Var_Copy$x44)
SF_Train_Correlation_Var_Copy$x42_WC<-fun(SF_Train_Correlation_Var_Copy$x42)
SF_Train_Correlation_Var_Copy$x69_WC<-fun(SF_Train_Correlation_Var_Copy$x69)
SF_Train_Correlation_Var_Copy$x71_WC<-fun(SF_Train_Correlation_Var_Copy$x71)
SF_Train_Correlation_Var_Copy$x0_WC<-fun(SF_Train_Correlation_Var_Copy$x0)
SF_Train_Correlation_Var_Copy$x27_WC<-fun(SF_Train_Correlation_Var_Copy$x27)
SF_Train_Correlation_Var_Copy$x58_WC<-fun(SF_Train_Correlation_Var_Copy$x58)
SF_Train_Correlation_Var_Copy$x5_WC<-fun(SF_Train_Correlation_Var_Copy$x5)
SF_Train_Correlation_Var_Copy$x37_WC<-fun(SF_Train_Correlation_Var_Copy$x37)

SF_Train_Correlation_Var_Copy_BP2 <- subset(SF_Train_Correlation_Var_Copy,
select = c(x90_WC, x44_WC, x42_WC, x69_WC, x71_WC, x0_WC, x27_WC, x58_WC,
x5_WC, x37_WC))
ggplot(data = melt(SF_Train_Correlation_Var_Copy_BP2), aes(x=variable,
y=value)) + geom_boxplot(aes(fill=variable))

## No id variables; using all as measure variables
```




```
# Capping

SF_Train_Correlation_Var_Copy$x42_WC[SF_Train_Correlation_Var_Copy$x42_WC >
200] = 200
SF_Train_Correlation_Var_Copy$x42_WC[SF_Train_Correlation_Var_Copy$x42_WC < -
200] = 200

max(SF_Train_Correlation_Var_Copy$x42_WC)

## [1] 200

min(SF_Train_Correlation_Var_Copy$x42_WC)

## [1] -199.9812

# Standardizing/normalizing data

SF_Train_Correlation_Var_Copy<-apply(SF_Train_Correlation_Var_Copy, MARGIN =
2, FUN = function(X) (X - min(X))/diff(range(X)))
SF_Train_Correlation_Var_Copy<-as.data.frame(SF_Train_Correlation_Var_Copy)

# Bucketing/binning/categorization

SF_Train_Correlation_Var_Copy$x90_WC <-
ifelse(SF_Train_Correlation_Var_Copy$x90_WC > 0.5, 1, 0)
SF_Train_Correlation_Var_Copy$x44_WC <-
ifelse(SF_Train_Correlation_Var_Copy$x44_WC > 0.5, 1, 0)
```

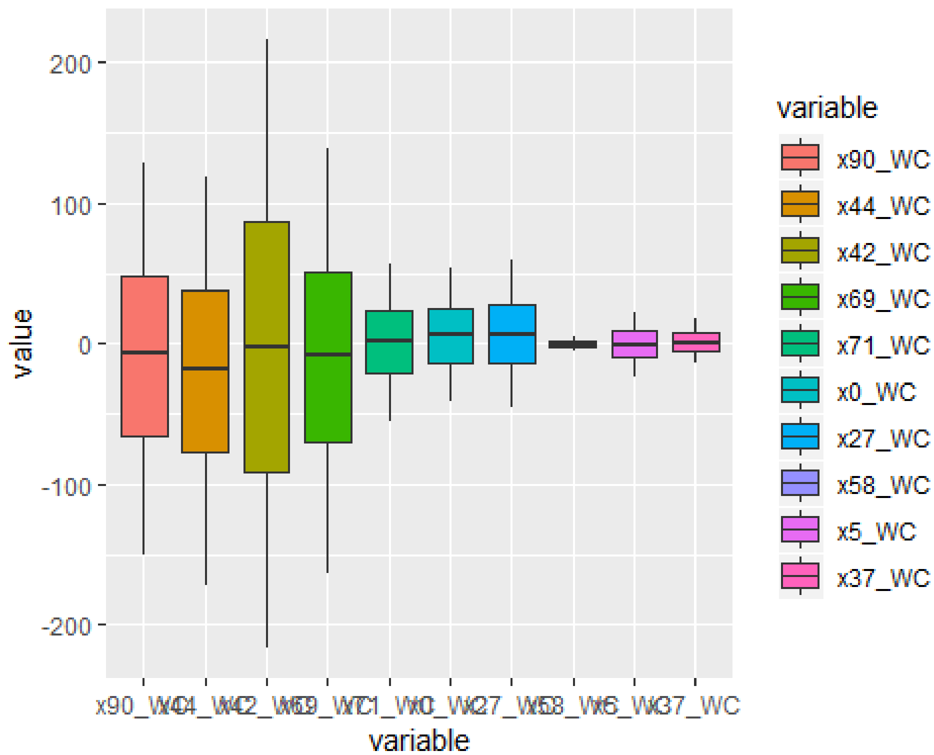
```
SF_Train_Correlation_Var_Copy$x0_WC <-
ifelse(SF_Train_Correlation_Var_Copy$x0_WC > 0.5, 1, 0)

