CKME 136 - Churn Analysis in Telecommunications

Antoaneta Mihaila

November 20th, 2017

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

Use RStudio for this assignment. Edit the file Churn and insert your R code where wherever you see the string "INSERT YOUR ANSWER HERE"

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

# 1. Read the data

library(caTools)  
library(caret)

## Warning: package 'caret' was built under R version 3.3.3

## Loading required package: lattice

## Warning: package 'lattice' was built under R version 3.3.3

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.3.3

library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 3.3.3

## Loading required package: rpart

## Warning: package 'rpart' was built under R version 3.3.3

library(gbm)

## Warning: package 'gbm' was built under R version 3.3.3

## Loading required package: survival

##   
## Attaching package: 'survival'

## The following object is masked from 'package:caret':  
##   
## cluster

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.3

library(kernlab)

##   
## Attaching package: 'kernlab'

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

#install.packages("h2o")  
library(h2o)

## Warning: package 'h2o' was built under R version 3.3.3

##   
## ----------------------------------------------------------------------  
##   
## Your next step is to start H2O:  
## > h2o.init()  
##   
## For H2O package documentation, ask for help:  
## > ??h2o  
##   
## After starting H2O, you can use the Web UI at http://localhost:54321  
## For more information visit http://docs.h2o.ai  
##   
## ----------------------------------------------------------------------

##   
## Attaching package: 'h2o'

## The following objects are masked from 'package:stats':  
##   
## cor, sd, var

## The following objects are masked from 'package:base':  
##   
## %\*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,  
## colnames<-, ifelse, is.character, is.factor, is.numeric, log,  
## log10, log1p, log2, round, signif, trunc

#install.packages("nnet")  
library(nnet)  
library(randomForest)

## Warning: package 'randomForest' was built under R version 3.3.3

## randomForest 4.6-12

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

##   
## Attaching package: 'randomForest'

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

library(xgboost)

## Warning: package 'xgboost' was built under R version 3.3.3

library(rpart)  
library(ROCR)

## Warning: package 'ROCR' was built under R version 3.3.3

## Loading required package: gplots

##   
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':  
##   
## lowess

library(pROC)

## Warning: package 'pROC' was built under R version 3.3.3

## Type 'citation("pROC")' for a citation.

##   
## Attaching package: 'pROC'

## The following object is masked from 'package:h2o':  
##   
## var

## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

library(ggplot2)  
library(corrplot)

## Warning: package 'corrplot' was built under R version 3.3.3

library(rpart.plot)  
  
churn\_data <- read.csv("churn.csv", sep = ",",header = F, stringsAsFactors = F)  
  
names(churn\_data) <- c("state", "account\_length","area\_code","phone\_number","international\_plan","voice\_mail\_plan",  
"number\_vmail\_messages","total\_day\_minutes","total\_day\_calls","total\_day\_charge",  
"total\_eve\_minutes","total\_eve\_calls","total\_eve\_charge",  
"total\_night\_minutes","total\_night\_calls","total\_night\_charge",  
"total\_intl\_minutes","total\_intl\_calls","total\_intl\_charge","number\_customer\_service\_calls","churn")  
head(churn\_data)

## state account\_length area\_code phone\_number international\_plan  
## 1 KS 128 415 382-4657 no  
## 2 OH 107 415 371-7191 no  
## 3 NJ 137 415 358-1921 no  
## 4 OH 84 408 375-9999 yes  
## 5 OK 75 415 330-6626 yes  
## 6 AL 118 510 391-8027 yes  
## voice\_mail\_plan number\_vmail\_messages total\_day\_minutes total\_day\_calls  
## 1 yes 25 265.1 110  
## 2 yes 26 161.6 123  
## 3 no 0 243.4 114  
## 4 no 0 299.4 71  
## 5 no 0 166.7 113  
## 6 no 0 223.4 98  
## total\_day\_charge total\_eve\_minutes total\_eve\_calls total\_eve\_charge  
## 1 45.07 197.4 99 16.78  
## 2 27.47 195.5 103 16.62  
## 3 41.38 121.2 110 10.30  
## 4 50.90 61.9 88 5.26  
## 5 28.34 148.3 122 12.61  
## 6 37.98 220.6 101 18.75  
## total\_night\_minutes total\_night\_calls total\_night\_charge  
## 1 244.7 91 11.01  
## 2 254.4 103 11.45  
## 3 162.6 104 7.32  
## 4 196.9 89 8.86  
## 5 186.9 121 8.41  
## 6 203.9 118 9.18  
## total\_intl\_minutes total\_intl\_calls total\_intl\_charge  
## 1 10.0 3 2.70  
## 2 13.7 3 3.70  
## 3 12.2 5 3.29  
## 4 6.6 7 1.78  
## 5 10.1 3 2.73  
## 6 6.3 6 1.70  
## number\_customer\_service\_calls churn  
## 1 1 False.  
## 2 1 False.  
## 3 0 False.  
## 4 2 False.  
## 5 3 False.  
## 6 0 False.

# 2. Descriprive Statistics

summary(churn\_data)

## state account\_length area\_code phone\_number   
## Length:5000 Min. : 1.0 Min. :408.0 Length:5000   
## Class :character 1st Qu.: 73.0 1st Qu.:408.0 Class :character   
## Mode :character Median :100.0 Median :415.0 Mode :character   
## Mean :100.3 Mean :436.9   
## 3rd Qu.:127.0 3rd Qu.:415.0   
## Max. :243.0 Max. :510.0   
## international\_plan voice\_mail\_plan number\_vmail\_messages  
## Length:5000 Length:5000 Min. : 0.000   
## Class :character Class :character 1st Qu.: 0.000   
## Mode :character Mode :character Median : 0.000   
## Mean : 7.755   
## 3rd Qu.:17.000   
## Max. :52.000   
## total\_day\_minutes total\_day\_calls total\_day\_charge total\_eve\_minutes  
## Min. : 0.0 Min. : 0 Min. : 0.00 Min. : 0.0   
## 1st Qu.:143.7 1st Qu.: 87 1st Qu.:24.43 1st Qu.:166.4   
## Median :180.1 Median :100 Median :30.62 Median :201.0   
## Mean :180.3 Mean :100 Mean :30.65 Mean :200.6   
## 3rd Qu.:216.2 3rd Qu.:113 3rd Qu.:36.75 3rd Qu.:234.1   
## Max. :351.5 Max. :165 Max. :59.76 Max. :363.7   
## total\_eve\_calls total\_eve\_charge total\_night\_minutes total\_night\_calls  
## Min. : 0.0 Min. : 0.00 Min. : 0.0 Min. : 0.00   
## 1st Qu.: 87.0 1st Qu.:14.14 1st Qu.:166.9 1st Qu.: 87.00   
## Median :100.0 Median :17.09 Median :200.4 Median :100.00   
## Mean :100.2 Mean :17.05 Mean :200.4 Mean : 99.92   
## 3rd Qu.:114.0 3rd Qu.:19.90 3rd Qu.:234.7 3rd Qu.:113.00   
## Max. :170.0 Max. :30.91 Max. :395.0 Max. :175.00   
## total\_night\_charge total\_intl\_minutes total\_intl\_calls total\_intl\_charge  
## Min. : 0.000 Min. : 0.00 Min. : 0.000 Min. :0.000   
## 1st Qu.: 7.510 1st Qu.: 8.50 1st Qu.: 3.000 1st Qu.:2.300   
## Median : 9.020 Median :10.30 Median : 4.000 Median :2.780   
## Mean : 9.018 Mean :10.26 Mean : 4.435 Mean :2.771   
## 3rd Qu.:10.560 3rd Qu.:12.00 3rd Qu.: 6.000 3rd Qu.:3.240   
## Max. :17.770 Max. :20.00 Max. :20.000 Max. :5.400   
## number\_customer\_service\_calls churn   
## Min. :0.00 Length:5000   
## 1st Qu.:1.00 Class :character   
## Median :1.00 Mode :character   
## Mean :1.57   
## 3rd Qu.:2.00   
## Max. :9.00

str(churn\_data)

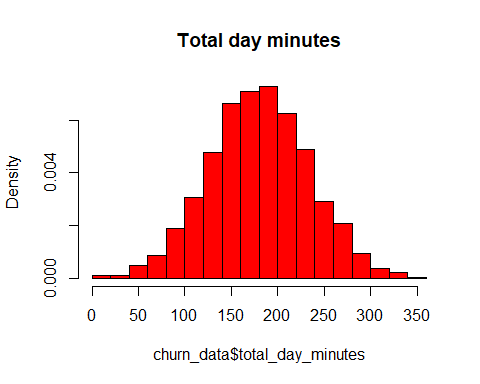
## 'data.frame': 5000 obs. of 21 variables:  
## $ state : chr "KS" "OH" "NJ" "OH" ...  
## $ account\_length : int 128 107 137 84 75 118 121 147 117 141 ...  
## $ area\_code : int 415 415 415 408 415 510 510 415 408 415 ...  
## $ phone\_number : chr " 382-4657" " 371-7191" " 358-1921" " 375-9999" ...  
## $ international\_plan : chr " no" " no" " no" " yes" ...  
## $ voice\_mail\_plan : chr " yes" " yes" " no" " no" ...  
## $ number\_vmail\_messages : int 25 26 0 0 0 0 24 0 0 37 ...  
## $ total\_day\_minutes : num 265 162 243 299 167 ...  
## $ total\_day\_calls : int 110 123 114 71 113 98 88 79 97 84 ...  
## $ total\_day\_charge : num 45.1 27.5 41.4 50.9 28.3 ...  
## $ total\_eve\_minutes : num 197.4 195.5 121.2 61.9 148.3 ...  
## $ total\_eve\_calls : int 99 103 110 88 122 101 108 94 80 111 ...  
## $ total\_eve\_charge : num 16.78 16.62 10.3 5.26 12.61 ...  
## $ total\_night\_minutes : num 245 254 163 197 187 ...  
## $ total\_night\_calls : int 91 103 104 89 121 118 118 96 90 97 ...  
## $ total\_night\_charge : num 11.01 11.45 7.32 8.86 8.41 ...  
## $ total\_intl\_minutes : num 10 13.7 12.2 6.6 10.1 6.3 7.5 7.1 8.7 11.2 ...  
## $ total\_intl\_calls : int 3 3 5 7 3 6 7 6 4 5 ...  
## $ total\_intl\_charge : num 2.7 3.7 3.29 1.78 2.73 1.7 2.03 1.92 2.35 3.02 ...  
## $ number\_customer\_service\_calls: int 1 1 0 2 3 0 3 0 1 0 ...  
## $ churn : chr " False." " False." " False." " False." ...

nrow(churn\_data)

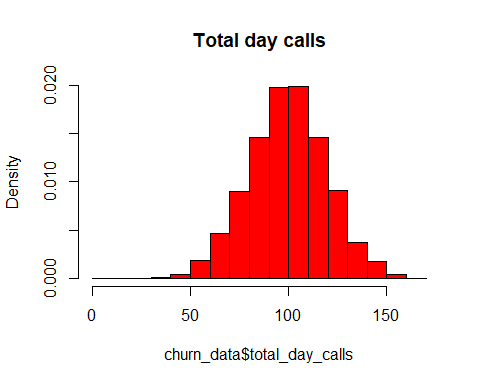
## [1] 5000

# 3. Plot the data

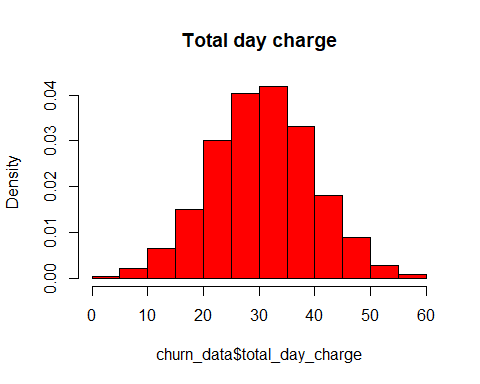
#Check the distribution of the data  
###################  
h1 <- hist(churn\_data$total\_day\_minutes, breaks = 15, col ="red", main = "Total day minutes", freq = FALSE)



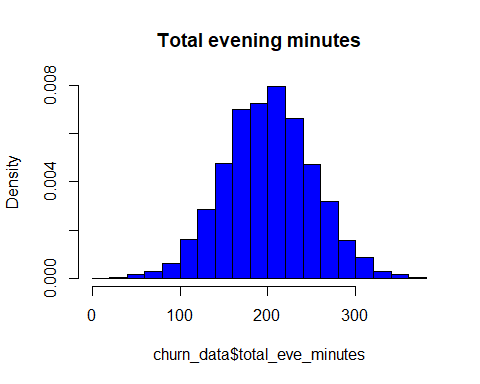
h2 <- hist(churn\_data$total\_day\_calls, breaks = 15, col ="red", main = "Total day calls", freq = FALSE)



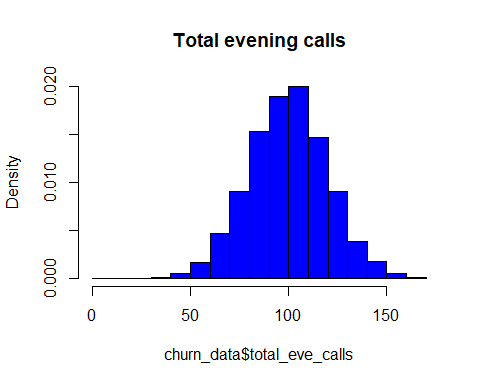
h3 <- hist(churn\_data$total\_day\_charge, breaks = 15, col ="red", main = "Total day charge", freq = FALSE)



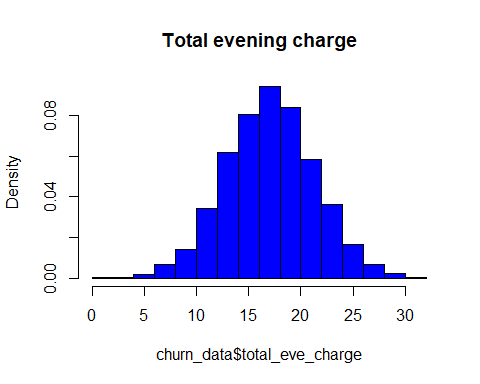
h4 <- hist(churn\_data$total\_eve\_minutes, breaks = 15, col ="blue", main = "Total evening minutes", freq = FALSE)



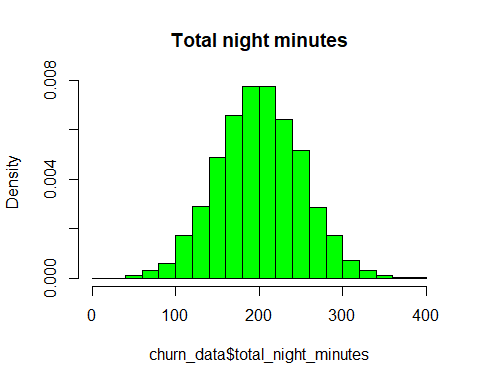
h5 <- hist(churn\_data$total\_eve\_calls, breaks = 15, col ="blue", main = "Total evening calls", freq = FALSE)



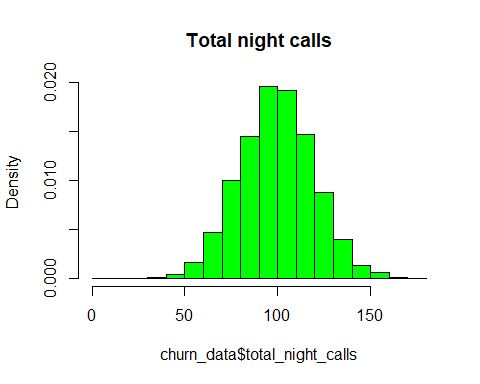
h6 <- hist(churn\_data$total\_eve\_charge, breaks = 15, col ="blue", main = "Total evening charge", freq = FALSE)



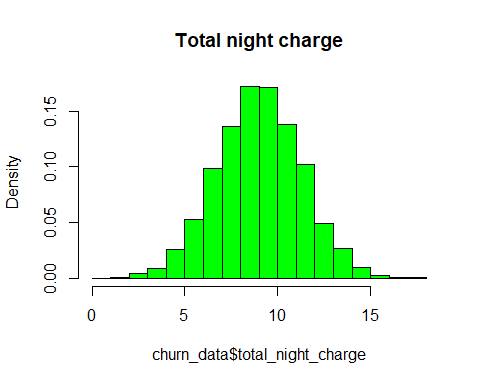
h7 <- hist(churn\_data$total\_night\_minutes, breaks = 15, col ="green", main = "Total night minutes", freq = FALSE)



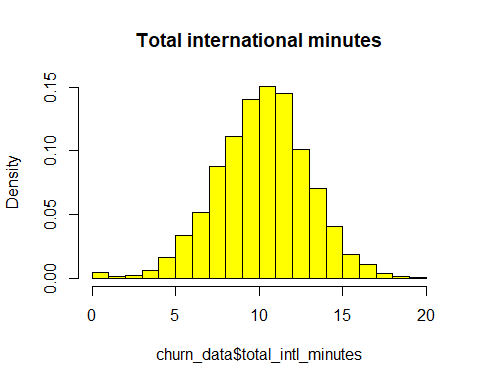
h8 <- hist(churn\_data$total\_night\_calls, breaks = 15, col ="green", main = "Total night calls", freq = FALSE)



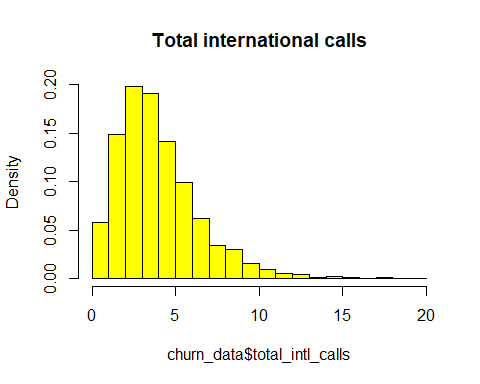
h9 <- hist(churn\_data$total\_night\_charge, breaks = 15, col ="green", main = "Total night charge", freq = FALSE)



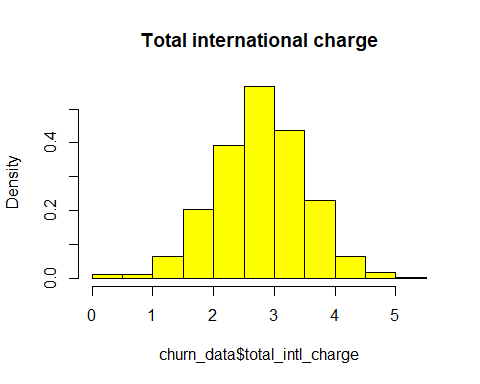
h10 <- hist(churn\_data$total\_intl\_minutes, breaks = 15, col ="yellow", main = "Total international minutes", freq = FALSE)



