Projeto 01 - Formação Cientista de Dados

Thales Henrique Barros Cardoso

# Visão Geral

O risco de fraude está em toda parte, mas para empresas que anunciam online, a fraude de cliques pode acontecer em um volume enorme, resultando em dados de cliques enganosos e dinheiro desperdiçado. Os canais de anúncios podem aumentar os custos simplesmente clicando no anúncio em grande escala. Com mais de 1 bilhão de dispositivos móveis inteligentes em uso ativo todos os meses, a China é o maior mercado móvel do mundo e, portanto, sofre com enormes volumes de tráfego fradulento. TalkingData, a maior plataforma independente de serviços de big data da China, cobre mais de 70% dos dispositivos móveis ativos em todo o país. Eles lidam com 3 bilhões de cliques por dia, dos quais 90% são potencialmente fraudulentos. Sua abordagem atual para evitar a fraude de cliques para desenvolvedores de aplicativos é medir a jornada do clique de um usuário em seu portfólio e sinalizar endereços IP que produzem muitos cliques, mas nunca acabam instalando aplicativos. Com essas informações, eles criaram uma lista negra de IP e uma lista negra de dispositivos. Embora tenham sucesso, eles desejam estar sempre um passo à frente dos fraudadores e recorreram à comunidade Kaggle para obter ajuda no desenvolvimento de sua solução. Em sua segunda competição com o Kaggle, você é desafiado a criar um algoritmo que preveja se um usuário fará o download de um aplicativo depois de clicar em um anúncio de aplicativo móvel. Para apoiar sua modelagem, eles forneceram um generoso conjunto de dados cobrindo aproximadamente 200 milhões de cliques em 4 dias!

## Composição dos dados

Campos de dados:

Cada linha dos dados de treinamento contém um registro de clique, com os seguintes recursos.

* ip: endereço ip do clique.
* app: id do app para marketing.
* device: ID do tipo de dispositivo do telefone celular do usuário (por exemplo, iphone 6 plus, iphone 7, huawei mate 7 etc.)
* os: id da versão do sistema operacional do telefone celular do usuário
* channel: id do canal do editor de anúncios para celular
* click\_time: carimbo de data / hora do clique (UTC)
* attribute\_time: se o usuário baixar o aplicativo depois de clicar em um anúncio, é o momento do download do aplicativo
* is\_attributed: o destino que deve ser previsto, indicando que o aplicativo foi baixado

Observe que ip, app, device, os e channel são codificados.

Os dados de teste são semelhantes, com as seguintes diferenças:

* click\_id: referência para fazer previsões
* is\_attributed: não incluído

O dataset está disponível em <https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data>

# Importação e leitura dos dados

# Escolha do diretório  
setwd("C:/CursosDSA/FCD/BigDataRAzure/Cap20/Projeto01")  
getwd()

## [1] "C:/CursosDSA/FCD/BigDataRAzure/Cap20/Projeto01"

# Maniulação de dados  
suppressMessages(library(data.table))  
suppressMessages(library(dplyr))  
suppressMessages(library(bigreadr))  
  
# Gráficos  
suppressMessages(library(ggplot2))  
suppressMessages(library(grid))  
suppressMessages(library(gridExtra))  
  
# Manipulação de datas  
suppressMessages(library(lubridate))  
  
# Machine learning  
suppressMessages(library(caret))  
suppressMessages(library(ROSE))  
suppressMessages(library(randomForest))  
suppressMessages(library(rpart))  
suppressMessages(library(ROCR))  
suppressMessages(library(C50))  
suppressMessages(library(e1071))  
  
# Executando a limpeza do environment  
rm(list=ls())

O dataset é por damasiado grande para funcionar de forma adequada dentro das especificações atuais do meu computador. Dessa forma foi necessário utilizar artifícios para ca criação de uma amostra menor e que fosse feita de forma randômica dentro do dataset original. Para isso vamos utilizar a biblioteca bigreadr que segundo a descrição do pacote serve para “ler arquivos de texto grandes dividindo-os em arquivos menores”.

# Conseguimos ler a quantidade de linhas do dataset original sem precisar carregar o arquivo.  
totalLines <- nlines("train.csv")  
  
# Criar um conjunto com 1/10 das linhas do dataset original  
setLines <- totalLines %/% 10  
  
## Com o total de linhas do dataset original dividiremos em arquivos menores para depois fazermos uma  
## amostragem randômica de um dataset com 1/10 do tammonho do original.  
  