# Interaction

SF_Train_Correlation_Var_Copy$Three_Var_mean <-
rowMeans(subset(SF_Train_Correlation_Var_Copy, select = c(x90, x44, x0)),
na.rm = TRUE)


#########################
# Validation Set
#########################

# Checking missing values

sum(is.na(SF_Validation_Copy))/prod(dim(SF_Validation_Copy))

## [1] 0.0002042553

SF_Validation_Copy %>% summarize_all(funs(sum(is.na(.)) / length(.)))

##       x0    x1    x2    x3 x4 x5    x6    x7    x8    x9   x10   x11 x12
## 1 3e-04 1e-04 2e-04 4e-04  0  0 4e-04 1e-04 3e-04 1e-04 1e-04 3e-04   0
##     x13   x14   x15   x16   x17   x18   x19   x20 x21 x22   x23   x24
## 1 6e-04 2e-04 3e-04 2e-04 3e-04 2e-04 2e-04 2e-04   0   0 3e-04 4e-04
##     x25   x26   x27   x28   x29 x30   x31   x32   x33   x36   x37   x38
## 1 3e-04 2e-04 5e-04 3e-04 1e-04   0 3e-04 1e-04 3e-04 3e-04 1e-04 1e-04
##     x39 x40   x42 x43   x44   x46   x47   x48   x49 x50   x51   x52   x53
## 1 2e-04   0 2e-04   0 1e-04 1e-04 1e-04 6e-04 3e-04   0 2e-04 1e-04 1e-04
##   x54   x55   x56   x57   x58   x59   x60   x61   x62   x63 x64   x65
## 1   0 1e-04 1e-04 3e-04 2e-04 3e-04 3e-04 3e-04 4e-04 2e-04   0 2e-04
##     x66   x67   x69   x70   x71 x72   x73   x74   x75 x76   x77   x78
## 1 2e-04 4e-04 3e-04 2e-04 1e-04   0 5e-04 3e-04 2e-04   0 5e-04 1e-04
##     x79   x80   x81   x82   x83 x84   x85   x86   x87   x88   x89   x90
## 1 3e-04 2e-04 2e-04 3e-04 1e-04   0 3e-04 3e-04 2e-04 2e-04 2e-04 3e-04
##   x91 x92   x94   x95   x96   x97   x98   x99
## 1   0   0 2e-04 1e-04 2e-04 4e-04 2e-04 5e-04

# Mean imputation

SF_Validation_Copy[] <- lapply(SF_Validation_Copy, function(x) {
  x[is.na(x)] <- mean(x, na.rm = TRUE)
  x
})