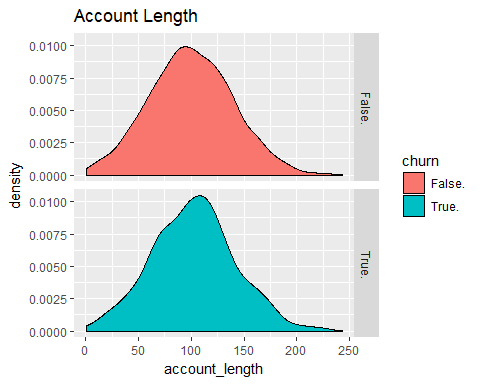
h11 <- hist(churn\_data$total\_intl\_calls, breaks = 15, col ="yellow", main = "Total international calls", freq = FALSE)



h12 <- hist(churn\_data$total\_intl\_charge, breaks = 15, col ="yellow", main = "Total international charge", freq = FALSE)



# we can notice that only total international minutes distribution is right squewed  
  
# Check if the account length has an impact on churn  
ggplot(churn\_data, aes(account\_length, fill = churn)) + geom\_density() + facet\_grid(churn~.)+labs(title="Account Length")



# It doesn't look that the account lenght has an impact on churn  
  
  
#Total day minutes summary  
summary(churn\_data$total\_day\_minutes)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0 143.7 180.1 180.3 216.2 351.5

#Total evening minutes summary  
summary(churn\_data$total\_eve\_minutes)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0 166.4 201.0 200.6 234.1 363.7

#Total night minutes summary  
summary(churn\_data$total\_night\_minutes)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0 166.9 200.4 200.4 234.7 395.0

#Total international minutes summary  
summary(churn\_data$total\_intl\_minutes)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.00 8.50 10.30 10.26 12.00 20.00

# If we look at the quantiles of the day, evening, night calls duration we can see that the day time has the shortest call duration  
#   
# Take a look at the charges  
summary(churn\_data$total\_day\_charge)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.00 24.43 30.62 30.65 36.75 59.76

summary(churn\_data$total\_eve\_charge)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.00 14.14 17.09 17.05 19.90 30.91

summary(churn\_data$total\_night\_charge)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.000 7.510 9.020 9.018 10.560 17.770

summary(churn\_data$total\_intl\_charge)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.000 2.300 2.780 2.771 3.240 5.400

# We notice that the day charge is higher than evening and night charge and the duration of the day calls are shorter than duration of evening and night calls

# 4. Clean the data

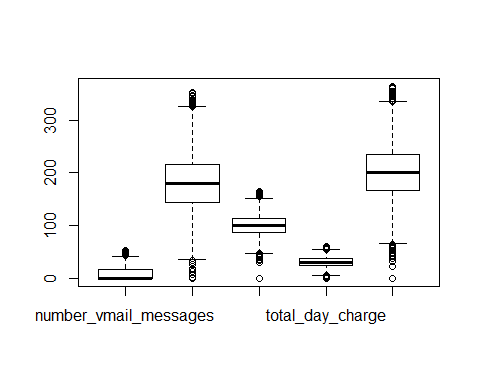
# check the missing data  
# From the summary we can see that are not missing data  
# We can also check with   
table(is.na(churn\_data))

##   
## FALSE   
## 105000

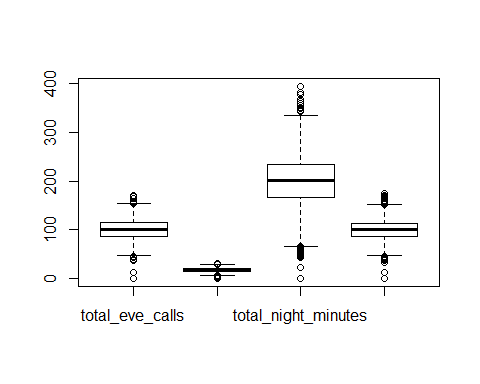
# Remove the columns that will not be used  
churn\_data$state <-NULL  
churn\_data$phone\_number <- NULL  
  
#str(churn\_data)  
# Change the international\_plan, voice\_mail\_plan, churn to factors  
churn\_data$churn <- as.factor(ifelse(grepl("False", churn\_data$churn), "NotChurner", "Churner"))  
churn\_data$international\_plan <- as.factor(ifelse(grepl("yes",churn\_data$international\_plan ), "IntlPlan", "NoIntlPlan"))  
churn\_data$voice\_mail\_plan <- as.factor(ifelse(grepl("yes", churn\_data$voice\_mail\_plan), "VmPlan", "NoVmPlan"))  
churn\_data$area\_code <- as.factor(churn\_data$area\_code)  
table(churn\_data$churn)

##   
## Churner NotChurner   
## 707 4293

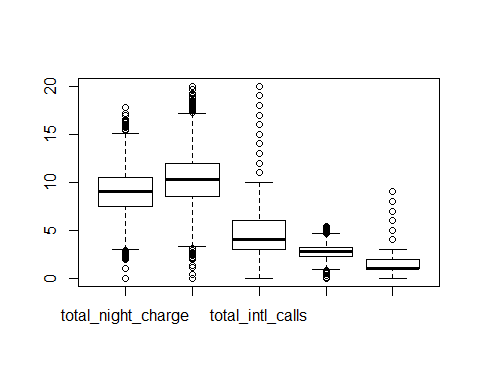
#table(churn\_data$international\_plan)  
#table(churn\_data$voice\_mail\_plan)  
###########  
#Check for outliers.Plot the boxplots for the numeric variables  
boxplot(churn\_data[5:9])



boxplot(churn\_data[10:13])



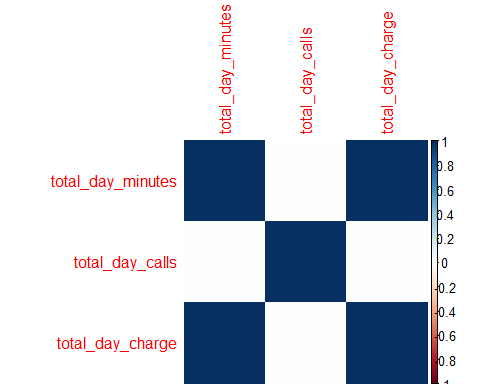
boxplot(churn\_data[14:18])



#Check the correlation between independent variables  
cor.day <- cor(churn\_data[,6:8], use = "complete.obs", method = "pearson")  
cor.day

## total\_day\_minutes total\_day\_calls total\_day\_charge  
## total\_day\_minutes 1.000000000 0.001935149 0.999999951  
## total\_day\_calls 0.001935149 1.000000000 0.001935884  
## total\_day\_charge 0.999999951 0.001935884 1.000000000

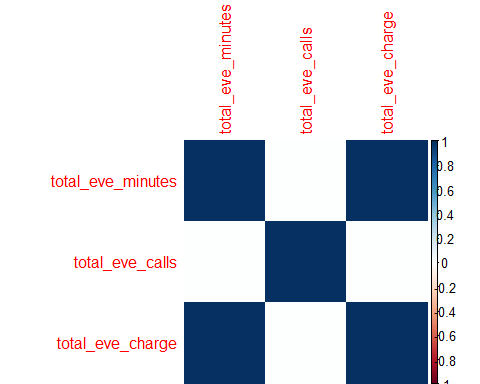
corrplot(cor.day, method = "color")



#We notice a strong correlation between total day minutes and total day charge  
cor.eve <- cor(churn\_data[,9:11], use = "complete.obs", method = "pearson")  
cor.eve

## total\_eve\_minutes total\_eve\_calls total\_eve\_charge  
## total\_eve\_minutes 1.000000000 0.002763019 0.999999775  
## total\_eve\_calls 0.002763019 1.000000000 0.002778097  
## total\_eve\_charge 0.999999775 0.002778097 1.000000000

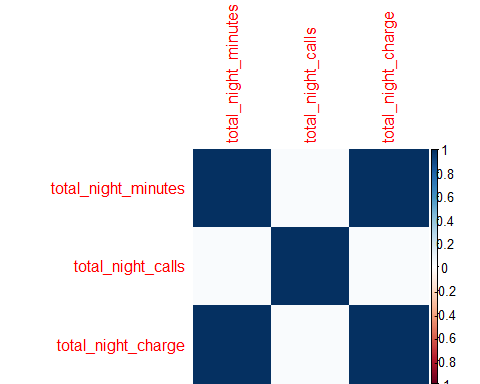
corrplot(cor.eve, method = "color")



#We notice a strong correlation between total evening minutes and total evening charge  
cor.night <- cor(churn\_data[,12:14], use = "complete.obs", method = "pearson")  
cor.night

## total\_night\_minutes total\_night\_calls  
## total\_night\_minutes 1.00000000 0.02697182  
## total\_night\_calls 0.02697182 1.00000000  
## total\_night\_charge 0.99999921 0.02694949  
## total\_night\_charge  
## total\_night\_minutes 0.99999921  
## total\_night\_calls 0.02694949  
## total\_night\_charge 1.00000000

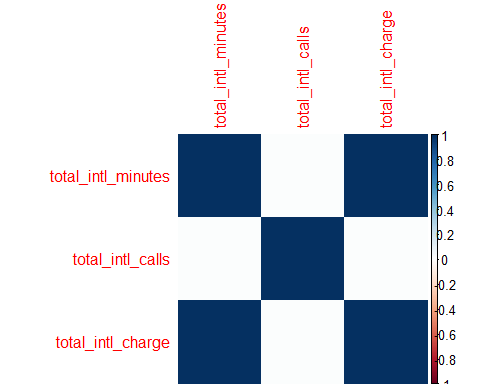
corrplot(cor.night, method = "color")



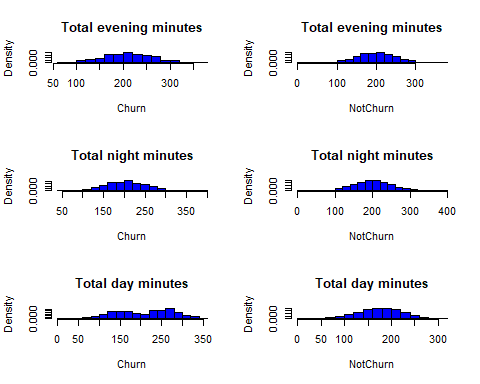
#We notice a strong correlation between total night minutes and total night charge  
cor.intl <- cor(churn\_data[,15:17], use = "complete.obs", method = "pearson")  
cor.intl

## total\_intl\_minutes total\_intl\_calls total\_intl\_charge  
## total\_intl\_minutes 1.00000000 0.01679148 0.99999266  
## total\_intl\_calls 0.01679148 1.00000000 0.01690009  
## total\_intl\_charge 0.99999266 0.01690009 1.00000000

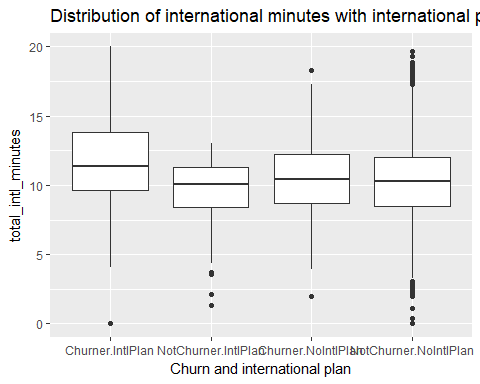
corrplot(cor.intl, method = "color")



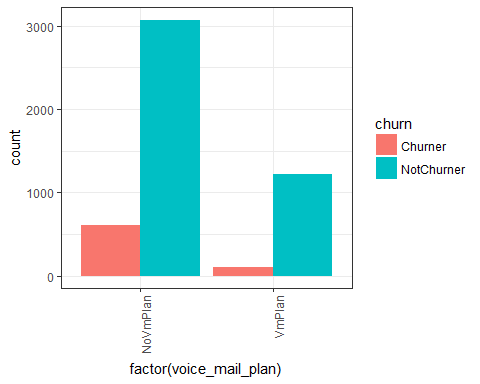
#We notice a strong correlation between total international minutes and total international charge, will will drop the charge  
  
  
#Check the influence on call duration on churn  
par(mfrow=c(3,2))  
hist(churn\_data$total\_eve\_minutes[churn\_data$churn=="Churner"], xlab ="Churn", main = "Total evening minutes", col="Blue", freq=FALSE, breaks=15)  
hist(churn\_data$total\_eve\_minutes[churn\_data$churn=="NotChurner"], xlab = "NotChurn", main = "Total evening minutes", col = "Blue", freq = FALSE, breaks = 15)  
hist(churn\_data$total\_night\_minutes[churn\_data$churn=="Churner"], xlab ="Churn", main = "Total night minutes", col="Blue", freq= FALSE, breaks=15)  
hist(churn\_data$total\_night\_minutes[churn\_data$churn=="NotChurner"], xlab = "NotChurn", main = "Total night minutes", col = "Blue", freq = FALSE,breaks = 15)  
hist(churn\_data$total\_day\_minutes[churn\_data$churn=="Churner"], xlab ="Churn", main = "Total day minutes", col="Blue", freq=FALSE, breaks =15)  
hist(churn\_data$total\_day\_minutes[churn\_data$churn=="NotChurner"], xlab = "NotChurn", main = "Total day minutes", col = "Blue", freq=FALSE, breaks=15)



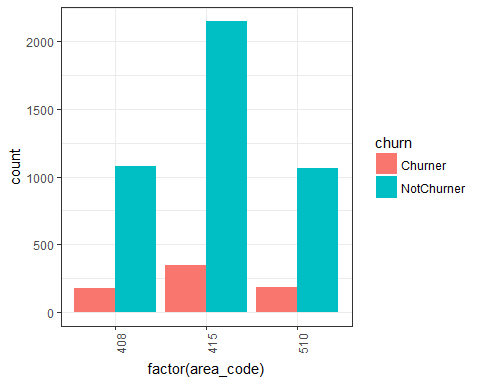
#Check the distribution of international minutes with churn  
p <- qplot(interaction(churn,international\_plan), total\_intl\_minutes, data=churn\_data, geom="boxplot",  
main = "Distribution of international minutes with international plan and churn", xlab = "Churn and international plan")  
p



#We can see that the customers with longer international call duration are more likely to churn  
#  
#Check if the customer has voice mail plan has an impact on churn  
  
vmplan <- ggplot(churn\_data,aes(x=factor(voice\_mail\_plan)))   
vmplan <- vmplan + geom\_bar(aes(fill=churn),position = "dodge")  
vmplan + theme\_bw() + theme(axis.text.x = element\_text(angle = 90, hjust = 1))



#We can see that the customers with voice mail plan are less likely to churn than the customers without a voice mail plan  
#16%of customers without voice mail plan will churn  
#7% of customers with voice plan will churn  
  
# Check the influence of area code on churn  
  
area <- ggplot(churn\_data,aes(x=factor(area\_code)))   
area <- area + geom\_bar(aes(fill=churn),position = "dodge")  
area + theme\_bw() + theme(axis.text.x = element\_text(angle = 90, hjust = 1))



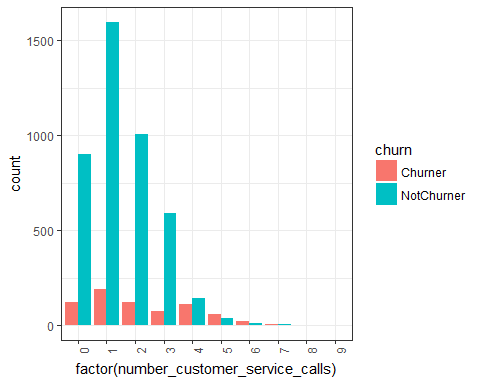
table(churn\_data$churn, churn\_data$area\_code)

##   
## 408 415 510  
## Churner 177 346 184  
## NotChurner 1082 2149 1062

#The area 416 has the high number of customers but the less percentage in churn (13% compare with 14% in the other 2 areas)   
#Check the number customer service calls and churn  
table(churn\_data$number\_customer\_service\_calls, churn\_data$churn)

##   
## Churner NotChurner  
## 0 121 902  
## 1 190 1596  
## 2 122 1005  
## 3 73 592  
## 4 111 141  
## 5 58 38  
## 6 22 12  
## 7 7 6  
## 8 1 1  
## 9 2 0

csc <- ggplot(churn\_data,aes(x=factor(number\_customer\_service\_calls)))   
csc <- csc + geom\_bar(aes(fill=churn),position = "dodge")  
csc + theme\_bw() + theme(axis.text.x = element\_text(angle = 90, hjust = 1))



# Check the number of zeros in the number customer service calls and voice mail messages  
a <- length(which(churn\_data$number\_customer\_service\_calls==0))  
b <- length(which(churn\_data$number\_vmail\_messages==0))  
a

## [1] 1023

b

## [1] 3678

#Conclusions from the descriptive statistics:  
#We have 707 churners out of 5000 customers, for a pure guess 14% of the customers will churn  
#1.There is a higher charge for calls during day time then for all other calls.  
#2.The duration of calls during day is smaller than the duration of other calls  
#There is a high number of zeros in the number of voice mail messages(3678)  
#There is a high percentage of churners among the customers with the international plan  
#Area code and account lenght don't have a high influence on churn  
#there is a high correlation between call duration and charge  
#str(churn\_data)

# 5.Split the data

#Split the data set in training and validation (80% / 20%)  
# 5.1 shuffle the full set  
set.seed(3152)  
churn\_data <- churn\_data[sample.int(nrow(churn\_data)),]  
# 5.2 Split the data set in train and test  
set.seed(441)  
split\_t <- sample.split(churn\_data$churn, SplitRatio = 0.8)  
churn\_train <- subset(churn\_data, split\_t == T)  
churn\_test <- subset(churn\_data, split\_t == F)  
#str(churn\_train)  
#str(churn\_test)  
  
# Compute observation weights (inverse priors) for the training set observations. We will use this weight for decision tree models to balance "Churner" and "NotChurner" and to make sure the "Churner" is present in all trees  
ch\_weight <- sum(churn\_train$churn == 'NotChurner') / sum(churn\_train$churn == 'Churner')  
ch\_weight

## [1] 6.067138

train\_weights <- ifelse(churn\_train$churn == 'Churner', ch\_weight, 1)

# 6.Logistic Regression Model

# 6.1 First set the train control for 'caret'. We do a 4-fold cross-validation and ask   
# for class probabilities  
fitControl <- trainControl(method = "cv", number = 4, classProbs = TRUE, summaryFunction = twoClassSummary)  
  
#6.2 Train the logistic regression model, compute the confusion matrix  
  
#Train the model  
mod.glm <- train(churn ~ ., data = churn\_train, method = "glm", weights = train\_weights,trControl = fitControl, metric = "ROC")  
  
summary(mod.glm)