# Divisão dos dados em 10 conjuntos  
split\_file("train.csv", every\_nlines = setLines, prefix\_out = "train", repeat\_header = T)

## $name\_in  
## [1] "C:\\CursosDSA\\FCD\\BigDataRAzure\\Cap20\\Projeto01\\train.csv"  
##   
## $prefix\_out  
## [1] "train"  
##   
## $nfiles  
## [1] 11  
##   
## $nlines\_part  
## [1] 18490389  
##   
## $nlines\_all  
## [1] 184903901  
##   
## $repeat\_header  
## [1] TRUE

# Visualização dos arquivos no diretório  
dir(pattern = ".txt")

## [1] "train\_1.txt" "train\_10.txt" "train\_11.txt" "train\_2.txt" "train\_3.txt"   
## [6] "train\_4.txt" "train\_5.txt" "train\_6.txt" "train\_7.txt" "train\_8.txt"   
## [11] "train\_9.txt"

## O arquivo train\_11.txt surgiu pois o resultado inteiro da divisão do número total de observações divido   
## por 10 gerou um módulo (resto de divisão) que gerou assim o arquivo train\_11.txt  
## Esse arquivo pode ser desconsiderado, pois tem poucas observações, portanto podemos deletar o arquivo  
  
nlines("train\_11.txt")

## [1] 2

# Deletando o arquivo do diretório  
file.remove("train\_11.txt")

## [1] TRUE

# Colocando o nome dos arquivos em um objeto  
setFiles <- c(dir(pattern = ".txt"))  
  
# Criando um data frame com amostras aleatórias dos 20 aquivos gerados  
sampleTrain <- data.table()  
for (i in setFiles){  
 train <- fread(i)  
 sampleTrain <- rbind(sampleTrain, train[sample(1:nrow(train), (setLines/10)),], use.names= FALSE)  
}  
  
# Salvando o arquivo com a amostragem finalizada  
fwrite(sampleTrain, file = "sampleTrain.csv")  
  
# Removendo alguns objetos do ambiente   
rm("i", "setFiles", "setLines", "totalLines", "train")  
  
# Removendo os arquivos ".txt" temporários  
file.remove(c(dir(pattern = ".txt")))

## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

# Lendo o arquivo salvo feito a partir de amostragem randômmica (Quando é preciso recomeçar o trabalho)  
# sampleTrain <- fread("sampleTrain.csv")

Com esse script é possível repartir o dataset original e criar um arquivo menor com observações coletadas de forma randômica, sendo possível agora realizar uma análise exploratória dos dados.

# Análise exploratória dos dados

## Visualização do comportamento dos dados

# Visualização do classe das variáveis  
str(sampleTrain)

## Classes 'data.table' and 'data.frame': 18490380 obs. of 8 variables:  
## $ ip : int 36407 210401 198782 76900 5916 193464 101941 5200 10434 36795 ...  
## $ app : int 15 21 9 18 12 15 8 6 2 18 ...  
## $ device : int 1 1 1 1 1 2 1 1 1 1 ...  
## $ os : int 19 25 13 13 8 17 19 18 19 3 ...  
## $ channel : int 245 128 232 439 178 259 145 459 469 107 ...  
## $ click\_time : POSIXct, format: "2017-11-06 19:44:04" "2017-11-06 22:50:06" ...  
## $ attributed\_time: POSIXct, format: NA NA ...  
## $ is\_attributed : int 0 0 0 0 0 0 0 0 0 0 ...  
## - attr(\*, ".internal.selfref")=<externalptr>

#Visualização das primeiras observações  
head(sampleTrain)

## ip app device os channel click\_time attributed\_time  
## 1: 36407 15 1 19 245 2017-11-06 19:44:04 <NA>  
## 2: 210401 21 1 25 128 2017-11-06 22:50:06 <NA>  
## 3: 198782 9 1 13 232 2017-11-06 23:31:02 <NA>  
## 4: 76900 18 1 13 439 2017-11-06 17:32:28 <NA>  
## 5: 5916 12 1 8 178 2017-11-07 02:24:57 <NA>  
## 6: 193464 15 2 17 259 2017-11-06 21:00:29 <NA>  
## is\_attributed  
## 1: 0  
## 2: 0  
## 3: 0  
## 4: 0  
## 5: 0  
## 6: 0

# Verificando presença de valores NA por coluna.  
colSums(is.na(sampleTrain))

## ip app device os channel   
## 0 0 0 0 0   
## click\_time attributed\_time is\_attributed   
## 0 18444803 0

# Porcentagem de valores NA na coluna attributed\_time  
100 \* sum(is.na(sampleTrain$attributed\_time)) / nrow(sampleTrain)

## [1] 99.75351

# Verificando a quantidade de linhas duplicadas e a porentagem em relação ao total do dataset.  
anyDuplicated(sampleTrain)

## [1] 28178

100 \* (anyDuplicated(sampleTrain) / nrow(sampleTrain))

## [1] 0.1523928

É constatado que a variável attributed\_time, que coleta o horário em que foi realizado download do aplicativo, tem muitos valores NA. A conjectura por trás dessa informação é que pouquíssimos cliques em anúncios acabam se concretizando em downloads. A quantidade de valores nulos chega a porcentagem de 99,7529%.