# Treating outliers by Winsorizing/Capping

# Winsorizing
```

```r
fun <- function(x){
  quantiles <- quantile( x, c(.05, .95 ) )
  x[ x < quantiles[1] ] <- quantiles[1]
  x[ x > quantiles[2] ] <- quantiles[2]
  x
}

SF_Validation_Correlation_Var_Copy_BP <- subset(SF_Validation_Copy, select =
c(x90, x44, x42, x69, x71, x0, x27, x58, x5, x37))
ggplot(data = melt(SF_Validation_Correlation_Var_Copy_BP), aes(x=variable,
y=value)) + geom_boxplot(aes(fill=variable))

## No id variables; using all as measure variables
```




```r
SF_Validation_Copy$x90_WC<-fun(SF_Validation_Copy$x90)
SF_Validation_Copy$x44_WC<-fun(SF_Validation_Copy$x44)
SF_Validation_Copy$x42_WC<-fun(SF_Validation_Copy$x42)
SF_Validation_Copy$x69_WC<-fun(SF_Validation_Copy$x69)
SF_Validation_Copy$x71_WC<-fun(SF_Validation_Copy$x71)
SF_Validation_Copy$x0_WC<-fun(SF_Validation_Copy$x0)
SF_Validation_Copy$x27_WC<-fun(SF_Validation_Copy$x27)
SF_Validation_Copy$x58_WC<-fun(SF_Validation_Copy$x58)
SF_Validation_Copy$x5_WC<-fun(SF_Validation_Copy$x5)
SF_Validation_Copy$x37_WC<-fun(SF_Validation_Copy$x37)

SF_Validation_Correlation_Var_Copy_BP2 <- subset(SF_Validation_Copy, select =
c(x90_WC, x44_WC, x42_WC, x69_WC, x71_WC, x0_WC, x27_WC, x58_WC, x5_WC,
```

```
x37_WC))
ggplot(data = melt(SF_Validation_Correlation_Var_Copy_BP2), aes(x=variable,
y=value)) + geom_boxplot(aes(fill=variable))

## No id variables; using all as measure variables
```




```
# Capping

SF_Validation_Copy$x42_WC[SF_Validation_Copy$x42_WC > 200] = 200
SF_Validation_Copy$x42_WC[SF_Validation_Copy$x42_WC < -200] = 200

max(SF_Validation_Copy$x42_WC)

## [1] 200

min(SF_Validation_Copy$x42_WC)

## [1] -199.7655

# Standardizing/normalizing data

SF_Validation_Copy<-apply(SF_Validation_Copy, MARGIN = 2, FUN = function(X)
(X - min(X))/diff(range(X)))
SF_Validation_Copy<-as.data.frame(SF_Validation_Copy)

# Bucketing/binning/categorization
```

```
SF_Validation_Copy$x90_WC <- ifelse(SF_Validation_Copy$x90_WC > 0.5, 1, 0)
SF_Validation_Copy$x44_WC <- ifelse(SF_Validation_Copy$x44_WC > 0.5, 1, 0)
SF_Validation_Copy$x0_WC <- ifelse(SF_Validation_Copy$x0_WC > 0.5, 1, 0)

# Interaction

SF_Validation_Copy$Three_Var_mean <- rowMeans(subset(SF_Validation_Copy,
select = c(x90, x44, x0)), na.rm = TRUE)

#############################################################################
###########################################
# Machine Learning Algorithms After Advanced Feature Engineering (Feature
Selection)
#############################################################################
###########################################

# Algorithm #1: Random Forest

SF_Train_Correlation_Var1$y<-
as.numeric(as.character(SF_Train_Correlation_Var1$y))
SF_Train_Correlation_Var1_y<-as.data.frame(SF_Train_Correlation_Var1$y)
SF_Train_Correlation_Var_Copy<-cbind(SF_Train_Correlation_Var_Copy,
SF_Train_Correlation_Var1_y)

names(SF_Train_Correlation_Var_Copy)[names(SF_Train_Correlation_Var_Copy) ==
"SF_Train_Correlation_Var1$y"] <- "y"
SF_Train_Correlation_Var_Copy$y<-as.factor(SF_Train_Correlation_Var_Copy$y)