##   
## Call:  
## NULL  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -6.6872 0.4164 0.7220 1.0600 3.0285   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 4.724e+00 3.738e-01 12.637 < 2e-16 \*\*\*  
## account\_length -2.443e-03 7.383e-04 -3.309 0.000937 \*\*\*  
## area\_code415 1.685e-01 7.230e-02 2.330 0.019800 \*   
## area\_code510 -6.752e-03 8.257e-02 -0.082 0.934824   
## international\_planNoIntlPlan 2.472e+00 9.573e-02 25.826 < 2e-16 \*\*\*  
## voice\_mail\_planVmPlan 1.527e+00 2.836e-01 5.382 7.36e-08 \*\*\*  
## number\_vmail\_messages -1.157e-02 9.054e-03 -1.278 0.201355   
## total\_day\_minutes -4.790e+00 1.737e+00 -2.758 0.005823 \*\*   
## total\_day\_calls -2.558e-03 1.477e-03 -1.731 0.083386 .   
## total\_day\_charge 2.810e+01 1.022e+01 2.750 0.005959 \*\*   
## total\_eve\_minutes -1.608e-01 8.786e-01 -0.183 0.854748   
## total\_eve\_calls 5.382e-05 1.474e-03 0.037 0.970867   
## total\_eve\_charge 1.800e+00 1.034e+01 0.174 0.861747   
## total\_night\_minutes -4.214e-01 4.568e-01 -0.923 0.356260   
## total\_night\_calls 1.024e-03 1.498e-03 0.683 0.494448   
## total\_night\_charge 9.283e+00 1.015e+01 0.914 0.360511   
## total\_intl\_minutes -8.970e-01 2.787e+00 -0.322 0.747573   
## total\_intl\_calls 3.919e-02 1.169e-02 3.354 0.000798 \*\*\*  
## total\_intl\_charge 3.013e+00 1.032e+01 0.292 0.770376   
## number\_customer\_service\_calls -6.205e-01 2.192e-02 -28.308 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 9521.1 on 3999 degrees of freedom  
## Residual deviance: 6972.0 on 3980 degrees of freedom  
## AIC: 6974.2  
##   
## Number of Fisher Scoring iterations: 5

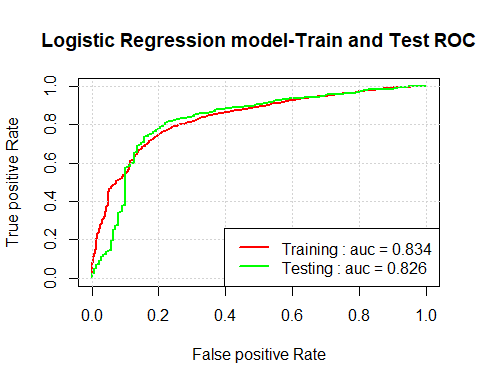
glm\_train\_cm <- confusionMatrix(predict(mod.glm, newdata = churn\_train), churn\_train$churn)  
glm\_test\_cm <- confusionMatrix(predict(mod.glm, newdata = churn\_test), churn\_test$churn)  
glm\_train\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 443 806  
## NotChurner 123 2628  
##   
## Accuracy : 0.7678   
## 95% CI : (0.7543, 0.7808)  
## No Information Rate : 0.8585   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3644   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.7827   
## Specificity : 0.7653   
## Pos Pred Value : 0.3547   
## Neg Pred Value : 0.9553   
## Prevalence : 0.1415   
## Detection Rate : 0.1108   
## Detection Prevalence : 0.3123   
## Balanced Accuracy : 0.7740   
##   
## 'Positive' Class : Churner   
##

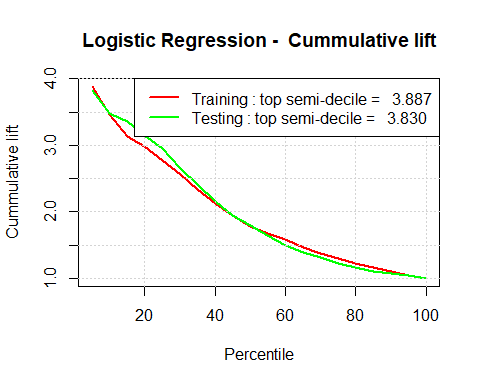
glm\_test\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 111 170  
## NotChurner 30 689  
##   
## Accuracy : 0.8   
## 95% CI : (0.7738, 0.8244)  
## No Information Rate : 0.859   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.4165   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.7872   
## Specificity : 0.8021   
## Pos Pred Value : 0.3950   
## Neg Pred Value : 0.9583   
## Prevalence : 0.1410   
## Detection Rate : 0.1110   
## Detection Prevalence : 0.2810   
## Balanced Accuracy : 0.7947   
##   
## 'Positive' Class : Churner   
##

# this model has an overall accuracy of  
# - train set: 87.12%  
# -test set: 86.7%  
  
# 6.2.1 - Compute the 'AUC' for the logistic regression model and plot the ROC and lift curves  
  
# 6.2.1.1 - ROC  
glmPred\_train <- prediction(predict(mod.glm, newdata = churn\_train, type = 'prob')[,2], churn\_train$churn)  
glmPerf\_train <- performance(glmPred\_train, 'tpr', 'fpr')  
glmAUC\_train <- as.numeric(performance(glmPred\_train, 'auc')@y.values)  
  
glmPred\_test <- prediction(predict(mod.glm, newdata = churn\_test, type = 'prob')[,2], churn\_test$churn)  
glmPerf\_test <- performance(glmPred\_test, 'tpr', 'fpr')  
glmAUC\_test <- as.numeric(performance(glmPred\_test, 'auc')@y.values)  
  
plot(glmPerf\_train, col = 'red', lwd = 2, main = 'Logistic Regression model-Train and Test ROC',   
 xlab = "False positive Rate", ylab = "True positive Rate")  
plot(glmPerf\_test, col = 'green', lwd = 2, add = T)  
legend\_names <- c('Training', 'Testing')  
legend\_auc <- c(glmAUC\_train, glmAUC\_test)  
grid()  
legend('bottomright', paste0(legend\_names, ' : auc = ', sprintf('%01.03f', legend\_auc)), lwd = 2, col = c('red', 'green'))



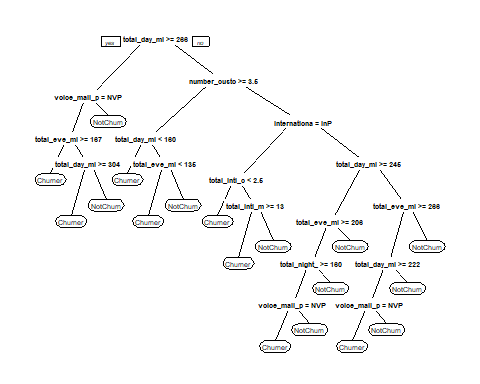
#6.2.1.2  
#Cumulative Lift  
#  
#Create a detaframe for semi-decile  
glm\_lift <- data.frame(percentile = as.integer(seq(5,100,5)))  
nr <- nrow(glm\_lift)  
qs <- predict(mod.glm, newdata = churn\_train, type = "prob")  
#write.csv(file="qstrain", x=qs)  
# add the churn column  
qs$churn <- churn\_train$churn  
#order the prediction dataframe by the Churner column in descending order  
qs <-qs[order(-qs$Churner),]  
  
#calculate the pure guess for the training set  
pri <- sum(churn\_train$churn == "Churner") / nrow(churn\_train)  
# calculate the lift  
current\_lift <- as.numeric(rep(1.0, times = nr)) # start from 1  
for (i in 1:(nr-1)){  
 rows <- as.integer(glm\_lift$percentile[i] \* nrow(qs) / 100) #the bins  
 post <- sum(qs$churn[1:rows] == "Churner") / rows # caclulate the probability of churner in each bin  
 current\_lift[i] <- post/pri # the current lift  
  
}  
glm\_lift$Train <- current\_lift # the lift for the training set  
  
# Repeat the steps for the test set  
qs <- predict(mod.glm, newdata = churn\_test, type = "prob")  
qs$churn <- churn\_test$churn  
qs <- qs[order(-qs$Churner),]  
  
pri <- sum(churn\_test$churn == "Churner") / nrow(churn\_test)  
  
current\_lift <- as.numeric(rep(1.0, times = nr)) # start from 1  
for (i in 1:(nr-1)){  
 rows <- as.integer(glm\_lift$percentile[i] \* nrow(qs) / 100) #the bins  
 post <- sum(qs$churn[1:rows] == "Churner") / rows # caclulate the probability of churner in each bin  
 current\_lift[i] <- post/pri # the current lift  
  
}  
glm\_lift$Test <- current\_lift   
  
#Plot the cumulative lift for train and test sets  
plot(glm\_lift$percentile,glm\_lift$Train, col = "red", type = "l", lwd = 2, main = "Logistic Regression - Cummulative lift", xlab = "Percentile", ylab = "Cummulative lift")  
# add the cummulative lift for the test set  
lines(glm\_lift$percentile, glm\_lift$Test, col = "green", lwd = 2)  
grid() # add the grid  
legend("topright", paste0(legend\_names, " : top semi-decile = ", sprintf("%0.03f", glm\_lift[1,2:3])), lwd = 2, col = c("red", "green"))



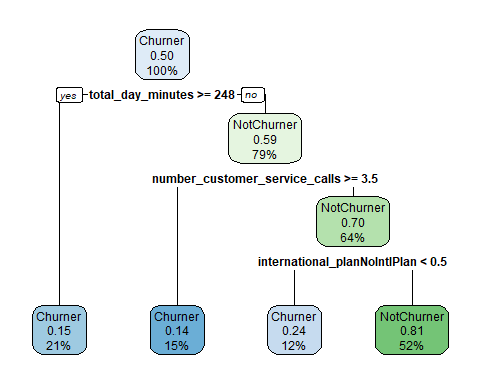
#Conclusion: The Logistic Regression model performs more than 4 times better than the pure guess for the top 5% of the predictions

# 7. Decision Tree Model

#7.1 Train the decision tree model on the training set  
mod.cart <- train(churn ~ ., data = churn\_train, method = "rpart", weights = train\_weights, trControl = fitControl, metric = "ROC")  
  
#print simple model  
tree <- rpart(churn ~.,method='class',data = churn\_train)  
prp(tree)



#Plot the tree  
  
rpart.plot(mod.cart$finalModel)



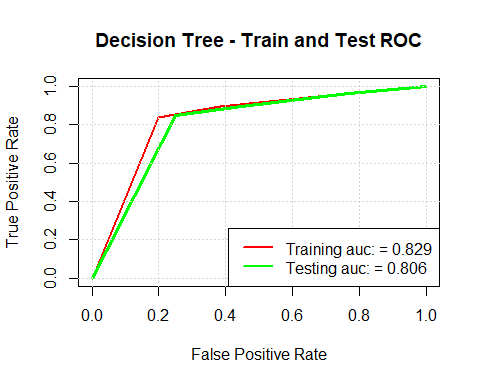
# 7.2 Predict and calculate the confusion matrix on train and test sets  
cart\_train\_cm <- confusionMatrix(predict(mod.cart, newdata = churn\_train), churn\_train$churn)  
cart\_train\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 454 555  
## NotChurner 112 2879  
##   
## Accuracy : 0.8332   
## 95% CI : (0.8213, 0.8447)  
## No Information Rate : 0.8585   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.4827   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.8021   
## Specificity : 0.8384   
## Pos Pred Value : 0.4500   
## Neg Pred Value : 0.9626   
## Prevalence : 0.1415   
## Detection Rate : 0.1135   
## Detection Prevalence : 0.2522   
## Balanced Accuracy : 0.8203   
##   
## 'Positive' Class : Churner   
##

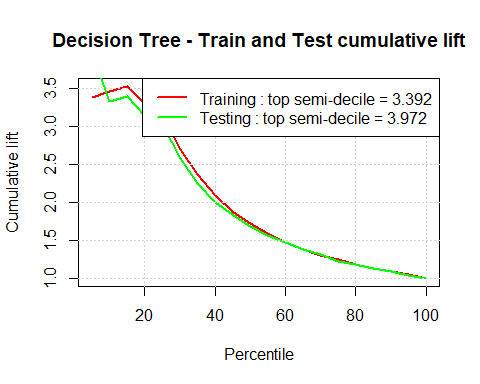
cart\_test\_cm <- confusionMatrix(predict(mod.cart, newdata = churn\_test), churn\_test$churn)  
cart\_test\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 106 128  
## NotChurner 35 731  
##   
## Accuracy : 0.837   
## 95% CI : (0.8126, 0.8594)  
## No Information Rate : 0.859   
## P-Value [Acc > NIR] : 0.9779   
##   
## Kappa : 0.4725   
## Mcnemar's Test P-Value : 5.762e-13   
##   
## Sensitivity : 0.7518   
## Specificity : 0.8510   
## Pos Pred Value : 0.4530   
## Neg Pred Value : 0.9543   
## Prevalence : 0.1410   
## Detection Rate : 0.1060   
## Detection Prevalence : 0.2340   
## Balanced Accuracy : 0.8014   
##   
## 'Positive' Class : Churner   
##

# This model has na overal accuracy of:  
# - training set: 83.32%  
# - testing set: 83.7%  
  
#7.3 Compute the "AUC" for the decision tree model and plot the ROC and lift curves  
#  
#7.3.1 - ROC  
  
cartPred\_train <- prediction(predict(mod.cart, newdata = churn\_train, type = "prob")[,2], churn\_train$churn)  
cartPerf\_train <- performance(cartPred\_train, "tpr", "fpr")  
cartAUC\_train <- as.numeric(performance(cartPred\_train, "auc")@y.values)  
  
cartPred\_test <- prediction(predict(mod.cart, newdata = churn\_test, type = "prob")[,2], churn\_test$churn)  
cartPerf\_test <- performance(cartPred\_test, "tpr", "fpr")  
cartAUC\_test <- as.numeric(performance(cartPred\_test, "auc")@y.values)  
  
#plot the ROC  
legend\_names <- c("Training", "Testing")  
legend\_auc <- c(cartAUC\_train, cartAUC\_test)  
plot(cartPerf\_train, col = "red", lwd =2, main = "Decision Tree - Train and Test ROC", xlab = "False Positive Rate", ylab = "True Positive Rate")  
plot(cartPerf\_test, col = "green", lwd = 3, add = T)  
grid()  
legend("bottomright", paste0(legend\_names, " auc: = ", sprintf("%01.03f", legend\_auc)),lwd = 2, col = c("red", "green"))



#7.3.2 Cumulative lift  
cart\_lift <- data.frame(percentile = as.integer(seq(5,100,5)))  
#cart\_lift  
nr <- nrow(cart\_lift)  
#nr  
qs <- predict(mod.cart, newdata = churn\_train, type = "prob")  
#qs  
qs$churn <- churn\_train$churn  
qs <- qs[order(-qs$Churner),]  
#qs  
pri <- sum(churn\_train$churn == "Churner")/ nrow(churn\_train)  
#pri  
current\_lift <- as.numeric(rep(1.0, times = nr))  
for (i in 1:(nr-1)){  
 rows <- as.integer(cart\_lift$percentile[i] \* nrow(qs) / 100)  
 post <- sum(qs$churn[1:rows] == "Churner") / rows  
 current\_lift[i] <- post / pri  
}  
cart\_lift$Train <- current\_lift  
  
qs <- predict(mod.cart, newdata = churn\_test, type = "prob")  
qs$churn <- churn\_test$churn  
qs <- qs[order(-qs$Churner),]  
pri <- sum(churn\_test$churn == "Churner")/ nrow(churn\_test)  
current\_lift <- as.numeric(rep(1.0, times = nr))  
for (i in 1:(nr-1)){  
 rows <- as.integer(cart\_lift$percentile[i] \* nrow(qs) / 100)  
 post <- sum(qs$churn[1:rows] == "Churner") / rows  
 current\_lift[i] <- post/pri  
}  
cart\_lift$Test <- current\_lift  
  
# Plot the cumulative lift for the decision tree  
plot(cart\_lift$percentile, cart\_lift$Train, col = "red", type = "l", lwd = 2, main = "Decision Tree - Train and Test cumulative lift", xlab = "Percentile", ylab = "Cumulative lift")  
lines(cart\_lift$percentile, cart\_lift$Test, col = "green", lwd = 2)  
grid()  
legend("topright", paste0(legend\_names, " : top semi-decile = ", sprintf("%01.03f", cart\_lift[1, 2:3])), lwd = 2, col = c("red", "green"))



# Conclusion: The decision tree model performs more than 3 times better than the pure guess for the top 5% of the predictions and it has better accuracy than the logistic regression model

# 8. Stochastic Gradient Boosting Model

#8.1 Train the stochastic gradient boosting model on the training set  
  
gbm.grid <- expand.grid(interaction.depth = c(5,9,12),  
 n.trees = c(100,300,500),  
 shrinkage = 0.1,  
 n.minobsinnode=10)  
mod.gbm <- train(churn ~ ., data = churn\_train, method = "gbm", trControl = fitControl,weights = train\_weights, metric = "ROC", tuneGrid = gbm.grid)

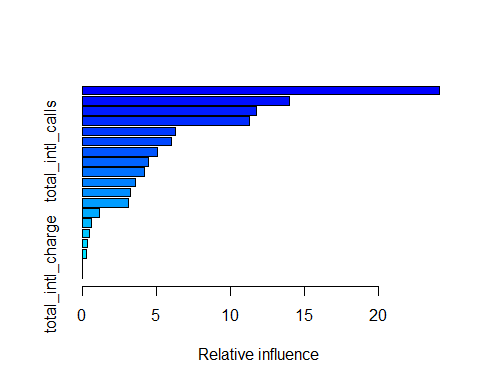
## Warning: package 'plyr' was built under R version 3.3.3

## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2925 nan 0.1000 0.0458  
## 2 1.2113 nan 0.1000 0.0383  
## 3 1.1485 nan 0.1000 0.0284  
## 4 1.0956 nan 0.1000 0.0262  
## 5 1.0507 nan 0.1000 0.0218  
## 6 1.0082 nan 0.1000 0.0196  
## 7 0.9651 nan 0.1000 0.0191  
## 8 0.9326 nan 0.1000 0.0145  
## 9 0.9070 nan 0.1000 0.0116  
## 10 0.8828 nan 0.1000 0.0108  
## 20 0.7399 nan 0.1000 0.0052  
## 40 0.6187 nan 0.1000 0.0002  
## 60 0.5533 nan 0.1000 -0.0007  
## 80 0.4976 nan 0.1000 -0.0001  
## 100 0.4572 nan 0.1000 -0.0010  
## 120 0.4174 nan 0.1000 -0.0010  
## 140 0.3886 nan 0.1000 -0.0003  
## 160 0.3588 nan 0.1000 0.0008  
## 180 0.3329 nan 0.1000 -0.0004  
## 200 0.3100 nan 0.1000 0.0002  
## 220 0.2900 nan 0.1000 -0.0002  
## 240 0.2719 nan 0.1000 -0.0005  
## 260 0.2537 nan 0.1000 -0.0007  
## 280 0.2390 nan 0.1000 -0.0002  
## 300 0.2258 nan 0.1000 -0.0005  
## 320 0.2093 nan 0.1000 -0.0001  
## 340 0.1957 nan 0.1000 0.0001  
## 360 0.1833 nan 0.1000 -0.0004  
## 380 0.1750 nan 0.1000 -0.0003  
## 400 0.1657 nan 0.1000 -0.0004  
## 420 0.1560 nan 0.1000 0.0003  
## 440 0.1490 nan 0.1000 -0.0002  
## 460 0.1407 nan 0.1000 -0.0001  
## 480 0.1326 nan 0.1000 -0.0003  
## 500 0.1254 nan 0.1000 -0.0001  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2745 nan 0.1000 0.0562  
## 2 1.1792 nan 0.1000 0.0465  
## 3 1.0990 nan 0.1000 0.0386  
## 4 1.0342 nan 0.1000 0.0303  
## 5 0.9806 nan 0.1000 0.0256  
## 6 0.9345 nan 0.1000 0.0204  
## 7 0.8886 nan 0.1000 0.0202  
## 8 0.8519 nan 0.1000 0.0158  
## 9 0.8212 nan 0.1000 0.0136  
## 10 0.7918 nan 0.1000 0.0134  
## 20 0.6261 nan 0.1000 0.0023  
## 40 0.4913 nan 0.1000 -0.0003  
## 60 0.4060 nan 0.1000 -0.0004  
## 80 0.3410 nan 0.1000 -0.0006  
## 100 0.2954 nan 0.1000 -0.0005  
## 120 0.2587 nan 0.1000 -0.0002  
## 140 0.2281 nan 0.1000 -0.0005  
## 160 0.2010 nan 0.1000 -0.0001  
## 180 0.1794 nan 0.1000 -0.0001  
## 200 0.1605 nan 0.1000 -0.0001  
## 220 0.1438 nan 0.1000 -0.0002  
## 240 0.1276 nan 0.1000 -0.0003  
## 260 0.1151 nan 0.1000 -0.0002  
## 280 0.1032 nan 0.1000 -0.0002  
## 300 0.0933 nan 0.1000 -0.0001  
## 320 0.0837 nan 0.1000 -0.0002  
## 340 0.0758 nan 0.1000 -0.0001  
## 360 0.0677 nan 0.1000 0.0001  
## 380 0.0620 nan 0.1000 -0.0001  
## 400 0.0561 nan 0.1000 -0.0000  
## 420 0.0509 nan 0.1000 -0.0001  
## 440 0.0465 nan 0.1000 -0.0001  
## 460 0.0427 nan 0.1000 -0.0001  
## 480 0.0388 nan 0.1000 -0.0001  
## 500 0.0352 nan 0.1000 -0.0000  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2672 nan 0.1000 0.0568  
## 2 1.1717 nan 0.1000 0.0473  
## 3 1.0891 nan 0.1000 0.0413  
## 4 1.0176 nan 0.1000 0.0329  
## 5 0.9599 nan 0.1000 0.0249  
## 6 0.9073 nan 0.1000 0.0245  
## 7 0.8639 nan 0.1000 0.0189  
## 8 0.8237 nan 0.1000 0.0176  
## 9 0.7894 nan 0.1000 0.0133  
## 10 0.7635 nan 0.1000 0.0107  
## 20 0.5741 nan 0.1000 0.0028  
## 40 0.4268 nan 0.1000 -0.0001  
## 60 0.3382 nan 0.1000 0.0002  
## 80 0.2830 nan 0.1000 -0.0006  
## 100 0.2360 nan 0.1000 -0.0004  
## 120 0.1969 nan 0.1000 -0.0003  
## 140 0.1653 nan 0.1000 0.0002  
## 160 0.1436 nan 0.1000 -0.0002  
## 180 0.1248 nan 0.1000 -0.0004  
## 200 0.1068 nan 0.1000 0.0002  
## 220 0.0927 nan 0.1000 -0.0004  
## 240 0.0819 nan 0.1000 -0.0001  
## 260 0.0712 nan 0.1000 -0.0002  
## 280 0.0621 nan 0.1000 -0.0002  
## 300 0.0538 nan 0.1000 -0.0001  
## 320 0.0461 nan 0.1000 -0.0000  
## 340 0.0407 nan 0.1000 -0.0001  
## 360 0.0360 nan 0.1000 -0.0001  
## 380 0.0314 nan 0.1000 -0.0001  
## 400 0.0278 nan 0.1000 -0.0001  
## 420 0.0244 nan 0.1000 -0.0001  
## 440 0.0216 nan 0.1000 -0.0001  
## 460 0.0189 nan 0.1000 -0.0001  
## 480 0.0168 nan 0.1000 -0.0001  
## 500 0.0148 nan 0.1000 -0.0001  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2884 nan 0.1000 0.0458  
## 2 1.2107 nan 0.1000 0.0401  
## 3 1.1417 nan 0.1000 0.0317  
## 4 1.0864 nan 0.1000 0.0258  
## 5 1.0391 nan 0.1000 0.0221  
## 6 0.9890 nan 0.1000 0.0236  
## 7 0.9536 nan 0.1000 0.0173  
## 8 0.9172 nan 0.1000 0.0163  
## 9 0.8902 nan 0.1000 0.0127  
## 10 0.8659 nan 0.1000 0.0117  
## 20 0.7157 nan 0.1000 0.0009  
## 40 0.5793 nan 0.1000 0.0010  
## 60 0.5137 nan 0.1000 0.0005  
## 80 0.4600 nan 0.1000 -0.0005  
## 100 0.4122 nan 0.1000 -0.0006  
## 120 0.3738 nan 0.1000 0.0002  
## 140 0.3450 nan 0.1000 -0.0007  
## 160 0.3149 nan 0.1000 -0.0001  
## 180 0.2915 nan 0.1000 0.0009  
## 200 0.2699 nan 0.1000 -0.0005  
## 220 0.2525 nan 0.1000 -0.0001  
## 240 0.2354 nan 0.1000 -0.0003  
## 260 0.2210 nan 0.1000 -0.0006  
## 280 0.2059 nan 0.1000 -0.0002  
## 300 0.1929 nan 0.1000 -0.0002  
## 320 0.1801 nan 0.1000 -0.0001  
## 340 0.1706 nan 0.1000 -0.0004  
## 360 0.1609 nan 0.1000 -0.0003  
## 380 0.1518 nan 0.1000 -0.0002  
## 400 0.1437 nan 0.1000 -0.0004  
## 420 0.1369 nan 0.1000 -0.0001  
## 440 0.1298 nan 0.1000 -0.0001  
## 460 0.1232 nan 0.1000 -0.0003  
## 480 0.1167 nan 0.1000 -0.0000  
## 500 0.1102 nan 0.1000 -0.0002  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2643 nan 0.1000 0.0569  
## 2 1.1649 nan 0.1000 0.0471  
## 3 1.0845 nan 0.1000 0.0376  
## 4 1.0208 nan 0.1000 0.0303  
## 5 0.9603 nan 0.1000 0.0271  
## 6 0.9120 nan 0.1000 0.0212  
## 7 0.8711 nan 0.1000 0.0177  
## 8 0.8340 nan 0.1000 0.0169  
## 9 0.8013 nan 0.1000 0.0124  
## 10 0.7695 nan 0.1000 0.0143  
## 20 0.5973 nan 0.1000 0.0026  
## 40 0.4516 nan 0.1000 -0.0008  
## 60 0.3684 nan 0.1000 -0.0004  
## 80 0.3094 nan 0.1000 0.0005  
## 100 0.2692 nan 0.1000 -0.0006  
## 120 0.2338 nan 0.1000 -0.0004  
## 140 0.2039 nan 0.1000 -0.0002  
## 160 0.1796 nan 0.1000 -0.0002  
## 180 0.1576 nan 0.1000 -0.0004  
## 200 0.1401 nan 0.1000 -0.0002  
## 220 0.1265 nan 0.1000 -0.0004  
## 240 0.1126 nan 0.1000 -0.0003  
## 260 0.1013 nan 0.1000 -0.0002  
## 280 0.0918 nan 0.1000 -0.0001  
## 300 0.0832 nan 0.1000 -0.0002  
## 320 0.0761 nan 0.1000 0.0000  
## 340 0.0688 nan 0.1000 -0.0001  
## 360 0.0613 nan 0.1000 -0.0002  
## 380 0.0549 nan 0.1000 -0.0002  
## 400 0.0498 nan 0.1000 -0.0000  
## 420 0.0449 nan 0.1000 -0.0000  
## 440 0.0407 nan 0.1000 -0.0001  
## 460 0.0369 nan 0.1000 -0.0001  
## 480 0.0337 nan 0.1000 -0.0001  
## 500 0.0309 nan 0.1000 -0.0001  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2707 nan 0.1000 0.0566  
## 2 1.1701 nan 0.1000 0.0488  
## 3 1.0840 nan 0.1000 0.0408  
## 4 1.0091 nan 0.1000 0.0348  
## 5 0.9445 nan 0.1000 0.0301  
## 6 0.8926 nan 0.1000 0.0234  
## 7 0.8439 nan 0.1000 0.0218  
## 8 0.8019 nan 0.1000 0.0167  
## 9 0.7667 nan 0.1000 0.0142  
## 10 0.7345 nan 0.1000 0.0130  
## 20 0.5516 nan 0.1000 0.0013  
## 40 0.3997 nan 0.1000 0.0001  
## 60 0.3119 nan 0.1000 0.0008  
## 80 0.2554 nan 0.1000 -0.0002  
## 100 0.2132 nan 0.1000 -0.0006  
## 120 0.1785 nan 0.1000 -0.0004  
## 140 0.1514 nan 0.1000 -0.0005  
## 160 0.1286 nan 0.1000 -0.0002  
## 180 0.1103 nan 0.1000 0.0000  
## 200 0.0935 nan 0.1000 0.0003  
## 220 0.0812 nan 0.1000 -0.0001  
## 240 0.0704 nan 0.1000 -0.0002  
## 260 0.0609 nan 0.1000 -0.0001  
## 280 0.0532 nan 0.1000 -0.0002  
## 300 0.0468 nan 0.1000 -0.0001  
## 320 0.0403 nan 0.1000 -0.0001  
## 340 0.0351 nan 0.1000 -0.0001  
## 360 0.0311 nan 0.1000 -0.0001  
## 380 0.0273 nan 0.1000 -0.0001  
## 400 0.0242 nan 0.1000 -0.0001  
## 420 0.0211 nan 0.1000 -0.0000  
## 440 0.0186 nan 0.1000 -0.0000  
## 460 0.0164 nan 0.1000 -0.0000  
## 480 0.0144 nan 0.1000 -0.0000  
## 500 0.0128 nan 0.1000 -0.0000  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2970 nan 0.1000 0.0415  
## 2 1.2197 nan 0.1000 0.0361  
## 3 1.1586 nan 0.1000 0.0302  
## 4 1.1060 nan 0.1000 0.0230  
## 5 1.0638 nan 0.1000 0.0209  
## 6 1.0215 nan 0.1000 0.0204  
## 7 0.9889 nan 0.1000 0.0158  
## 8 0.9602 nan 0.1000 0.0124  
## 9 0.9306 nan 0.1000 0.0115  
## 10 0.9060 nan 0.1000 0.0112  
## 20 0.7621 nan 0.1000 0.0066  
## 40 0.6330 nan 0.1000 0.0007  
## 60 0.5597 nan 0.1000 -0.0000  
## 80 0.5089 nan 0.1000 -0.0005  
## 100 0.4598 nan 0.1000 -0.0004  
## 120 0.4188 nan 0.1000 -0.0005  
## 140 0.3880 nan 0.1000 0.0000  
## 160 0.3536 nan 0.1000 -0.0002  
## 180 0.3310 nan 0.1000 -0.0003  
## 200 0.3081 nan 0.1000 -0.0005  
## 220 0.2893 nan 0.1000 -0.0006  
## 240 0.2698 nan 0.1000 -0.0004  
## 260 0.2548 nan 0.1000 -0.0001  
## 280 0.2406 nan 0.1000 -0.0001  
## 300 0.2264 nan 0.1000 -0.0002  
## 320 0.2132 nan 0.1000 -0.0005  
## 340 0.1989 nan 0.1000 -0.0000  
## 360 0.1876 nan 0.1000 0.0001  
## 380 0.1778 nan 0.1000 -0.0002  
## 400 0.1667 nan 0.1000 -0.0002  
## 420 0.1568 nan 0.1000 -0.0002  
## 440 0.1479 nan 0.1000 -0.0003  
## 460 0.1396 nan 0.1000 -0.0000  
## 480 0.1316 nan 0.1000 -0.0002  
## 500 0.1237 nan 0.1000 -0.0003  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2727 nan 0.1000 0.0557  
## 2 1.1777 nan 0.1000 0.0443  
## 3 1.0984 nan 0.1000 0.0376  
## 4 1.0339 nan 0.1000 0.0283  
## 5 0.9774 nan 0.1000 0.0248  
## 6 0.9309 nan 0.1000 0.0215  
## 7 0.8869 nan 0.1000 0.0191  
## 8 0.8524 nan 0.1000 0.0147  
## 9 0.8169 nan 0.1000 0.0145  
## 10 0.7939 nan 0.1000 0.0089  
## 20 0.6312 nan 0.1000 0.0024  
## 40 0.4899 nan 0.1000 -0.0014  
## 60 0.4131 nan 0.1000 -0.0002  
## 80 0.3541 nan 0.1000 -0.0009  
## 100 0.3144 nan 0.1000 -0.0014  
## 120 0.2690 nan 0.1000 -0.0003  
## 140 0.2387 nan 0.1000 -0.0007  
## 160 0.2111 nan 0.1000 -0.0000  
## 180 0.1859 nan 0.1000 -0.0004  
## 200 0.1664 nan 0.1000 -0.0004  
## 220 0.1504 nan 0.1000 -0.0006  
## 240 0.1341 nan 0.1000 -0.0004  
## 260 0.1188 nan 0.1000 -0.0001  
## 280 0.1066 nan 0.1000 -0.0001  
## 300 0.0964 nan 0.1000 -0.0004  
## 320 0.0876 nan 0.1000 -0.0001  
## 340 0.0795 nan 0.1000 -0.0001  
## 360 0.0717 nan 0.1000 -0.0001  
## 380 0.0655 nan 0.1000 -0.0001  
## 400 0.0588 nan 0.1000 0.0001  
## 420 0.0533 nan 0.1000 -0.0001  
## 440 0.0485 nan 0.1000 -0.0002  
## 460 0.0437 nan 0.1000 -0.0001  
## 480 0.0395 nan 0.1000 -0.0002  
## 500 0.0359 nan 0.1000 -0.0000  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2666 nan 0.1000 0.0566  
## 2 1.1685 nan 0.1000 0.0480  
## 3 1.0843 nan 0.1000 0.0401  
## 4 1.0139 nan 0.1000 0.0328  
## 5 0.9559 nan 0.1000 0.0256  
## 6 0.9051 nan 0.1000 0.0238  
## 7 0.8606 nan 0.1000 0.0198  
## 8 0.8222 nan 0.1000 0.0156  
## 9 0.7894 nan 0.1000 0.0140  
## 10 0.7634 nan 0.1000 0.0094  
## 20 0.5810 nan 0.1000 0.0022  
## 40 0.4243 nan 0.1000 -0.0007  
## 60 0.3429 nan 0.1000 -0.0003  
## 80 0.2826 nan 0.1000 -0.0005  
## 100 0.2336 nan 0.1000 -0.0005  
## 120 0.1953 nan 0.1000 -0.0004  
## 140 0.1634 nan 0.1000 0.0000  
## 160 0.1374 nan 0.1000 -0.0007  
## 180 0.1179 nan 0.1000 -0.0001  
## 200 0.1023 nan 0.1000 -0.0002  
## 220 0.0882 nan 0.1000 -0.0001  
## 240 0.0773 nan 0.1000 -0.0002  
## 260 0.0668 nan 0.1000 -0.0001  
## 280 0.0571 nan 0.1000 -0.0001  
## 300 0.0494 nan 0.1000 -0.0001  
## 320 0.0428 nan 0.1000 -0.0001  
## 340 0.0376 nan 0.1000 -0.0000  
## 360 0.0338 nan 0.1000 -0.0001  
## 380 0.0297 nan 0.1000 -0.0001  
## 400 0.0262 nan 0.1000 -0.0001  
## 420 0.0232 nan 0.1000 -0.0001  
## 440 0.0203 nan 0.1000 0.0000  
## 460 0.0178 nan 0.1000 -0.0000  
## 480 0.0156 nan 0.1000 -0.0000  
## 500 0.0139 nan 0.1000 -0.0000  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2950 nan 0.1000 0.0447  
## 2 1.2269 nan 0.1000 0.0353  
## 3 1.1639 nan 0.1000 0.0298  
## 4 1.1126 nan 0.1000 0.0253  
## 5 1.0657 nan 0.1000 0.0222  
## 6 1.0266 nan 0.1000 0.0180  
## 7 0.9925 nan 0.1000 0.0156  
## 8 0.9625 nan 0.1000 0.0144  
## 9 0.9336 nan 0.1000 0.0108  
## 10 0.9101 nan 0.1000 0.0103  
## 20 0.7566 nan 0.1000 0.0030  
## 40 0.6320 nan 0.1000 -0.0009  
## 60 0.5588 nan 0.1000 0.0003  
## 80 0.5074 nan 0.1000 -0.0008  
## 100 0.4646 nan 0.1000 -0.0013  
## 120 0.4243 nan 0.1000 -0.0012  
## 140 0.3920 nan 0.1000 -0.0002  
## 160 0.3674 nan 0.1000 -0.0009  
## 180 0.3448 nan 0.1000 -0.0003  
## 200 0.3216 nan 0.1000 0.0000  
## 220 0.2995 nan 0.1000 0.0005  
## 240 0.2802 nan 0.1000 -0.0002  
## 260 0.2661 nan 0.1000 -0.0007  
## 280 0.2483 nan 0.1000 -0.0004  
## 300 0.2329 nan 0.1000 -0.0001  
## 320 0.2181 nan 0.1000 -0.0000  
## 340 0.2042 nan 0.1000 -0.0005  
## 360 0.1945 nan 0.1000 -0.0002  
## 380 0.1831 nan 0.1000 -0.0003  
## 400 0.1747 nan 0.1000 -0.0004  
## 420 0.1644 nan 0.1000 -0.0002  
## 440 0.1536 nan 0.1000 -0.0002  
## 460 0.1456 nan 0.1000 -0.0002  
## 480 0.1380 nan 0.1000 -0.0000  
## 500 0.1296 nan 0.1000 -0.0002  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2792 nan 0.1000 0.0537  
## 2 1.1925 nan 0.1000 0.0412  
## 3 1.1097 nan 0.1000 0.0370  
## 4 1.0429 nan 0.1000 0.0322  
## 5 0.9896 nan 0.1000 0.0253  
## 6 0.9373 nan 0.1000 0.0228  
## 7 0.8992 nan 0.1000 0.0173  
## 8 0.8645 nan 0.1000 0.0161  
## 9 0.8342 nan 0.1000 0.0126  
## 10 0.8070 nan 0.1000 0.0128  
## 20 0.6445 nan 0.1000 0.0037  
## 40 0.5050 nan 0.1000 0.0003  
## 60 0.4258 nan 0.1000 -0.0005  
## 80 0.3657 nan 0.1000 -0.0003  
## 100 0.3152 nan 0.1000 -0.0008  
## 120 0.2768 nan 0.1000 -0.0001  
## 140 0.2430 nan 0.1000 -0.0003  
## 160 0.2157 nan 0.1000 -0.0001  
## 180 0.1918 nan 0.1000 -0.0004  
## 200 0.1709 nan 0.1000 -0.0001  
## 220 0.1497 nan 0.1000 -0.0002  
## 240 0.1345 nan 0.1000 -0.0003  
## 260 0.1210 nan 0.1000 -0.0001  
## 280 0.1089 nan 0.1000 -0.0005  
## 300 0.0987 nan 0.1000 -0.0001  
## 320 0.0892 nan 0.1000 0.0001  
## 340 0.0805 nan 0.1000 0.0000  
## 360 0.0723 nan 0.1000 -0.0002  
## 380 0.0654 nan 0.1000 -0.0001  
## 400 0.0595 nan 0.1000 -0.0001  
## 420 0.0539 nan 0.1000 -0.0001  
## 440 0.0492 nan 0.1000 -0.0001  
## 460 0.0447 nan 0.1000 -0.0001  
## 480 0.0408 nan 0.1000 0.0001  
## 500 0.0370 nan 0.1000 -0.0001  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2660 nan 0.1000 0.0573  
## 2 1.1734 nan 0.1000 0.0441  
## 3 1.0934 nan 0.1000 0.0379  
## 4 1.0225 nan 0.1000 0.0322  
## 5 0.9605 nan 0.1000 0.0268  
## 6 0.9163 nan 0.1000 0.0183  
## 7 0.8704 nan 0.1000 0.0183  
## 8 0.8358 nan 0.1000 0.0139  
## 9 0.8042 nan 0.1000 0.0120  
## 10 0.7728 nan 0.1000 0.0119  
## 20 0.5931 nan 0.1000 0.0031  
## 40 0.4432 nan 0.1000 0.0000  
## 60 0.3579 nan 0.1000 -0.0006  
## 80 0.3001 nan 0.1000 -0.0004  
## 100 0.2515 nan 0.1000 -0.0005  
## 120 0.2160 nan 0.1000 -0.0003  
## 140 0.1817 nan 0.1000 -0.0001  
## 160 0.1540 nan 0.1000 -0.0003  
## 180 0.1326 nan 0.1000 -0.0001  
## 200 0.1149 nan 0.1000 -0.0004  
## 220 0.0992 nan 0.1000 -0.0000  
## 240 0.0863 nan 0.1000 -0.0003  
## 260 0.0756 nan 0.1000 -0.0001  
## 280 0.0650 nan 0.1000 -0.0000  
## 300 0.0577 nan 0.1000 -0.0001  
## 320 0.0506 nan 0.1000 -0.0001  
## 340 0.0445 nan 0.1000 -0.0001  
## 360 0.0388 nan 0.1000 -0.0001  
## 380 0.0345 nan 0.1000 -0.0001  
## 400 0.0302 nan 0.1000 0.0000  
## 420 0.0264 nan 0.1000 -0.0001  
## 440 0.0233 nan 0.1000 -0.0001  
## 460 0.0203 nan 0.1000 -0.0000  
## 480 0.0180 nan 0.1000 -0.0000  
## 500 0.0158 nan 0.1000 -0.0001  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2654 nan 0.1000 0.0574  
## 2 1.1635 nan 0.1000 0.0480  
## 3 1.0801 nan 0.1000 0.0411  
## 4 1.0130 nan 0.1000 0.0316  
## 5 0.9561 nan 0.1000 0.0257  
## 6 0.9025 nan 0.1000 0.0234  
## 7 0.8607 nan 0.1000 0.0178  
## 8 0.8207 nan 0.1000 0.0187  
## 9 0.7888 nan 0.1000 0.0138  
## 10 0.7567 nan 0.1000 0.0146  
## 20 0.5898 nan 0.1000 0.0014  
## 40 0.4581 nan 0.1000 0.0008  
## 60 0.3738 nan 0.1000 -0.0010  
## 80 0.3126 nan 0.1000 -0.0002  
## 100 0.2667 nan 0.1000 -0.0002  
## 120 0.2281 nan 0.1000 -0.0000  
## 140 0.1971 nan 0.1000 -0.0003  
## 160 0.1715 nan 0.1000 -0.0002  
## 180 0.1518 nan 0.1000 -0.0004  
## 200 0.1339 nan 0.1000 -0.0001  
## 220 0.1201 nan 0.1000 -0.0004  
## 240 0.1063 nan 0.1000 -0.0002  
## 260 0.0943 nan 0.1000 0.0002  
## 280 0.0848 nan 0.1000 -0.0002  
## 300 0.0757 nan 0.1000 0.0000  
## 320 0.0674 nan 0.1000 -0.0001  
## 340 0.0606 nan 0.1000 -0.0001  
## 360 0.0543 nan 0.1000 -0.0001  
## 380 0.0489 nan 0.1000 -0.0001  
## 400 0.0434 nan 0.1000 -0.0001  
## 420 0.0394 nan 0.1000 -0.0000  
## 440 0.0353 nan 0.1000 -0.0002  
## 460 0.0320 nan 0.1000 -0.0000  
## 480 0.0291 nan 0.1000 -0.0000  
## 500 0.0265 nan 0.1000 -0.0001