Em relação as observações duplicadas, elas são muito poucas em relação ao número total de observações, nesse caso não há problemas em excluí-las. Como registro, as linhas duplicadas representam apenas 0,1956% das observações, um valor irrisório que não causará alteração na análise dos dados se forem excluídas.

## Visualização através de gráficos

Para a geração de gráficos de barra par melhor visualizar os dados, vamos transformar as variáveis inteiras em do tipo fator, pelo menos temporariamente, pois posteriormente durante o treinamento dos modelos de machine learning, elas precisarão retornar a serem do tipo “integer”.

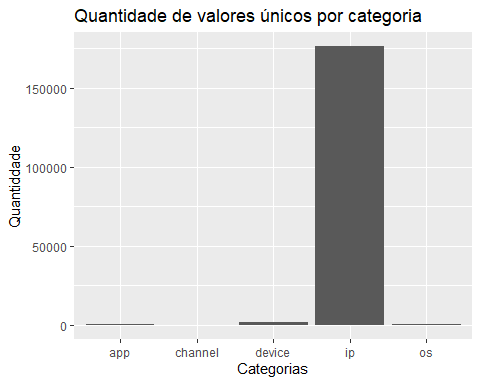
# Transformação das variáveis inteiras em categóricas  
sampleTrain$ip <- factor(sampleTrain$ip)   
sampleTrain$app <- factor(sampleTrain$app)   
sampleTrain$device <- factor(sampleTrain$device)   
sampleTrain$os <- factor(sampleTrain$os)   
sampleTrain$channel <- factor(sampleTrain$channel)   
sampleTrain$is\_attributed <- factor(sampleTrain$is\_attributed)   
  
# Verificando a classe das variáveis novamente  
str(sampleTrain)

## Classes 'data.table' and 'data.frame': 18490380 obs. of 8 variables:  
## $ ip : Factor w/ 176368 levels "1","5","6","9",..: 10873 90099 82871 23188 1770 79604 30660 1562 3216 10999 ...  
## $ app : Factor w/ 475 levels "0","1","2","3",..: 16 22 10 19 13 16 9 7 3 19 ...  
## $ device : Factor w/ 1639 levels "0","1","2","4",..: 2 2 2 2 2 3 2 2 2 2 ...  
## $ os : Factor w/ 400 levels "0","1","2","3",..: 20 26 14 14 9 18 20 19 20 4 ...  
## $ channel : Factor w/ 188 levels "0","3","4","5",..: 79 34 70 149 51 85 42 163 169 17 ...  
## $ click\_time : POSIXct, format: "2017-11-06 19:44:04" "2017-11-06 22:50:06" ...  
## $ attributed\_time: POSIXct, format: NA NA ...  
## $ is\_attributed : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...  
## - attr(\*, ".internal.selfref")=<externalptr>

# Contando o número de valores únicos em cada coluna  
numUnique <- function(x){  
 return(length(unique(x)))  
}  
  
apply(sampleTrain, 2, numUnique)

## ip app device os channel   
## 176368 475 1639 400 188   
## click\_time attributed\_time is\_attributed   
## 259252 40538 2

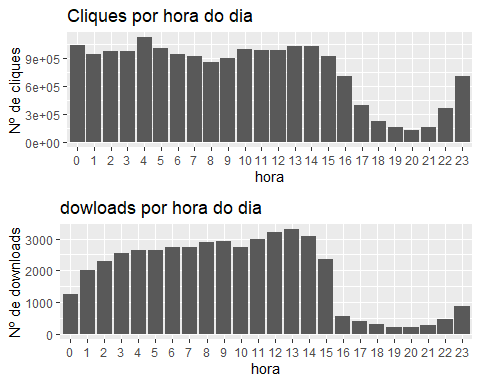
# Inicio da análise exploratória  
uniqueValues <- data.frame(colnames(sampleTrain), sapply(sampleTrain, numUnique))  
colnames(uniqueValues) <- c("categorical", "value")  
uniqueValues <- uniqueValues[-c(6:8), ]  
  