## 75% of the sample size
smp_size <- floor(0.80 * nrow(SF_Train_Correlation_Var_Copy))

## set the seed to make your partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(SF_Train_Correlation_Var_Copy)), size =
smp_size)

train <- SF_Train_Correlation_Var_Copy[train_ind, ]
test <- SF_Train_Correlation_Var_Copy[-train_ind, ]

# for reproducibility

set.seed(123)

#train

rf1<-randomForest(y~ x90 + x44 + x0 + x42 + x69 + x71 +
                    x0 + x40 + x25 + x95 + x8 + x53 + x61 + x22 + x10 + x78 +
                    x21 + x74 + x20 + x63 + x75 + x57 + x56 + x19 + x18 + x49
+
```

```
                    x96 + x97 + x50 + x99 + x4 + x3 + x80 + x70 + x83 + x58 +
                    x5 + x37 + x27 + x12 + x66,
                  data = train, ntree = 500,
                  mtry = 12, importance = TRUE, na.action = na.omit)

print(rf1)

##
## Call:
##  randomForest(formula = y ~ x90 + x44 + x0 + x42 + x69 + x71 +      x0 +
x40 + x25 + x95 + x8 + x53 + x61 + x22 + x10 + x78 +      x21 + x74 + x20 +
x63 + x75 + x57 + x56 + x19 + x18 + x49 +      x96 + x97 + x50 + x99 + x4 +
x3 + x80 + x70 + x83 + x58 +      x5 + x37 + x27 + x12 + x66, data = train,
ntree = 500, mtry = 12,      importance = TRUE, na.action = na.omit)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 12
##
##         OOB estimate of  error rate: 8.69%
## Confusion matrix:
##       0    1 class.error
## 0 25431   81 0.003174976
## 1  2700 3788 0.416152898

#test

rf2<-randomForest(y~ x90 + x44 + x0 + x42 + x69 + x71 +
                    x0 + x40 + x25 + x95 + x8 + x53 + x61 + x22 + x10 + x78 +
                    x21 + x74 + x20 + x63 + x75 + x57 + x56 + x19 + x18 + x49
+
                    x96 + x97 + x50 + x99 + x4 + x3 + x80 + x70 + x83 + x58 +
                    x5 + x37 + x27 + x12 + x66,
                  data = train, ntree = 500,
                  mtry = 12, importance = TRUE, na.action = na.omit)
print(rf2)

##
## Call:
##  randomForest(formula = y ~ x90 + x44 + x0 + x42 + x69 + x71 +      x0 +
x40 + x25 + x95 + x8 + x53 + x61 + x22 + x10 + x78 +      x21 + x74 + x20 +
x63 + x75 + x57 + x56 + x19 + x18 + x49 +      x96 + x97 + x50 + x99 + x4 +
x3 + x80 + x70 + x83 + x58 +      x5 + x37 + x27 + x12 + x66, data = train,
ntree = 500, mtry = 12,      importance = TRUE, na.action = na.omit)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 12
##
##         OOB estimate of  error rate: 8.64%
## Confusion matrix:
##       0    1 class.error
```

```
## 0 25426   86 0.003370963
## 1  2680 3808 0.413070284

test$pred_randomForest<-predict(rf1, test)
test_rf_comparison<-test %>% select(x90, x44, x0, x42, x69, x71,
                                      x40, x25, x95, x8, x53, x61, x22, x10,
x78,
                                      x21, x74, x20, x63, x75, x57, x56, x19,
x18, x49,
                                      x96, x97, x50, x99, x4, x3, x80, x70,
x83, x58,
                                      x5, x37, x27, x12, x66, y,
pred_randomForest)
test_rf_comparison$Misclassified <- ifelse(test_rf_comparison$y ==
test_rf_comparison$pred_randomForest, 1, 0)