print(mod.gbm)

## Stochastic Gradient Boosting   
##   
## 4000 samples  
## 18 predictor  
## 2 classes: 'Churner', 'NotChurner'   
##   
## No pre-processing  
## Resampling: Cross-Validated (4 fold)   
## Summary of sample sizes: 3000, 3000, 3000, 3000   
## Resampling results across tuning parameters:  
##   
## interaction.depth n.trees ROC Sens Spec   
## 5 100 0.9241601 0.8533988 0.9327317  
## 5 300 0.9232402 0.8304740 0.9569016  
## 5 500 0.9247725 0.8145790 0.9662191  
## 9 100 0.9312021 0.8463190 0.9539881  
## 9 300 0.9306798 0.8162771 0.9708767  
## 9 500 0.9281736 0.8056887 0.9764091  
## 12 100 0.9255253 0.8304115 0.9609788  
## 12 300 0.9311716 0.8109954 0.9775753  
## 12 500 0.9335940 0.7951254 0.9825266  
##   
## Tuning parameter 'shrinkage' was held constant at a value of 0.1  
##   
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10  
## ROC was used to select the optimal model using the largest value.  
## The final values used for the model were n.trees = 500,  
## interaction.depth = 12, shrinkage = 0.1 and n.minobsinnode = 10.

summary(mod.gbm)



## var rel.inf  
## total\_day\_minutes total\_day\_minutes 24.1358595  
## number\_customer\_service\_calls number\_customer\_service\_calls 14.0179128  
## international\_planNoIntlPlan international\_planNoIntlPlan 11.7499013  
## total\_eve\_minutes total\_eve\_minutes 11.2870248  
## total\_night\_minutes total\_night\_minutes 6.2812090  
## total\_intl\_minutes total\_intl\_minutes 6.0339979  
## total\_intl\_calls total\_intl\_calls 5.0530714  
## account\_length account\_length 4.4514890  
## total\_day\_calls total\_day\_calls 4.1919799  
## voice\_mail\_planVmPlan voice\_mail\_planVmPlan 3.5896412  
## total\_night\_calls total\_night\_calls 3.2251247  
## total\_eve\_calls total\_eve\_calls 3.1329872  
## number\_vmail\_messages number\_vmail\_messages 1.1610213  
## total\_night\_charge total\_night\_charge 0.6034713  
## total\_eve\_charge total\_eve\_charge 0.4556864  
## area\_code415 area\_code415 0.3633418  
## area\_code510 area\_code510 0.2662805  
## total\_day\_charge total\_day\_charge 0.0000000  
## total\_intl\_charge total\_intl\_charge 0.0000000

varImp(mod.gbm,numTrees = 100)

## gbm variable importance  
##   
## Overall  
## total\_day\_minutes 100.0000  
## number\_customer\_service\_calls 61.2127  
## international\_planNoIntlPlan 53.0377  
## total\_eve\_minutes 44.4307  
## total\_night\_minutes 22.1445  
## total\_intl\_minutes 21.4511  
## total\_intl\_calls 21.0195  
## voice\_mail\_planVmPlan 15.7238  
## total\_day\_calls 14.0027  
## account\_length 13.9789  
## total\_eve\_calls 10.2543  
## total\_night\_calls 9.5917  
## number\_vmail\_messages 4.2016  
## total\_night\_charge 1.7537  
## total\_eve\_charge 1.6000  
## area\_code415 1.0994  
## area\_code510 0.7951  
## total\_day\_charge 0.0000  
## total\_intl\_charge 0.0000

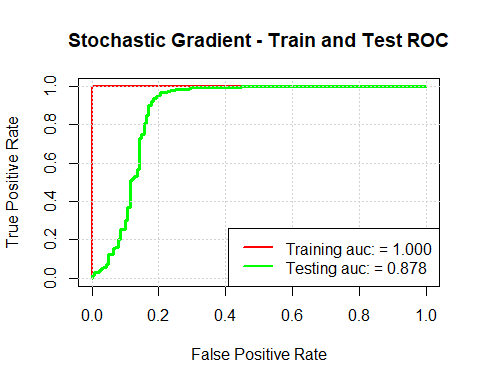
# 8.2 Predict and calculate the confusion matrix on train and test sets  
gbm\_train\_cm <- confusionMatrix(predict(mod.gbm, newdata = churn\_train), churn\_train$churn)  
gbm\_train\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 566 0  
## NotChurner 0 3434  
##   
## Accuracy : 1   
## 95% CI : (0.9991, 1)  
## No Information Rate : 0.8585   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 1   
## Mcnemar's Test P-Value : NA   
##   
## Sensitivity : 1.0000   
## Specificity : 1.0000   
## Pos Pred Value : 1.0000   
## Neg Pred Value : 1.0000   
## Prevalence : 0.1415   
## Detection Rate : 0.1415   
## Detection Prevalence : 0.1415   
## Balanced Accuracy : 1.0000   
##   
## 'Positive' Class : Churner   
##

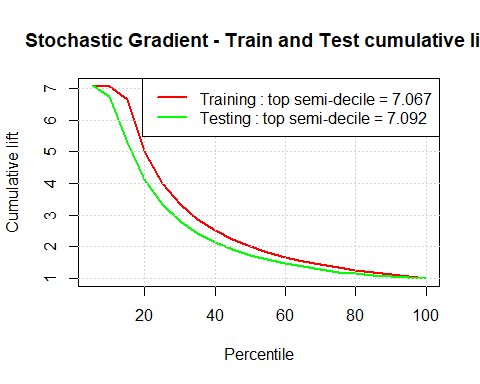
gbm\_test\_cm <- confusionMatrix(predict(mod.gbm, newdata = churn\_test), churn\_test$churn)  
gbm\_test\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 107 19  
## NotChurner 34 840  
##   
## Accuracy : 0.947   
## 95% CI : (0.9312, 0.9601)  
## No Information Rate : 0.859   
## P-Value [Acc > NIR] : < 2e-16   
##   
## Kappa : 0.771   
## Mcnemar's Test P-Value : 0.05447   
##   
## Sensitivity : 0.7589   
## Specificity : 0.9779   
## Pos Pred Value : 0.8492   
## Neg Pred Value : 0.9611   
## Prevalence : 0.1410   
## Detection Rate : 0.1070   
## Detection Prevalence : 0.1260   
## Balanced Accuracy : 0.8684   
##   
## 'Positive' Class : Churner   
##

# This model has an overall accuracy of:  
# - training set: 99.95%  
# - testing set: 94.2%  
  
#8.3 Compute the "AUC" for the Stochastic Gradient Boosting model and plot the ROC and cumulative lift curves  
#  
#8.3.1 - ROC  
  
gbmPred\_train <- prediction(predict(mod.gbm, newdata = churn\_train, type = "prob")[,2], churn\_train$churn)  
gbmPerf\_train <- performance(gbmPred\_train, "tpr", "fpr")  
gbmAUC\_train <- as.numeric(performance(gbmPred\_train, "auc")@y.values)  
  
gbmPred\_test <- prediction(predict(mod.gbm, newdata = churn\_test, type = "prob")[,2], churn\_test$churn)  
gbmPerf\_test <- performance(gbmPred\_test, "tpr", "fpr")  
gbmAUC\_test <- as.numeric(performance(gbmPred\_test, "auc")@y.values)  
#plot the ROC  
legend\_names <- c("Training", "Testing")  
legend\_auc <- c(gbmAUC\_train, gbmAUC\_test)  
plot(gbmPerf\_train, col = "red", lwd =2, main = "Stochastic Gradient - Train and Test ROC", xlab = "False Positive Rate", ylab = "True Positive Rate")  
plot(gbmPerf\_test, col = "green", lwd = 3, add = T)  
grid()  
legend("bottomright", paste0(legend\_names, " auc: = ", sprintf("%01.03f", legend\_auc)),lwd = 2, col = c("red", "green"))



#8.3.2 Cumulative lift  
gbm\_lift <- data.frame(percentile = as.integer(seq(5,100,5)))  
nr <- nrow(gbm\_lift)  
qs <- predict(mod.gbm, newdata = churn\_train, type = "prob")  
#qs  
qs$churn <- churn\_train$churn  
qs <- qs[order(-qs$Churner),]  
#qs  
pri <- sum(churn\_train$churn == "Churner")/ nrow(churn\_train)  
#pri  
current\_lift <- as.numeric(rep(1.0, times = nr))  
for (i in 1:(nr-1)){  
 rows <- as.integer(gbm\_lift$percentile[i] \* nrow(qs) / 100)  
 post <- sum(qs$churn[1:rows] == "Churner") / rows  
 current\_lift[i] <- post / pri  
}  
gbm\_lift$Train <- current\_lift  
  
qs <- predict(mod.gbm, newdata = churn\_test, type = "prob")  
qs$churn <- churn\_test$churn  
qs <- qs[order(-qs$Churner),]  
pri <- sum(churn\_test$churn == "Churner")/ nrow(churn\_test)  
current\_lift <- as.numeric(rep(1.0, times = nr))  
for (i in 1:(nr-1)){  
 rows <- as.integer(gbm\_lift$percentile[i] \* nrow(qs) / 100)  
 post <- sum(qs$churn[1:rows] == "Churner") / rows  
 current\_lift[i] <- post / pri  
}  
gbm\_lift$Test <- current\_lift  
  
# Plot the cumulative lift for the stochastic gradient boosting  
plot(gbm\_lift$percentile, gbm\_lift$Train, col = "red", type = "l", lwd = 2, main = "Stochastic Gradient - Train and Test cumulative lift", xlab = "Percentile", ylab = "Cumulative lift")  
lines(cart\_lift$percentile, gbm\_lift$Test, col = "green", lwd = 2)  
grid()  
legend("topright", paste0(legend\_names, " : top semi-decile = ", sprintf("%01.03f", gbm\_lift[1, 2:3])), lwd = 2, col = c("red", "green"))



# Conclusion: The Stochastic Gradient Boosting model performs more than 7 times better than the pure guess for the top 5% of the predictions and it has a better accuracy than both the logistic regression model and decision tree model

# 9. Support Vector Machines (with a non-linear kernel)

#9.1 Train the support vector machine model on the training set  
  
mod.svm <- train(churn ~ ., data = churn\_train, method = "svmRadial", weights = train\_weights,trControl = fitControl, metric = 'ROC')  
print(mod.svm)

## Support Vector Machines with Radial Basis Function Kernel   
##   
## 4000 samples  
## 18 predictor  
## 2 classes: 'Churner', 'NotChurner'   
##   
## No pre-processing  
## Resampling: Cross-Validated (4 fold)   
## Summary of sample sizes: 2999, 3000, 3001, 3000   
## Resampling results across tuning parameters:  
##   
## C ROC Sens Spec   
## 0.25 0.8015091 0.03710918 0.9909742  
## 0.50 0.8015214 0.03886974 0.9906841  
## 1.00 0.8015379 0.04240336 0.9906841  
##   
## Tuning parameter 'sigma' was held constant at a value of 0.0334194  
## ROC was used to select the optimal model using the largest value.  
## The final values used for the model were sigma = 0.0334194 and C = 1.

summary(mod.svm)

## Length Class Mode   
## 1 ksvm S4

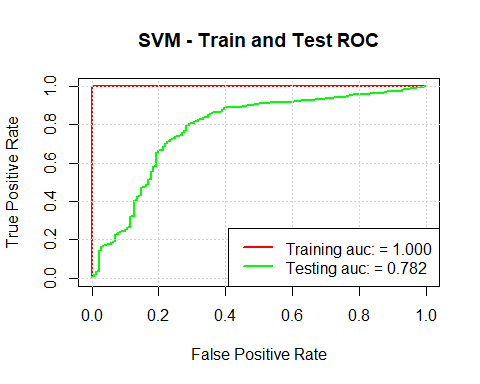
# 9.2 Predict and calculate the confusion matrix on train and test sets  
svm\_train\_cm <- confusionMatrix(predict(mod.svm, newdata = churn\_train), churn\_train$churn)  
svm\_test\_cm <- confusionMatrix(predict(mod.svm, newdata = churn\_test), churn\_test$churn)  
svm\_train\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 566 0  
## NotChurner 0 3434  
##   
## Accuracy : 1   
## 95% CI : (0.9991, 1)  
## No Information Rate : 0.8585   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 1   
## Mcnemar's Test P-Value : NA   
##   
## Sensitivity : 1.0000   
## Specificity : 1.0000   
## Pos Pred Value : 1.0000   
## Neg Pred Value : 1.0000   
## Prevalence : 0.1415   
## Detection Rate : 0.1415   
## Detection Prevalence : 0.1415   
## Balanced Accuracy : 1.0000   
##   
## 'Positive' Class : Churner   
##