# Vizualização da quantidade de valores únicos  
ggplot(data = uniqueValues, aes(x = categorical, y = value)) +  
 geom\_bar(stat="identity") +  
 xlab("Categorias") +  
 ylab("Quantiddade") +  
 ggtitle("Quantidade de valores únicos por categoria")



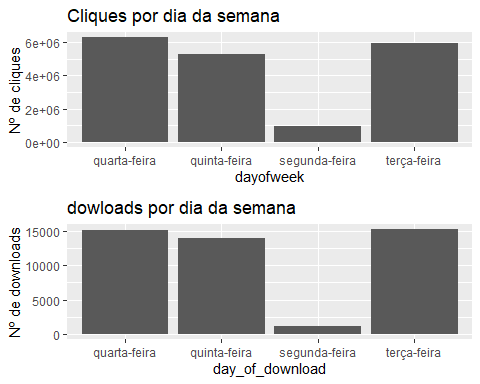
É notório uma quantidade com muitos valores únicos para a categoria IP. Pelo seu próprio significado de internet protocol, indica que muitos aparelhos diferentes acessam os anúncios. É natural que sejam uma quantidade maior em relação aos outros campos como device ou os (sistema operacional), tê-los na mesma proporção seria impossível. Esse primeiro gráfico mostra coerência com o mundo da telefonia móvel.

Para tentar ajudar a visualização do comportamento dos cliques nos anúncios e dos downloads feitos, vamos criar variáveis que coletam os dias da semana e a hora, a partir da variável click\_time, nas quais ocorreram os cliques nos anúncios e foram feitos os downloads. Com isso poderemos visualizar através de gráficos de barra esse comportamento e se o comportamento dos cliques possuem uma distribuição similar aos downloads feitos.

# Criação de variáveis relacionadas a data e horários  
sampleTrain$hora <- factor(hour(sampleTrain$click\_time))  
sampleTrain$dayofweek <- factor(weekdays(sampleTrain$click\_time))  
  
# Visualização de cliques nos anúncios por hora em um dia  
graph1 <- ggplot(data = sampleTrain, aes(x = hora)) +  
 geom\_bar() +   
 ggtitle("Cliques por hora do dia") +  
 ylab("Nº de cliques")  
  
# Visualização de cliques por dia da semana  
graph2 <- ggplot(data = sampleTrain, aes(x = dayofweek)) +  
 geom\_bar() +  
 ggtitle("Cliques por dia da semana") +  
 ylab("Nº de cliques")  
  
# Visualização do horário e data dos downloads do aplicativo  
sampleTrain$hour\_of\_download <- factor(hour(sampleTrain$attributed\_time))  
sampleTrain$day\_of\_download <- factor(weekdays(sampleTrain$attributed\_time))  
  
# Variável temporária para geração do gráfico  
temp <- data.table()  
temp$hour\_of\_dowload <- sampleTrain$hour\_of\_download  
temp$day\_of\_download <- sampleTrain$day\_of\_download  
  
# Variável com muitos valores NA, sendo necessário a exclusão deles.  
temp <- na.omit(temp)  
  
# Visualização dos downloads por hora do dia  
graph3 <- ggplot(data = temp, aes(x = hour\_of\_dowload)) +  
 geom\_bar() +  
 ggtitle("dowloads por hora do dia") +  
 ylab("Nº de downloads") +   
 xlab("hora")  
  
# Visualização de dowloads por dia da semana  
graph4 <- ggplot(data = temp, aes(x = day\_of\_download)) +  
 geom\_bar() +  
 ggtitle("dowloads por dia da semana") +  
 ylab("Nº de downloads")  
  
# Visualização dos gráficos em grade.  
grid.arrange(graph1, graph3)



grid.arrange(graph2, graph4)



No gráfico de cliques por hora do dia podemos observar que há uma diminuição de cliques por volta das 16 horas até as 23 horas, onde se retoma a atividade maior de cliques nos anúncios. Dentro da nossa amostragem verifica-se que durante alguns dias da semana não há registro de cliques, ente eles sexta-feira, sábado e domingo. O que seria algo bastante estranho já que no mundo conectado de hoje as pessoas não deixam de usar seus telefones no fim de semana. Seria um problema da amostragem?