# View(test_rf_comparison)

# write.csv(test_rf_comparison,
"C:/Users/puj83/OneDrive/CV/Cases/InsuranceX/test_rf_comparison.csv")

#Validation

SF_Validation_Copy$pred_randomForest<-predict(rf1, SF_Validation_Copy)
write.csv(SF_Validation_Copy$pred_randomForest,
"C:/Users/puj83/OneDrive/CV/Cases/InsuranceX/SF_Validation_RF_Probabilities.c
sv")

# Algorithm #2: Adaboost

train_var <- subset(train, select = c(x90, x44, x0, x42, x69, x71,
                                      x40, x25, x95, x8, x53, x61, x22, x10,
x78,
                                      x21, x74, x20, x63, x75, x57, x56, x19,
x18, x49,
                                      x96, x97, x50, x99, x4, x3, x80, x70,
x83, x58,
                                      x5, x37, x27, x12, x66))
ind_Attr1<-names(train_var)

test_var <- subset(test, select = c(x90, x44, x0, x42, x69, x71,
                                    x40, x25, x95, x8, x53, x61, x22, x10,
x78,
                                    x21, x74, x20, x63, x75, x57, x56, x19,
x18, x49,
                                    x96, x97, x50, x99, x4, x3, x80, x70,
x83, x58,
                                    x5, x37, x27, x12, x66))
ind_Attr2<-names(test_var)
```

```r
# Build best ada boost model
ada1<-ada(x = train[,ind_Attr1],
          y = train$y,
          iter=100, loss="logistic",verbose=TRUE) # 100 Iterations
```

```
## FINAL: iter= 20  rate= 7.270273e-10
## FINAL: iter= 20  rate= 7.350812e-10
## FINAL: iter= 20  rate= 7.978094e-10
## FINAL: iter= 20  rate= 7.891144e-10
## FINAL: iter= 20  rate= 8.541243e-10
## FINAL: iter= 20  rate= 9.257497e-10
## FINAL: iter= 20  rate= 9.199033e-10
## FINAL: iter= 20  rate= 8.573362e-10
## FINAL: iter= 20  rate= 9.067585e-10
## FINAL: iter= 20  rate= 9.542565e-10
## FINAL: iter= 20  rate= 9.066468e-10
## FINAL: iter= 20  rate= 1.050889e-09
## FINAL: iter= 20  rate= 8.146492e-10
## FINAL: iter= 20  rate= 9.74536e-10
## FINAL: iter= 20  rate= 8.835592e-10
## FINAL: iter= 20  rate= 7.159046e-10
## FINAL: iter= 20  rate= 4.352105e-10
## FINAL: iter= 20  rate= 8.680678e-10
## FINAL: iter= 20  rate= 8.976717e-10
## FINAL: iter= 20  rate= 6.75081e-10
## FINAL: iter= 20  rate= 9.836579e-10
## FINAL: iter= 20  rate= 5.673822e-10
## FINAL: iter= 20  rate= 5.911755e-10
## FINAL: iter= 20  rate= 8.103907e-10
## FINAL: iter= 20  rate= 7.080509e-10
## FINAL: iter= 20  rate= 6.698432e-10
## FINAL: iter= 20  rate= 1.264922e-10
## FINAL: iter= 20  rate= 7.534306e-10
## FINAL: iter= 20  rate= 3.156793e-10
## FINAL: iter= 20  rate= 5.384017e-10
## FINAL: iter= 20  rate= 3.488313e-10
## FINAL: iter= 20  rate= 7.88669e-10
## FINAL: iter= 20  rate= 9.244461e-10
## FINAL: iter= 20  rate= 4.466367e-10
## FINAL: iter= 20  rate= 4.124026e-10
## FINAL: iter= 20  rate= 1.984488e-10
## FINAL: iter= 20  rate= 5.830776e-10
## FINAL: iter= 20  rate= 6.079827e-10
## FINAL: iter= 20  rate= 2.166816e-10
## FINAL: iter= 20  rate= 5.341926e-10
## FINAL: iter= 20  rate= 3.148042e-10
## FINAL: iter= 20  rate= 5.403687e-10
## FINAL: iter= 20  rate= 4.044786e-10
## FINAL: iter= 1  rate= 4.739246e-11
```