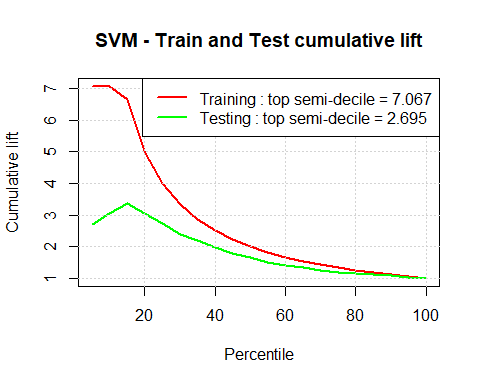
svm\_test\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 5 8  
## NotChurner 136 851  
##   
## Accuracy : 0.856   
## 95% CI : (0.8327, 0.8772)  
## No Information Rate : 0.859   
## P-Value [Acc > NIR] : 0.6285   
##   
## Kappa : 0.0421   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.03546   
## Specificity : 0.99069   
## Pos Pred Value : 0.38462   
## Neg Pred Value : 0.86221   
## Prevalence : 0.14100   
## Detection Rate : 0.00500   
## Detection Prevalence : 0.01300   
## Balanced Accuracy : 0.51307   
##   
## 'Positive' Class : Churner   
##

# This model has an overall accuracy of:  
# - training set: 100%  
# - testing set: 85.6%  
  
#9.3 Compute the "AUC" for the svm model and plot the ROC and cumulative lift curves  
#  
#9.3.1 - ROC  
  
svmPred\_train <- prediction(predict(mod.svm, newdata = churn\_train, type = "prob")[,2], churn\_train$churn)  
svmPerf\_train <- performance(svmPred\_train, "tpr", "fpr")  
svmAUC\_train <- as.numeric(performance(svmPred\_train, "auc")@y.values)  
  
svmPred\_test <- prediction(predict(mod.svm, newdata = churn\_test, type = "prob")[,2], churn\_test$churn)  
svmPerf\_test <- performance(svmPred\_test, "tpr", "fpr")  
svmAUC\_test <- as.numeric(performance(svmPred\_test, "auc")@y.values)  
  
#plot the ROC  
legend\_names <- c("Training", "Testing")  
legend\_auc <- c(svmAUC\_train, svmAUC\_test)  
plot(svmPerf\_train, col = "red", lwd =2, main = "SVM - Train and Test ROC", xlab = "False Positive Rate", ylab = "True Positive Rate")  
plot(svmPerf\_test, col = "green", lwd = 2, add = T)  
grid()  
legend("bottomright", paste0(legend\_names, " auc: = ", sprintf("%01.03f", legend\_auc)),lwd = 2, col = c("red", "green"))



#9.3.2 Cumulative lift  
svm\_lift <- data.frame(percentile = as.integer(seq(5,100,5)))  
nr <- nrow(svm\_lift)  
qs <- predict(mod.svm, newdata = churn\_train, type = "prob")  
#qs  
qs$churn <- churn\_train$churn  
qs <- qs[order(-qs$Churner),]  
#qs  
pri <- sum(churn\_train$churn == "Churner")/ nrow(churn\_train)  
#pri  
current\_lift <- as.numeric(rep(1.0, times = nr))  
for (i in 1:(nr-1)){  
 rows <- as.integer(svm\_lift$percentile[i] \* nrow(qs) / 100)  
 post <- sum(qs$churn[1:rows] == "Churner") / rows  
 current\_lift[i] <- post / pri  
}  
svm\_lift$Train <- current\_lift  
  
qs <- predict(mod.svm, newdata = churn\_test, type = "prob")  
qs$churn <- churn\_test$churn  
qs <- qs[order(-qs$Churner),]  
pri <- sum(churn\_test$churn == "Churner")/ nrow(churn\_test)  
current\_lift <- as.numeric(rep(1.0, times = nr))  
for (i in 1:(nr-1)){  
 rows <- as.integer(svm\_lift$percentile[i] \* nrow(qs) / 100)  
 post <- sum(qs$churn[1:rows] == "Churner") / rows  
 current\_lift[i] <- post / pri  
}  
svm\_lift$Test <- current\_lift  
  
# Plot the cumulative lift for the support vector machine  
plot(svm\_lift$percentile, svm\_lift$Train, col = "red", type = "l", lwd = 2, main = "SVM - Train and Test cumulative lift", xlab = "Percentile", ylab = "Cumulative lift")  
lines(svm\_lift$percentile, svm\_lift$Test, col = "green", lwd = 2)  
grid()  
legend("topright", paste0(legend\_names, " : top semi-decile = ", sprintf("%01.03f", svm\_lift[1, 2:3])), lwd = 2, col = c("red", "green"))



# Conclusion: We get overfit on the svm model, we try to eliminate some variable that can cause overfit: number of voice mail messages, area code and the charges  
names(churn\_data)

## [1] "account\_length" "area\_code"   
## [3] "international\_plan" "voice\_mail\_plan"   
## [5] "number\_vmail\_messages" "total\_day\_minutes"   
## [7] "total\_day\_calls" "total\_day\_charge"   
## [9] "total\_eve\_minutes" "total\_eve\_calls"   
## [11] "total\_eve\_charge" "total\_night\_minutes"   
## [13] "total\_night\_calls" "total\_night\_charge"   
## [15] "total\_intl\_minutes" "total\_intl\_calls"   
## [17] "total\_intl\_charge" "number\_customer\_service\_calls"  
## [19] "churn"

churn\_data1 <- subset(churn\_data[,-c(2,5,8,11,14,17)])  
names(churn\_data1)

## [1] "account\_length" "international\_plan"   
## [3] "voice\_mail\_plan" "total\_day\_minutes"   
## [5] "total\_day\_calls" "total\_eve\_minutes"   
## [7] "total\_eve\_calls" "total\_night\_minutes"   
## [9] "total\_night\_calls" "total\_intl\_minutes"   
## [11] "total\_intl\_calls" "number\_customer\_service\_calls"  
## [13] "churn"

#take out the calls  
churn\_data2 <-subset(churn\_data1[,-c(5,7,9,11)])  
names(churn\_data2)

## [1] "account\_length" "international\_plan"   
## [3] "voice\_mail\_plan" "total\_day\_minutes"   
## [5] "total\_eve\_minutes" "total\_night\_minutes"   
## [7] "total\_intl\_minutes" "number\_customer\_service\_calls"  
## [9] "churn"

set.seed(3155)  
churn\_data2 <- churn\_data2[sample.int(nrow(churn\_data2)),]  
# 5.2 Split the data set in train and validation  
set.seed(443)  
sample2 <- sample.split(churn\_data2$churn, SplitRatio = .8)  
churn\_train <- subset(churn\_data2, sample2 == T)  
churn\_test <- subset(churn\_data2, sample2 ==F)  
str(churn\_test)

## 'data.frame': 1000 obs. of 9 variables:  
## $ account\_length : int 77 69 82 103 93 167 65 78 101 124 ...  
## $ international\_plan : Factor w/ 2 levels "IntlPlan","NoIntlPlan": 2 2 2 2 2 2 2 2 2 2 ...  
## $ voice\_mail\_plan : Factor w/ 2 levels "NoVmPlan","VmPlan": 1 1 1 2 1 1 1 2 1 1 ...  
## $ total\_day\_minutes : num 168 185 179 139 198 ...  
## $ total\_eve\_minutes : num 202 219 229 142 236 ...  
## $ total\_night\_minutes : num 173 244 188 184 127 ...  
## $ total\_intl\_minutes : num 10 5.5 13.2 11.8 12.6 11.1 7.5 13.2 10.1 7.1 ...  
## $ number\_customer\_service\_calls: int 3 0 1 1 2 4 1 0 3 0 ...  
## $ churn : Factor w/ 2 levels "Churner","NotChurner": 2 2 2 2 2 1 2 2 2 2 ...

str(churn\_train)

## 'data.frame': 4000 obs. of 9 variables:  
## $ account\_length : int 116 83 146 74 90 158 82 81 161 190 ...  
## $ international\_plan : Factor w/ 2 levels "IntlPlan","NoIntlPlan": 2 2 1 2 2 2 2 2 2 2 ...  
## $ voice\_mail\_plan : Factor w/ 2 levels "NoVmPlan","VmPlan": 1 1 1 1 1 1 1 1 1 1 ...  
## $ total\_day\_minutes : num 179 209 133 124 161 ...  
## $ total\_eve\_minutes : num 200 215 263 262 136 ...  
## $ total\_night\_minutes : num 169 248 214 268 209 ...  
## $ total\_intl\_minutes : num 15.8 13 11.2 11.7 9.1 10.4 10.9 13.1 9.3 14.7 ...  
## $ number\_customer\_service\_calls: int 1 1 1 2 2 1 1 0 1 1 ...  
## $ churn : Factor w/ 2 levels "Churner","NotChurner": 2 2 2 2 2 2 2 1 2 2 ...

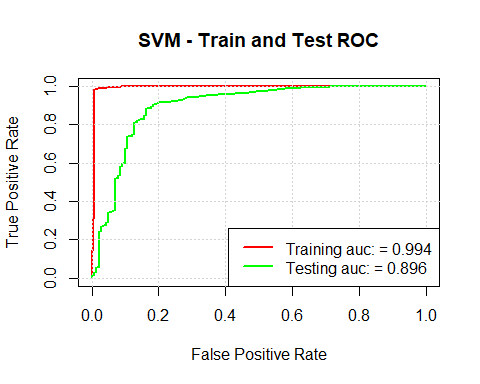
#train the svm model on new train set  
mod.svm1 <- train(churn ~ ., data = churn\_train, method = "svmRadial", weights = train\_weights,trControl = fitControl, metric = 'ROC')  
#Confusion matrix  
svm\_train\_cm1 <- confusionMatrix(predict(mod.svm1, newdata = churn\_train), churn\_train$churn)  
svm\_test\_cm1 <- confusionMatrix(predict(mod.svm1, newdata = churn\_test), churn\_test$churn)  
svm\_train\_cm1

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 511 5  
## NotChurner 55 3429  
##   
## Accuracy : 0.985   
## 95% CI : (0.9807, 0.9885)  
## No Information Rate : 0.8585   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9359   
## Mcnemar's Test P-Value : 2.518e-10   
##   
## Sensitivity : 0.9028   
## Specificity : 0.9985   
## Pos Pred Value : 0.9903   
## Neg Pred Value : 0.9842   
## Prevalence : 0.1415   
## Detection Rate : 0.1278   
## Detection Prevalence : 0.1290   
## Balanced Accuracy : 0.9507   
##   
## 'Positive' Class : Churner   
##

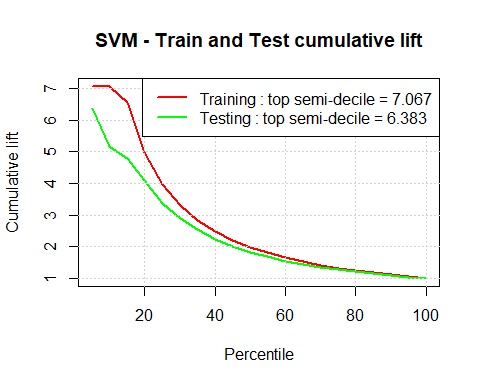
svm\_test\_cm1

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 80 34  
## NotChurner 61 825  
##   
## Accuracy : 0.905   
## 95% CI : (0.8851, 0.9225)  
## No Information Rate : 0.859   
## P-Value [Acc > NIR] : 7.087e-06   
##   
## Kappa : 0.5737   
## Mcnemar's Test P-Value : 0.007641   
##   
## Sensitivity : 0.5674   
## Specificity : 0.9604   
## Pos Pred Value : 0.7018   
## Neg Pred Value : 0.9312   
## Prevalence : 0.1410   
## Detection Rate : 0.0800   
## Detection Prevalence : 0.1140   
## Balanced Accuracy : 0.7639   
##   
## 'Positive' Class : Churner   
##

# This model on the new data has an overall accuracy of:  
# - training set: 98.55%  
# - testing set: 90.1%  
  
#Compute the "AUC" for the svm model and plot the ROC and cumulative lift curves  
#ROC  
  
svmPred\_train1 <- prediction(predict(mod.svm1, newdata = churn\_train, type = "prob")[,2], churn\_train$churn)  
svmPerf\_train1 <- performance(svmPred\_train1, "tpr", "fpr")  
svmAUC\_train1 <- as.numeric(performance(svmPred\_train1, "auc")@y.values)  
  
svmPred\_test1 <- prediction(predict(mod.svm1, newdata = churn\_test, type = "prob")[,2], churn\_test$churn)  
svmPerf\_test1 <- performance(svmPred\_test1, "tpr", "fpr")  
svmAUC\_test1 <- as.numeric(performance(svmPred\_test1, "auc")@y.values)  
  
#plot the ROC  
legend\_names <- c("Training", "Testing")  
legend\_auc <- c(svmAUC\_train1, svmAUC\_test1)  
plot(svmPerf\_train1, col = "red", lwd =2, main = "SVM - Train and Test ROC", xlab = "False Positive Rate", ylab = "True Positive Rate")  
plot(svmPerf\_test1, col = "green", lwd = 2, add = T)  
grid()  
legend("bottomright", paste0(legend\_names, " auc: = ", sprintf("%01.03f", legend\_auc)),lwd = 2, col = c("red", "green"))



#9.3.2 Cumulative lift  
svm\_lift <- data.frame(percentile = as.integer(seq(5,100,5)))  
nr <- nrow(svm\_lift)  
qs <- predict(mod.svm1, newdata = churn\_train, type = "prob")  
#qs  
qs$churn <- churn\_train$churn  
qs <- qs[order(-qs$Churner),]  
#qs  
pri <- sum(churn\_train$churn == "Churner")/ nrow(churn\_train)  
#pri  
current\_lift <- as.numeric(rep(1.0, times = nr))  
for (i in 1:(nr-1)){  
 rows <- as.integer(svm\_lift$percentile[i] \* nrow(qs) / 100)  
 post <- sum(qs$churn[1:rows] == "Churner") / rows  
 current\_lift[i] <- post / pri  
}  
svm\_lift$Train <- current\_lift  
  
qs <- predict(mod.svm1, newdata = churn\_test, type = "prob")  
qs$churn <- churn\_test$churn  
qs <- qs[order(-qs$Churner),]  
pri <- sum(churn\_test$churn == "Churner")/ nrow(churn\_test)  
current\_lift <- as.numeric(rep(1.0, times = nr))  
for (i in 1:(nr-1)){  
 rows <- as.integer(svm\_lift$percentile[i] \* nrow(qs) / 100)  
 post <- sum(qs$churn[1:rows] == "Churner") / rows  
 current\_lift[i] <- post / pri  
}  
svm\_lift$Test <- current\_lift  
  
# Plot the cumulative lift for the support vector machine  
plot(svm\_lift$percentile, svm\_lift$Train, col = "red", type = "l", lwd = 2, main = "SVM - Train and Test cumulative lift", xlab = "Percentile", ylab = "Cumulative lift")  
lines(svm\_lift$percentile, svm\_lift$Test, col = "green", lwd = 2)  
grid()  
legend("topright", paste0(legend\_names, " : top semi-decile = ", sprintf("%01.03f", svm\_lift[1, 2:3])), lwd = 2, col = c("red", "green"))



# by eliminating calls variable we improve the accuracy on the test set, further investigation will be needed

# 10. eXtreme Gradient Boosting model

#10.1 Train the model on the training set  
xgb.grid <- expand.grid(nrounds = c(100,300), max\_depth = c(5,9,12), eta = 0.12, gamma = 0, colsample\_bytree = c(0.8,0.95), min\_child\_weight = 1, subsample = c(0.8,0.95))  
  
mod.xgb <- train(churn ~ ., data = churn\_train, method = "xgbTree", weights = train\_weights, lambda = 1.5, trControl = fitControl, metric = "ROC", tuneGrid = xgb.grid, preProc = c("center", "scale"))  
summary(mod.xgb)

## Length Class Mode   
## handle 1 xgb.Booster.handle externalptr  
## raw 158052 -none- raw   
## niter 1 -none- numeric   
## call 6 -none- call   
## params 9 -none- list   
## callbacks 1 -none- list   
## xNames 8 -none- character   
## problemType 1 -none- character   
## tuneValue 7 data.frame list   
## obsLevels 2 -none- character   
## param 1 -none- list

print(mod.xgb)

## eXtreme Gradient Boosting   
##   
## 4000 samples  
## 8 predictor  
## 2 classes: 'Churner', 'NotChurner'   
##   
## Pre-processing: centered (8), scaled (8)   
## Resampling: Cross-Validated (4 fold)   
## Summary of sample sizes: 3000, 3000, 3001, 2999   
## Resampling results across tuning parameters:  
##   
## max\_depth colsample\_bytree subsample nrounds ROC Sens   
## 5 0.80 0.80 100 0.9010975 0.6465138  
## 5 0.80 0.80 300 0.8949371 0.6412321  
## 5 0.80 0.95 100 0.9024801 0.6553916  
## 5 0.80 0.95 300 0.8953330 0.6535811  
## 5 0.95 0.80 100 0.9015289 0.6465263  
## 5 0.95 0.80 300 0.8936677 0.6518455  
## 5 0.95 0.95 100 0.9030638 0.6535935  
## 5 0.95 0.95 300 0.8948603 0.6500599  
## 9 0.80 0.80 100 0.8991480 0.6411572  
## 9 0.80 0.80 300 0.8974283 0.6429677  
## 9 0.80 0.95 100 0.9002326 0.6465138  
## 9 0.80 0.95 300 0.8988707 0.6536060  
## 9 0.95 0.80 100 0.8983012 0.6623839  
## 9 0.95 0.80 300 0.8965265 0.6624338  
## 9 0.95 0.95 100 0.8997144 0.6659674  
## 9 0.95 0.95 300 0.8984171 0.6553666  
## 12 0.80 0.80 100 0.8983244 0.6553167  
## 12 0.80 0.80 300 0.8984604 0.6411947  
## 12 0.80 0.95 100 0.8966483 0.6358880  
## 12 0.80 0.95 300 0.8962035 0.6394341  
## 12 0.95 0.80 100 0.9005768 0.6624089  
## 12 0.95 0.80 300 0.8999481 0.6642069  
## 12 0.95 0.95 100 0.9012146 0.6606483  
## 12 0.95 0.95 300 0.9006792 0.6624213  
## Spec   
## 0.9854404  
## 0.9836938  
## 0.9854407  
## 0.9848587  
## 0.9854404  
## 0.9810728  
## 0.9866059  
## 0.9839845  
## 0.9877707  
## 0.9866059  
## 0.9874793  
## 0.9857331  
## 0.9851500  
## 0.9836952  
## 0.9877711  
## 0.9845686  
## 0.9860245  
## 0.9848597  
## 0.9857324  
## 0.9845680  
## 0.9863166  
## 0.9831131  
## 0.9871890  
## 0.9831121  
##   
## Tuning parameter 'eta' was held constant at a value of 0.12  
##   
## Tuning parameter 'gamma' was held constant at a value of 0  
##   
## Tuning parameter 'min\_child\_weight' was held constant at a value of 1  
## ROC was used to select the optimal model using the largest value.  
## The final values used for the model were nrounds = 100, max\_depth = 5,  
## eta = 0.12, gamma = 0, colsample\_bytree = 0.95, min\_child\_weight = 1  
## and subsample = 0.95.

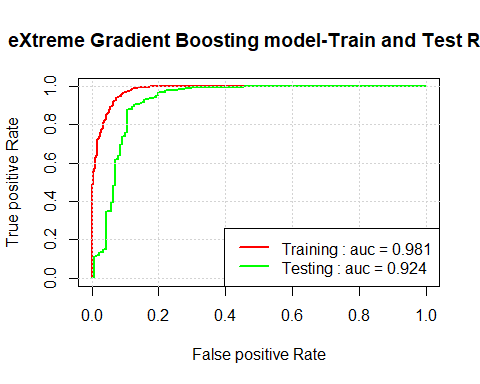
# 10.2 Predict and calculate the confusion matrix on train and test sets  
xgb\_train\_cm <- confusionMatrix(predict(mod.xgb, newdata = churn\_train), churn\_train$churn)  
xgb\_test\_cm <- confusionMatrix(predict(mod.xgb, newdata = churn\_test), churn\_test$churn)  
xgb\_train\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 453 2  
## NotChurner 113 3432  
##   
## Accuracy : 0.9712   
## 95% CI : (0.9656, 0.9762)  
## No Information Rate : 0.8585   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8711   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 0.8004   
## Specificity : 0.9994   
## Pos Pred Value : 0.9956   
## Neg Pred Value : 0.9681   
## Prevalence : 0.1415   
## Detection Rate : 0.1133   
## Detection Prevalence : 0.1138   
## Balanced Accuracy : 0.8999   
##   
## 'Positive' Class : Churner   
##

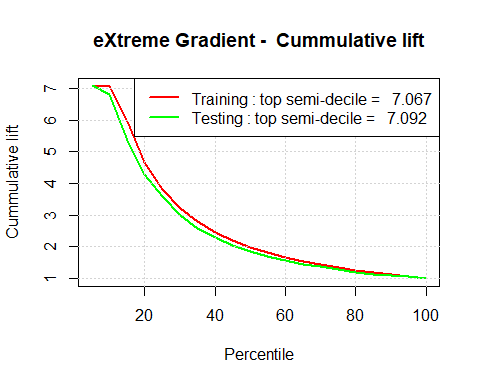
xgb\_test\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 103 14  
## NotChurner 38 845  
##   
## Accuracy : 0.948   
## 95% CI : (0.9324, 0.9609)  
## No Information Rate : 0.859   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.7689   
## Mcnemar's Test P-Value : 0.001425   
##   
## Sensitivity : 0.7305   
## Specificity : 0.9837   
## Pos Pred Value : 0.8803   
## Neg Pred Value : 0.9570   
## Prevalence : 0.1410   
## Detection Rate : 0.1030   
## Detection Prevalence : 0.1170   
## Balanced Accuracy : 0.8571   
##   
## 'Positive' Class : Churner   
##

# This model has an overall accuracy of:  
# - training set: 97.12%  
# - testing set: 94.8%  
  
# 10.3 - Compute the 'AUC' for the logistic regression model and plot the ROC and lift curves  
  
# 10.3.1 - ROC  
xgbPred\_train <- prediction(predict(mod.xgb, newdata = churn\_train, type = 'prob')[,2], churn\_train$churn)  
xgbPerf\_train <- performance(xgbPred\_train, 'tpr', 'fpr')  
xgbAUC\_train <- as.numeric(performance(xgbPred\_train, 'auc')@y.values)  
  
xgbPred\_test <- prediction(predict(mod.xgb, newdata = churn\_test, type = 'prob')[,2], churn\_test$churn)  
xgbPerf\_test <- performance(xgbPred\_test, 'tpr', 'fpr')  
xgbAUC\_test <- as.numeric(performance(xgbPred\_test, 'auc')@y.values)  
  
plot(xgbPerf\_train, col = 'red', lwd = 2, main = 'eXtreme Gradient Boosting model-Train and Test ROC',   
 xlab = "False positive Rate", ylab = "True positive Rate")  
plot(xgbPerf\_test, col = 'green', lwd = 2, add = T)  
legend\_names <- c('Training', 'Testing')  
legend\_auc <- c(xgbAUC\_train, xgbAUC\_test)  
grid()  
legend('bottomright', paste0(legend\_names, ' : auc = ', sprintf('%01.03f', legend\_auc)), lwd = 2, col = c('red', 'green'))



#10.3.2  
#Cumulative Lift  
#  
#Create a detaframe for semi-decile  
xgb\_lift <- data.frame(percentile = as.integer(seq(5,100,5)))  
nr <- nrow(xgb\_lift)  
qs <- predict(mod.xgb, newdata = churn\_train, type = "prob")  
  
qs$churn <- churn\_train$churn  
#order the prediction dataframe by the Churner column in descending order  
qs <-qs[order(-qs$Churner),]  
  
#calculate the pure guess for the training set  
pri <- sum(churn\_train$churn == "Churner") / nrow(churn\_train)  
# calculate the lift  
current\_lift <- as.numeric(rep(1.0, times = nr)) # start from 1  
for (i in 1:(nr-1)){  
 rows <- as.integer(xgb\_lift$percentile[i] \* nrow(qs) / 100) #the bins  
 post <- sum(qs$churn[1:rows] == "Churner") / rows # caclulate the probability of churner in each bin  
 current\_lift[i] <- post/pri # the current lift  
}  
xgb\_lift$Train <- current\_lift # the lift for the training set  
  
# Repeat the steps for the test set  
qs <- predict(mod.xgb, newdata = churn\_test, type = "prob")  
qs$churn <- churn\_test$churn  
qs <- qs[order(-qs$Churner),]  
  
pri <- sum(churn\_test$churn == "Churner") / nrow(churn\_test)  
  
current\_lift <- as.numeric(rep(1.0, times = nr)) # start from 1  
for (i in 1:(nr-1)){  
 rows <- as.integer(xgb\_lift$percentile[i] \* nrow(qs) / 100) #the bins  
 post <- sum(qs$churn[1:rows] == "Churner") / rows # caclulate the probability of churner in each bin  
 current\_lift[i] <- post/pri # the current lift  
  
}  
xgb\_lift$Test <- current\_lift   
  
#Plot the cumulative lift for train and test sets  
plot(xgb\_lift$percentile,xgb\_lift$Train, col = "red", type = "l", lwd = 2, main = "eXtreme Gradient - Cummulative lift", xlab = "Percentile", ylab = "Cummulative lift")  
# add the cummulative lift for the test set  
lines(xgb\_lift$percentile, xgb\_lift$Test, col = "green", lwd = 2)  
grid() # add the grid  
legend("topright", paste0(legend\_names, " : top semi-decile = ", sprintf("%0.03f", xgb\_lift[1,2:3])), lwd = 2, col = c("red", "green"))



#Conclusion: The eXtreme Gradient Boosting model performs more than 7 times better than the pure guess for the top 5% of the predictions.

# 11. Random Forest Model

#11.1 Train the Random Forest model, compute the confusion matrix  
mod.rf <- train(churn ~ ., data = churn\_train, method = 'rf', weights = train\_weights, trControl = fitControl, metric = 'ROC', prox = T)  
print(mod.rf)

## Random Forest   
##   
## 4000 samples  
## 8 predictor  
## 2 classes: 'Churner', 'NotChurner'   
##   
## No pre-processing  
## Resampling: Cross-Validated (4 fold)   
## Summary of sample sizes: 3001, 3000, 3000, 2999   
## Resampling results across tuning parameters:  
##   
## mtry ROC Sens Spec   
## 2 0.9067702 0.6059335 0.9947576  
## 5 0.9047700 0.6501099 0.9886418  
## 8 0.9003750 0.6624838 0.9863115  
##   
## ROC was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 2.

summary(mod.rf)

## Length Class Mode   
## call 5 -none- call   
## type 1 -none- character  
## predicted 4000 factor numeric   
## err.rate 1500 -none- numeric   
## confusion 6 -none- numeric   
## votes 8000 matrix numeric   
## oob.times 4000 -none- numeric   
## classes 2 -none- character  
## importance 8 -none- numeric   
## importanceSD 0 -none- NULL   
## localImportance 0 -none- NULL   
## proximity 16000000 -none- numeric   
## ntree 1 -none- numeric   
## mtry 1 -none- numeric   
## forest 14 -none- list   
## y 4000 factor numeric   
## test 0 -none- NULL   
## inbag 0 -none- NULL   
## xNames 8 -none- character  
## problemType 1 -none- character  
## tuneValue 1 data.frame list   
## obsLevels 2 -none- character  
## param 1 -none- list

varImp(mod.rf)

## rf variable importance  
##   
## Overall  
## total\_day\_minutes 100.00  
## total\_eve\_minutes 36.50  
## number\_customer\_service\_calls 29.59  
## total\_intl\_minutes 23.95  
## total\_night\_minutes 23.04  
## international\_planNoIntlPlan 15.86  
## account\_length 14.95  
## voice\_mail\_planVmPlan 0.00

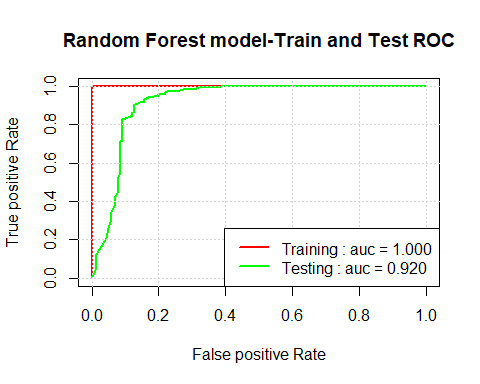
rf\_train\_cm <- confusionMatrix(predict(mod.rf, newdata = churn\_train), churn\_train$churn)  
rf\_test\_cm <- confusionMatrix(predict(mod.rf, newdata = churn\_test), churn\_test$churn)  
rf\_train\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 547 0  
## NotChurner 19 3434  
##   
## Accuracy : 0.9952   
## 95% CI : (0.9926, 0.9971)  
## No Information Rate : 0.8585   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9802   
## Mcnemar's Test P-Value : 3.636e-05   
##   
## Sensitivity : 0.9664   
## Specificity : 1.0000   
## Pos Pred Value : 1.0000   
## Neg Pred Value : 0.9945   
## Prevalence : 0.1415   
## Detection Rate : 0.1368   
## Detection Prevalence : 0.1368   
## Balanced Accuracy : 0.9832   
##   
## 'Positive' Class : Churner   
##

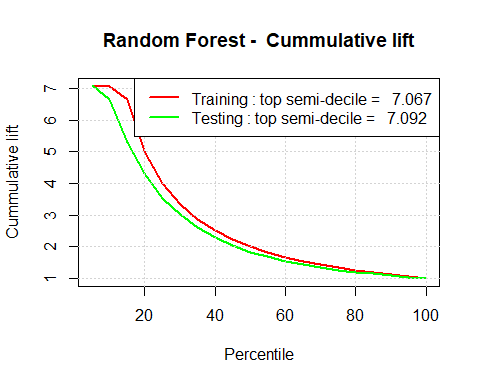
rf\_test\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 89 6  
## NotChurner 52 853  
##   
## Accuracy : 0.942   
## 95% CI : (0.9257, 0.9557)  
## No Information Rate : 0.859   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.7228   
## Mcnemar's Test P-Value : 3.446e-09   
##   
## Sensitivity : 0.6312   
## Specificity : 0.9930   
## Pos Pred Value : 0.9368   
## Neg Pred Value : 0.9425   
## Prevalence : 0.1410   
## Detection Rate : 0.0890   
## Detection Prevalence : 0.0950   
## Balanced Accuracy : 0.8121   
##   
## 'Positive' Class : Churner   
##

# this model has an overall accuracy of  
# - train set: 99.5%  
# -test set: 94.2%  
  
# 11.2 - Compute the 'AUC' for the Random Forest model and plot the ROC and lift curves  
  
# 11.2.1 - ROC  
rfPred\_train <- prediction(predict(mod.rf, newdata = churn\_train, type = 'prob')[,2], churn\_train$churn)  
rfPerf\_train <- performance(rfPred\_train, 'tpr', 'fpr')  
rfAUC\_train <- as.numeric(performance(rfPred\_train, 'auc')@y.values)  
  
rfPred\_test <- prediction(predict(mod.rf, newdata = churn\_test, type = 'prob')[,2], churn\_test$churn)  
rfPerf\_test <- performance(rfPred\_test, 'tpr', 'fpr')  
rfAUC\_test <- as.numeric(performance(rfPred\_test, 'auc')@y.values)  
  
plot(rfPerf\_train, col = 'red', lwd = 2, main = 'Random Forest model-Train and Test ROC',   
 xlab = "False positive Rate", ylab = "True positive Rate")  
plot(rfPerf\_test, col = 'green', lwd = 2, add = T)  
legend\_names <- c('Training', 'Testing')  
legend\_auc <- c(rfAUC\_train, rfAUC\_test)  
grid()  
legend('bottomright', paste0(legend\_names, ' : auc = ', sprintf('%01.03f', legend\_auc)), lwd = 2, col = c('red', 'green'))



#11.2.2  
#Cumulative Lift  
#  
#Create a detaframe for semi-decile  
rf\_lift <- data.frame(percentile = as.integer(seq(5,100,5)))  
nr <- nrow(rf\_lift)  
qs <- predict(mod.rf, newdata = churn\_train, type = "prob")  
  
# add the churn column  
qs$churn <- churn\_train$churn  
#order the prediction dataframe by the Churner column in descending order  
qs <-qs[order(-qs$Churner),]  
  
#calculate the pure guess for the training set  
pri <- sum(churn\_train$churn == "Churner") / nrow(churn\_train)  
# calculate the lift  
current\_lift <- as.numeric(rep(1.0, times = nr)) # start from 1  
for (i in 1:(nr-1)){  
 rows <- as.integer(rf\_lift$percentile[i] \* nrow(qs) / 100) #the bins  
 post <- sum(qs$churn[1:rows] == "Churner") / rows # caclulate the probability of churner in each bin  
 current\_lift[i] <- post/pri # the current lift  
  
}  
rf\_lift$Train <- current\_lift # the lift for the training set  
  
# Repeat the steps for the test set  
qs <- predict(mod.rf, newdata = churn\_test, type = "prob")  
qs$churn <- churn\_test$churn  
qs <- qs[order(-qs$Churner),]  
  
pri <- sum(churn\_test$churn == "Churner") / nrow(churn\_test)  
  
current\_lift <- as.numeric(rep(1.0, times = nr)) # start from 1  
for (i in 1:(nr-1)){  
 rows <- as.integer(rf\_lift$percentile[i] \* nrow(qs) / 100) #the bins  
 post <- sum(qs$churn[1:rows] == "Churner") / rows # caclulate the probability of churner in each bin  
 current\_lift[i] <- post/pri # the current lift  
  
}  
rf\_lift$Test <- current\_lift   
  
#Plot the cumulative lift for train and test sets  
plot(rf\_lift$percentile,rf\_lift$Train, col = "red", type = "l", lwd = 2, main = "Random Forest - Cummulative lift", xlab = "Percentile", ylab = "Cummulative lift")  
# add the cummulative lift for the test set  
lines(rf\_lift$percentile, rf\_lift$Test, col = "green", lwd = 2)  
grid() # add the grid  
legend("topright", paste0(legend\_names, " : top semi-decile = ", sprintf("%0.03f", rf\_lift[1,2:3])), lwd = 2, col = c("red", "green"))



#Conclusion: The Random Forest model performs 7 times better than the pure guess for the top 5% of the predictions  
  
# Beacuse we noticed the overfit we will eliminate some variables  
names(churn\_data)

## [1] "account\_length" "area\_code"   
## [3] "international\_plan" "voice\_mail\_plan"   
## [5] "number\_vmail\_messages" "total\_day\_minutes"   
## [7] "total\_day\_calls" "total\_day\_charge"   
## [9] "total\_eve\_minutes" "total\_eve\_calls"   
## [11] "total\_eve\_charge" "total\_night\_minutes"   
## [13] "total\_night\_calls" "total\_night\_charge"   
## [15] "total\_intl\_minutes" "total\_intl\_calls"   
## [17] "total\_intl\_charge" "number\_customer\_service\_calls"  
## [19] "churn"

churn\_data1 <- subset(churn\_data[,-c(2,5,8,11,14,17)])  
names(churn\_data1)

## [1] "account\_length" "international\_plan"   
## [3] "voice\_mail\_plan" "total\_day\_minutes"   
## [5] "total\_day\_calls" "total\_eve\_minutes"   
## [7] "total\_eve\_calls" "total\_night\_minutes"   
## [9] "total\_night\_calls" "total\_intl\_minutes"   
## [11] "total\_intl\_calls" "number\_customer\_service\_calls"  
## [13] "churn"

#take out the calls  
churn\_data2 <-subset(churn\_data1[,-c(5,7,9,11)])  
names(churn\_data2)

## [1] "account\_length" "international\_plan"   
## [3] "voice\_mail\_plan" "total\_day\_minutes"   
## [5] "total\_eve\_minutes" "total\_night\_minutes"   
## [7] "total\_intl\_minutes" "number\_customer\_service\_calls"  
## [9] "churn"

set.seed(3155)  
churn\_data2 <- churn\_data2[sample.int(nrow(churn\_data2)),]  
# 5.2 Split the data set in train and validation  
set.seed(443)  
sample2 <- sample.split(churn\_data2$churn, SplitRatio = .8)  
churn\_train <- subset(churn\_data2, sample2 == T)  
churn\_test <- subset(churn\_data2, sample2 ==F)  
str(churn\_test)

## 'data.frame': 1000 obs. of 9 variables:  
## $ account\_length : int 77 69 82 103 93 167 65 78 101 124 ...  
## $ international\_plan : Factor w/ 2 levels "IntlPlan","NoIntlPlan": 2 2 2 2 2 2 2 2 2 2 ...  
## $ voice\_mail\_plan : Factor w/ 2 levels "NoVmPlan","VmPlan": 1 1 1 2 1 1 1 2 1 1 ...  
## $ total\_day\_minutes : num 168 185 179 139 198 ...  
## $ total\_eve\_minutes : num 202 219 229 142 236 ...  
## $ total\_night\_minutes : num 173 244 188 184 127 ...  
## $ total\_intl\_minutes : num 10 5.5 13.2 11.8 12.6 11.1 7.5 13.2 10.1 7.1 ...  
## $ number\_customer\_service\_calls: int 3 0 1 1 2 4 1 0 3 0 ...  
## $ churn : Factor w/ 2 levels "Churner","NotChurner": 2 2 2 2 2 1 2 2 2 2 ...

str(churn\_train)

## 'data.frame': 4000 obs. of 9 variables:  
## $ account\_length : int 116 83 146 74 90 158 82 81 161 190 ...  
## $ international\_plan : Factor w/ 2 levels "IntlPlan","NoIntlPlan": 2 2 1 2 2 2 2 2 2 2 ...  
## $ voice\_mail\_plan : Factor w/ 2 levels "NoVmPlan","VmPlan": 1 1 1 1 1 1 1 1 1 1 ...  
## $ total\_day\_minutes : num 179 209 133 124 161 ...  
## $ total\_eve\_minutes : num 200 215 263 262 136 ...  
## $ total\_night\_minutes : num 169 248 214 268 209 ...  
## $ total\_intl\_minutes : num 15.8 13 11.2 11.7 9.1 10.4 10.9 13.1 9.3 14.7 ...  
## $ number\_customer\_service\_calls: int 1 1 1 2 2 1 1 0 1 1 ...  
## $ churn : Factor w/ 2 levels "Churner","NotChurner": 2 2 2 2 2 2 2 1 2 2 ...