```
## FINAL: iter= 20  rate= 4.34258e-10
## FINAL: iter= 20  rate= 4.591445e-10
## FINAL: iter= 20  rate= 3.149217e-10
## FINAL: iter= 20  rate= 2.811438e-10
## FINAL: iter= 20  rate= 3.559336e-10
## FINAL: iter= 20  rate= 3.04989e-10
## FINAL: iter= 20  rate= 2.232833e-10
## FINAL: iter= 20  rate= 4.242693e-10
## FINAL: iter= 20  rate= 2.464675e-10
## FINAL: iter= 20  rate= 3.399279e-10
## FINAL: iter= 20  rate= 2.431495e-10
## FINAL: iter= 20  rate= 4.539175e-10
## FINAL: iter= 20  rate= 5.294197e-10
## FINAL: iter= 20  rate= 3.835664e-10
## FINAL: iter= 20  rate= 3.43124e-10
## FINAL: iter= 20  rate= 2.211558e-10
## FINAL: iter= 20  rate= 3.906954e-10
## FINAL: iter= 20  rate= 1.104908e-10
## FINAL: iter= 20  rate= 2.0413e-10
## FINAL: iter= 20  rate= 7.36642e-10
## FINAL: iter= 20  rate= 1.379461e-10
## FINAL: iter= 20  rate= 6.117702e-10
## FINAL: iter= 20  rate= 1.781088e-10
## FINAL: iter= 20  rate= 1.693436e-10
## FINAL: iter= 20  rate= 5.242139e-10
## FINAL: iter= 20  rate= 1.948211e-10
## FINAL: iter= 20  rate= 3.054947e-10
## FINAL: iter= 20  rate= 8.227172e-10
## FINAL: iter= 1  rate= 9.334476e-11
## FINAL: iter= 20  rate= 2.015881e-10
## FINAL: iter= 20  rate= 2.174163e-10
## FINAL: iter= 20  rate= 5.851969e-10
## FINAL: iter= 20  rate= 1.571685e-10
## FINAL: iter= 20  rate= 3.991679e-10
## FINAL: iter= 20  rate= 3.393663e-10
## FINAL: iter= 20  rate= 3.217976e-10
## FINAL: iter= 20  rate= 3.846654e-10
## FINAL: iter= 20  rate= 2.464514e-10
## FINAL: iter= 20  rate= 4.126385e-10
## FINAL: iter= 20  rate= 4.595006e-10
## FINAL: iter= 20  rate= 2.00796e-10
## FINAL: iter= 20  rate= 7.08271e-10
## FINAL: iter= 20  rate= 2.279958e-10
## FINAL: iter= 20  rate= 5.640797e-10
## FINAL: iter= 20  rate= 5.554978e-10
## FINAL: iter= 20  rate= 6.729028e-10
## FINAL: iter= 20  rate= 1.911991e-10
## FINAL: iter= 20  rate= 1.792385e-10
## FINAL: iter= 20  rate= 4.954627e-10
## FINAL: iter= 20  rate= 2.952691e-10
```

```
## FINAL: iter= 20  rate= 4.930212e-10
## FINAL: iter= 20  rate= 3.219048e-10
## FINAL: iter= 20  rate= 2.805726e-10
## FINAL: iter= 1  rate= 5.196241e-11
## FINAL: iter= 20  rate= 4.343211e-10
## FINAL: iter= 20  rate= 1.608355e-10
```