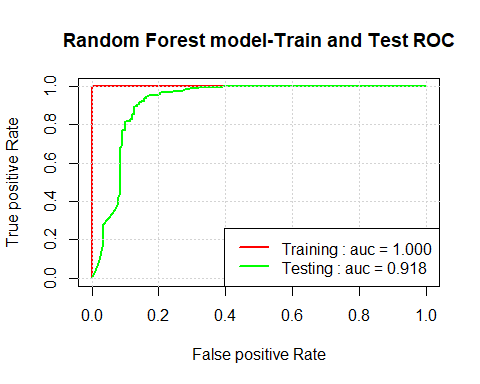
#And we train again the model on the new data  
mod.rf <- train(churn ~ ., data = churn\_train, method = 'rf', weights = train\_weights, trControl = fitControl, metric = 'ROC', prox = T)  
rf\_train\_cm <- confusionMatrix(predict(mod.rf, newdata = churn\_train), churn\_train$churn)  
rf\_test\_cm <- confusionMatrix(predict(mod.rf, newdata = churn\_test), churn\_test$churn)  
rf\_train\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 557 0  
## NotChurner 9 3434  
##   
## Accuracy : 0.9978   
## 95% CI : (0.9957, 0.999)  
## No Information Rate : 0.8585   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9907   
## Mcnemar's Test P-Value : 0.007661   
##   
## Sensitivity : 0.9841   
## Specificity : 1.0000   
## Pos Pred Value : 1.0000   
## Neg Pred Value : 0.9974   
## Prevalence : 0.1415   
## Detection Rate : 0.1393   
## Detection Prevalence : 0.1393   
## Balanced Accuracy : 0.9920   
##   
## 'Positive' Class : Churner   
##

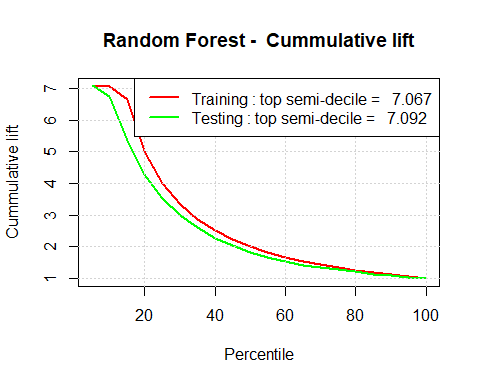
rf\_test\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 89 4  
## NotChurner 52 855  
##   
## Accuracy : 0.944   
## 95% CI : (0.9279, 0.9574)  
## No Information Rate : 0.859   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.7305   
## Mcnemar's Test P-Value : 3.372e-10   
##   
## Sensitivity : 0.6312   
## Specificity : 0.9953   
## Pos Pred Value : 0.9570   
## Neg Pred Value : 0.9427   
## Prevalence : 0.1410   
## Detection Rate : 0.0890   
## Detection Prevalence : 0.0930   
## Balanced Accuracy : 0.8133   
##   
## 'Positive' Class : Churner   
##

# this model has an overall accuracy of  
# - train set: 99.3%  
# -test set: 93.6%  
  
# Compute the 'AUC' for the Random Forest model and plot the ROC and lift curves  
  
# ROC  
rfPred\_train <- prediction(predict(mod.rf, newdata = churn\_train, type = 'prob')[,2], churn\_train$churn)  
rfPerf\_train <- performance(rfPred\_train, 'tpr', 'fpr')  
rfAUC\_train <- as.numeric(performance(rfPred\_train, 'auc')@y.values)  
  
rfPred\_test <- prediction(predict(mod.rf, newdata = churn\_test, type = 'prob')[,2], churn\_test$churn)  
rfPerf\_test <- performance(rfPred\_test, 'tpr', 'fpr')  
rfAUC\_test <- as.numeric(performance(rfPred\_test, 'auc')@y.values)  
  
plot(rfPerf\_train, col = 'red', lwd = 2, main = 'Random Forest model-Train and Test ROC',   
 xlab = "False positive Rate", ylab = "True positive Rate")  
plot(rfPerf\_test, col = 'green', lwd = 2, add = T)  
legend\_names <- c('Training', 'Testing')  
legend\_auc <- c(rfAUC\_train, rfAUC\_test)  
grid()  
legend('bottomright', paste0(legend\_names, ' : auc = ', sprintf('%01.03f', legend\_auc)), lwd = 2, col = c('red', 'green'))



#Cumulative lift  
rf\_lift <- data.frame(percentile = as.integer(seq(5,100,5)))  
nr <- nrow(rf\_lift)  
qs <- predict(mod.rf, newdata = churn\_train, type = "prob")  
  
# add the churn column  
qs$churn <- churn\_train$churn  
#order the prediction dataframe by the Churner column in descending order  
qs <-qs[order(-qs$Churner),]  
  
#calculate the pure guess for the training set  
pri <- sum(churn\_train$churn == "Churner") / nrow(churn\_train)  
# calculate the lift  
current\_lift <- as.numeric(rep(1.0, times = nr)) # start from 1  
for (i in 1:(nr-1)){  
 rows <- as.integer(rf\_lift$percentile[i] \* nrow(qs) / 100) #the bins  
 post <- sum(qs$churn[1:rows] == "Churner") / rows # caclulate the probability of churner in each bin  
 current\_lift[i] <- post/pri # the current lift  
  
}  
rf\_lift$Train <- current\_lift # the lift for the training set  
  
# Repeat the steps for the test set  
qs <- predict(mod.rf, newdata = churn\_test, type = "prob")  
qs$churn <- churn\_test$churn  
qs <- qs[order(-qs$Churner),]  
  
pri <- sum(churn\_test$churn == "Churner") / nrow(churn\_test)  
  
current\_lift <- as.numeric(rep(1.0, times = nr)) # start from 1  
for (i in 1:(nr-1)){  
 rows <- as.integer(rf\_lift$percentile[i] \* nrow(qs) / 100) #the bins  
 post <- sum(qs$churn[1:rows] == "Churner") / rows # caclulate the probability of churner in each bin  
 current\_lift[i] <- post/pri # the current lift  
  
}  
rf\_lift$Test <- current\_lift   
  
#Plot the cumulative lift for train and test sets  
plot(rf\_lift$percentile,rf\_lift$Train, col = "red", type = "l", lwd = 2, main = "Random Forest - Cummulative lift", xlab = "Percentile", ylab = "Cummulative lift")  
# add the cummulative lift for the test set  
lines(rf\_lift$percentile, rf\_lift$Test, col = "green", lwd = 2)  
grid() # add the grid  
legend("topright", paste0(legend\_names, " : top semi-decile = ", sprintf("%0.03f", rf\_lift[1,2:3])), lwd = 2, col = c("red", "green"))



# 12. Neural Networks

#12.1 Train the Neural Networks model, compute the confusion matrix  
  
nnet.grid <- expand.grid(size = c(10, 30), decay = c(0.1, 0.2))  
nnet.grid

## size decay  
## 1 10 0.1  
## 2 30 0.1  
## 3 10 0.2  
## 4 30 0.2

mod.ann <- train(churn ~ ., data = churn\_train, method = 'nnet', weights = train\_weights, trControl = fitControl, metric = 'ROC', trace = F, tuneGrid = nnet.grid, maxit = 2000)  
print(mod.ann)

## Neural Network   
##   
## 4000 samples  
## 8 predictor  
## 2 classes: 'Churner', 'NotChurner'   
##   
## No pre-processing  
## Resampling: Cross-Validated (4 fold)   
## Summary of sample sizes: 3000, 3000, 3001, 2999   
## Resampling results across tuning parameters:  
##   
## size decay ROC Sens Spec   
## 10 0.1 0.8905156 0.6379108 0.9711674  
## 10 0.2 0.8964714 0.5989412 0.9691281  
## 30 0.1 0.8678040 0.5795250 0.9560264  
## 30 0.2 0.8743646 0.6078314 0.9551530  
##   
## ROC was used to select the optimal model using the largest value.  
## The final values used for the model were size = 10 and decay = 0.2.

summary(mod.ann)

## a 8-10-1 network with 101 weights  
## options were - entropy fitting decay=0.2  
## b->h1 i1->h1 i2->h1 i3->h1 i4->h1 i5->h1 i6->h1 i7->h1 i8->h1   
## -0.24 -0.05 -0.35 0.32 0.30 -0.36 -0.16 0.63 1.34   
## b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2 i7->h2 i8->h2   
## -0.01 -0.46 0.05 0.06 0.09 0.40 0.13 0.19 0.43   
## b->h3 i1->h3 i2->h3 i3->h3 i4->h3 i5->h3 i6->h3 i7->h3 i8->h3   
## -0.14 -0.04 1.90 -2.74 0.22 -0.33 0.04 0.11 0.00   
## b->h4 i1->h4 i2->h4 i3->h4 i4->h4 i5->h4 i6->h4 i7->h4 i8->h4   
## -0.20 0.88 0.67 -0.13 0.57 -0.91 -0.11 1.40 0.91   
## b->h5 i1->h5 i2->h5 i3->h5 i4->h5 i5->h5 i6->h5 i7->h5 i8->h5   
## -0.26 -0.01 -0.55 -4.21 0.05 0.02 -0.03 0.15 0.77   
## b->h6 i1->h6 i2->h6 i3->h6 i4->h6 i5->h6 i6->h6 i7->h6 i8->h6   
## -0.13 -0.01 0.64 0.62 0.76 -1.73 0.50 0.90 -0.93   
## b->h7 i1->h7 i2->h7 i3->h7 i4->h7 i5->h7 i6->h7 i7->h7 i8->h7   
## 0.03 0.07 -1.74 0.23 -0.04 0.09 -0.04 0.67 0.60   
## b->h8 i1->h8 i2->h8 i3->h8 i4->h8 i5->h8 i6->h8 i7->h8 i8->h8   
## -2.37 0.00 4.41 -0.71 0.02 0.00 0.00 -0.10 -1.17   
## b->h9 i1->h9 i2->h9 i3->h9 i4->h9 i5->h9 i6->h9 i7->h9 i8->h9   
## 18.16 0.00 0.08 3.63 -0.04 -0.02 -0.01 -0.01 0.18   
## b->h10 i1->h10 i2->h10 i3->h10 i4->h10 i5->h10 i6->h10 i7->h10 i8->h10   
## 0.12 0.36 -0.29 -1.16 0.06 -0.10 0.01 0.30 1.36   
## b->o h1->o h2->o h3->o h4->o h5->o h6->o h7->o h8->o h9->o   
## -1.22 -1.44 -1.69 3.25 -0.70 -3.77 -2.20 -2.99 4.73 11.94   
## h10->o   
## -2.56

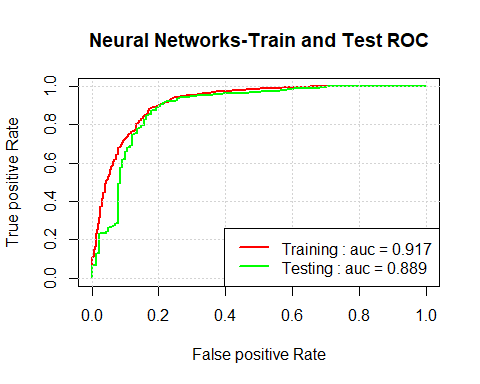
#Confusion matrix  
ann\_train\_cm <- confusionMatrix(predict(mod.ann, newdata = churn\_train), churn\_train$churn)  
ann\_test\_cm <- confusionMatrix(predict(mod.ann, newdata = churn\_test), churn\_test$churn)  
ann\_train\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 349 94  
## NotChurner 217 3340  
##   
## Accuracy : 0.9222   
## 95% CI : (0.9135, 0.9304)  
## No Information Rate : 0.8585   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.648   
## Mcnemar's Test P-Value : 4.581e-12   
##   
## Sensitivity : 0.61661   
## Specificity : 0.97263   
## Pos Pred Value : 0.78781   
## Neg Pred Value : 0.93899   
## Prevalence : 0.14150   
## Detection Rate : 0.08725   
## Detection Prevalence : 0.11075   
## Balanced Accuracy : 0.79462   
##   
## 'Positive' Class : Churner   
##

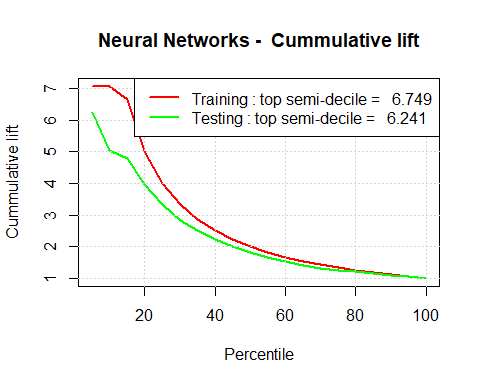
ann\_test\_cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Churner NotChurner  
## Churner 71 27  
## NotChurner 70 832  
##   
## Accuracy : 0.903   
## 95% CI : (0.883, 0.9206)  
## No Information Rate : 0.859   
## P-Value [Acc > NIR] : 1.744e-05   
##   
## Kappa : 0.5411   
## Mcnemar's Test P-Value : 2.004e-05   
##   
## Sensitivity : 0.5035   
## Specificity : 0.9686   
## Pos Pred Value : 0.7245   
## Neg Pred Value : 0.9224   
## Prevalence : 0.1410   
## Detection Rate : 0.0710   
## Detection Prevalence : 0.0980   
## Balanced Accuracy : 0.7361   
##   
## 'Positive' Class : Churner   
##

# this model has an overall accuracy of  
# - train set: 93.7%  
# -test set: 93.5%  
  
# 12.2 - Compute the 'AUC' for the Neural Networks model and plot the ROC and lift curves  
  
# 12.2.1 - ROC  
annPred\_train <- prediction(predict(mod.ann, newdata = churn\_train, type = 'prob')[,2], churn\_train$churn)  
annPerf\_train <- performance(annPred\_train, 'tpr', 'fpr')  
annAUC\_train <- as.numeric(performance(annPred\_train, 'auc')@y.values)  
  
annPred\_test <- prediction(predict(mod.ann, newdata = churn\_test, type = 'prob')[,2], churn\_test$churn)  
annPerf\_test <- performance(annPred\_test, 'tpr', 'fpr')  
annAUC\_test <- as.numeric(performance(annPred\_test, 'auc')@y.values)  
  
plot(annPerf\_train, col = 'red', lwd = 2, main = 'Neural Networks-Train and Test ROC', xlab = "False positive Rate", ylab = "True positive Rate")  
plot(annPerf\_test, col = 'green', lwd = 2, add = T)  
legend\_names <- c('Training', 'Testing')  
legend\_auc <- c(annAUC\_train, annAUC\_test)  
grid()  
legend('bottomright', paste0(legend\_names, ' : auc = ', sprintf('%01.03f', legend\_auc)), lwd = 2, col = c('red', 'green'))



#12.2.2  
#Cumulative Lift  
#  
#Create a detaframe for semi-decile  
ann\_lift <- data.frame(percentile = as.integer(seq(5,100,5)))  
nr <- nrow(ann\_lift)  
qs <- predict(mod.ann, newdata = churn\_train, type = "prob")  
#write.csv(file="qstrain", x=qs)  
# add the churn column  
qs$churn <- churn\_train$churn  
#order the prediction dataframe by the Churner column in descending order  
qs <-qs[order(-qs$Churner),]  
  
#calculate the pure guess for the training set  
pri <- sum(churn\_train$churn == "Churner") / nrow(churn\_train)  
# calculate the lift  
current\_lift <- as.numeric(rep(1.0, times = nr)) # start from 1  
for (i in 1:(nr-1)){  
 rows <- as.integer(ann\_lift$percentile[i] \* nrow(qs) / 100) #the bins  
 post <- sum(qs$churn[1:rows] == "Churner") / rows # caclulate the probability of churner in each bin  
 current\_lift[i] <- post/pri # the current lift  
  
}  
ann\_lift$Train <- current\_lift # the lift for the training set  
  
# Repeat the steps for the test set  
qs <- predict(mod.ann, newdata = churn\_test, type = "prob")  
qs$churn <- churn\_test$churn  
qs <- qs[order(-qs$Churner),]  
  
pri <- sum(churn\_test$churn == "Churner") / nrow(churn\_test)  
  
current\_lift <- as.numeric(rep(1.0, times = nr)) # start from 1  
for (i in 1:(nr-1)){  
 rows <- as.integer(ann\_lift$percentile[i] \* nrow(qs) / 100) #the bins  
 post <- sum(qs$churn[1:rows] == "Churner") / rows # caclulate the probability of churner in each bin  
 current\_lift[i] <- post/pri # the current lift  
  
}  
ann\_lift$Test <- current\_lift   
  
#Plot the cumulative lift for train and test sets  
plot(ann\_lift$percentile,rf\_lift$Train, col = "red", type = "l", lwd = 2, main = "Neural Networks - Cummulative lift", xlab = "Percentile", ylab = "Cummulative lift")  
# add the cummulative lift for the test set  
lines(ann\_lift$percentile, ann\_lift$Test, col = "green", lwd = 2)  
grid() # add the grid  
legend("topright", paste0(legend\_names, " : top semi-decile = ", sprintf("%0.03f", ann\_lift[1,2:3])), lwd = 2, col = c("red", "green"))



#Conclusion: The Neural Networks model performs 7 times better than the pure guess for the top 5% of the predictions.  
#Better performance can be obtained by adding more hidden layers to the network.