```r
# Look at the model summary
summary(ada1)
```

```
## Call:
## ada(train[, ind_Attr1], y = train$y, loss = "logistic", iter = 100,
##     verbose = TRUE)
##
## Loss: logistic Method: discrete   Iteration: 100
##
## Training Results
##
## Accuracy: 0.932 Kappa: 0.768
```

```r
# Build best ada boost model
ada2<-ada(x = test[,ind_Attr2],
          y = test$y,
          iter=100, loss="logistic",verbose=TRUE) # 100 Iterations
```

```
## FINAL: iter= 20  rate= 1.195168e-08
## FINAL: iter= 20  rate= 1.373453e-08
## FINAL: iter= 20  rate= 1.200256e-08
## FINAL: iter= 20  rate= 1.288116e-08
## FINAL: iter= 20  rate= 1.49356e-08
## FINAL: iter= 20  rate= 1.349332e-08
## FINAL: iter= 20  rate= 1.488461e-08
## FINAL: iter= 20  rate= 1.818764e-08
## FINAL: iter= 20  rate= 1.479619e-08
## FINAL: iter= 20  rate= 1.54999e-08
## FINAL: iter= 20  rate= 1.678202e-08
## FINAL: iter= 20  rate= 1.234952e-08
## FINAL: iter= 20  rate= 1.27415e-08
## FINAL: iter= 20  rate= 1.573005e-08
## FINAL: iter= 20  rate= 1.348809e-08
## FINAL: iter= 20  rate= 1.208222e-08
## FINAL: iter= 20  rate= 1.387138e-08
## FINAL: iter= 20  rate= 1.468057e-08
## FINAL: iter= 20  rate= 1.034939e-08
## FINAL: iter= 20  rate= 1.41367e-08
## FINAL: iter= 20  rate= 1.370265e-08
## FINAL: iter= 20  rate= 1.559521e-08
## FINAL: iter= 20  rate= 1.111798e-08
## FINAL: iter= 20  rate= 1.407988e-08
## FINAL: iter= 20  rate= 9.399562e-09
## FINAL: iter= 20  rate= 1.498483e-08
```

```
## FINAL: iter= 20  rate= 1.065625e-08
## FINAL: iter= 20  rate= 1.547611e-08
## FINAL: iter= 20  rate= 1.048973e-08
## FINAL: iter= 20  rate= 4.153946e-09
## FINAL: iter= 20  rate= 8.159132e-09
## FINAL: iter= 20  rate= 1.450159e-08
## FINAL: iter= 20  rate= 7.947011e-09
## FINAL: iter= 20  rate= 8.349986e-09
## FINAL: iter= 20  rate= 9.488003e-09
## FINAL: iter= 20  rate= 1.254506e-08
## FINAL: iter= 20  rate= 5.528693e-09
## FINAL: iter= 20  rate= 9.199314e-09
## FINAL: iter= 20  rate= 4.944792e-09
## FINAL: iter= 20  rate= 8.118268e-09
## FINAL: iter= 20  rate= 3.86583e-09
## FINAL: iter= 20  rate= 7.194203e-09
## FINAL: iter= 20  rate= 7.586195e-09
## FINAL: iter= 20  rate= 1.147636e-08
## FINAL: iter= 20  rate= 7.841426e-09
## FINAL: iter= 20  rate= 7.267615e-09
## FINAL: iter= 20  rate= 1.077324e-08
## FINAL: iter= 20  rate= 1.348579e-08
## FINAL: iter= 20  rate= 1.081396e-08
## FINAL: iter= 20  rate= 7.298567e-09
## FINAL: iter= 20  rate= 3.974059e-09
## FINAL: iter= 20  rate= 9.121228e-09
## FINAL: iter= 20  rate= 8.190527e-09
## FINAL: iter= 20  rate= 7.684173e-09
## FINAL: iter= 20  rate= 6.024397e-09
## FINAL: iter= 20  rate= 9.411653e-09
## FINAL: iter= 20  rate= 2.775772e-09
## FINAL: iter= 20  rate= 8.306979e-09
## FINAL: iter= 20  rate= 9.554501e-09
## FINAL: iter= 20  rate= 1.332125e-08
## FINAL: iter= 20  rate= 2.465232e-09
## FINAL: iter= 20  rate= 7.740251e-09
## FINAL: iter= 20  rate= 9.487451e-09
## FINAL: iter= 20  rate= 1.666704e-08
## FINAL: iter= 20  rate= 6.940449e-09
## FINAL: iter= 20  rate= 2.494297e-09
## FINAL: iter= 20  rate= 3.037242e-09
## FINAL: iter= 20  rate= 8.780883e-09
## FINAL: iter= 20  rate= 1.022022e-08
## FINAL: iter= 20  rate= 4.649636e-09
## FINAL: iter= 20  rate= 8.929052e-09
## FINAL: iter= 20  rate= 3.06658e-09
## FINAL: iter= 20  rate= 9.755575e-09
## FINAL: iter= 20  rate= 7.227495e-09
## FINAL: iter= 20  rate= 9.860004e-09
## FINAL: iter= 20  rate= 4.144415e-09
```

```
## FINAL: iter= 20  rate= 7.500793e-09
## FINAL: iter= 20  rate= 1.011373e-08
## FINAL: iter= 20  rate= 8.539173e-09
## FINAL: iter= 20  rate= 1.376126e-08
## FINAL: iter= 20  rate= 9.434911e-09
## FINAL: iter= 20  rate= 2.07325e-09
## FINAL: iter= 20  rate= 7.438951e-09
## FINAL: iter= 20  rate= 6.339554e-09
## FINAL: iter= 20  rate= 9.644056e-09
## FINAL: iter= 20  rate= 8.996895e-09
## FINAL: iter= 20  rate= 6.441065e-09
## FINAL: iter= 20  rate= 7.258042e-09
## FINAL: iter= 20  rate= 3.824125e-09
## FINAL: iter= 20  rate= 3.950805e-09
## FINAL: iter= 20  rate= 1.049527e-08
## FINAL: iter= 20  rate= 5.176843e-09
## FINAL: iter= 20  rate= 1.655619e-09
## FINAL: iter= 20  rate= 6.153958e-09
## FINAL: iter= 20  rate= 1.25339e-08
## FINAL: iter= 20  rate= 7.396718e-09
## FINAL: iter= 20  rate= 1.10849e-08
## FINAL: iter= 20  rate= 9.085713e-09
## FINAL: iter= 20  rate= 7.867943e-09
## FINAL: iter= 20  rate= 9.773363e-09

# Look at the model summary
summary(ada2)

## Call:
## ada(test[, ind_Attr2], y = test$y, loss = "logistic", iter = 100,
##     verbose = TRUE)
##
## Loss: logistic Method: discrete   Iteration: 100
##
## Training Results
##
## Accuracy: 0.968 Kappa: 0.897

# Predict on train data
pred_Train<-predict(ada1, train[,ind_Attr1])

# Build confusion matrix and find accuracy
cm_Train = table(train$y, pred_Train)
accu_Train= sum(diag(cm_Train))/sum(cm_Train)
rm(pred_Train, cm_Train)

# Predict on test data
pred_Test = predict(ada1, test[,ind_Attr2])

# Build confusion matrix and find accuracy
```

```
cm_Test = table(test$y, pred_Test)
accu_Test= sum(diag(cm_Test))/sum(cm_Test)
rm(pred_Test, cm_Test)

#Validation

SF_Validation_Copy$pred_ada<-predict(ada1, SF_Validation_Copy)
write.csv(SF_Validation_Copy$pred_ada,
"C:/Users/puj83/OneDrive/CV/Cases/InsuranceX/SF_Validation_AdaBoost_Probabili
ties.csv")
```