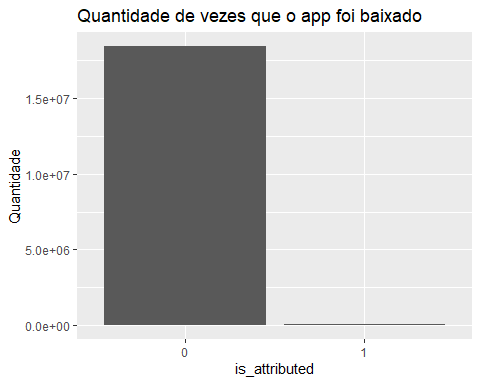
Com esses gráficos conseguimos observar que o comportamento dos downloads feitos possui a mesma características de distribuição, considerando as devidas proporções, dos cliques dados nos anúncios. O que pra mim indica coerência nos dados.

Na sequência visualizaremos a variável target. Como ela se comporta em relação ao balanceamento do resultado.

# Porcentagem das observações da variavel target  
prop.table(table(sampleTrain$is\_attributed)) \* 100

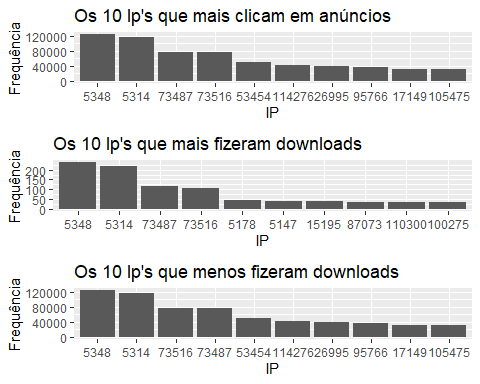
##   
## 0 1   
## 99.7535097 0.2464903

# Vizualização distribuição da variável target  
ggplot(data = sampleTrain, aes(x = is\_attributed)) +  
 geom\_bar() +  
 ggtitle("Quantidade de vezes que o app foi baixado") +  
 ylab("Quantidade")

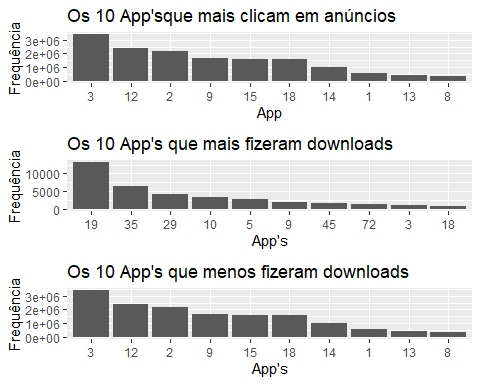


É visível um desbalanceamento dos valores da variável is\_attributed que pode prejudicar mais à frente a criação do modelo de machine learning. Essa questão será resolvida no momento em que façamos a divisão dos dados de treino e teste do modelo que se pretende construir. Continuemos assim com a construção dos demais gráficos.

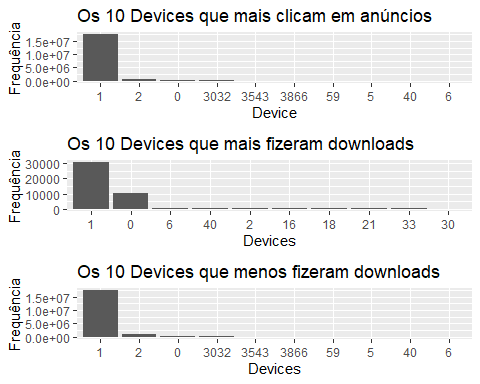
# Os 10 ip's que mais clicaram em anuncios  
g1 <- sampleTrain %>%   
 select(ip) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("IP") +  
 ylab("Frequência") +  
 ggtitle("Os 10 Ip's que mais clicam em anúncios")  
  
# Os 10 app's que mais clicaram em anuncios  
g2 <- sampleTrain %>%   
 select(app) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("App") +  
 ylab("Frequência") +  
 ggtitle("Os 10 App'sque mais clicam em anúncios")  
  
# Os 10 devices que mais clicaram em anuncios  
g3 <- sampleTrain %>%   
 select(device) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("Device") +  
 ylab("Frequência") +  
 ggtitle("Os 10 Devices que mais clicam em anúncios")  
  
# Os 10 os' que mais clicaram em anuncios  
g4 <- sampleTrain %>%   
 select(os) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("Os") +  
 ylab("Frequência") +  
 ggtitle("Os 10 Os' que mais clicam em anúncios")  
  
# Os 10 channels que mais clicaram em anuncios  
g5 <- sampleTrain %>%   
 select(channel) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("IP") +  
 ylab("Frequência") +  
 ggtitle("Os 10 Channels que mais clicam em anúncios")  
  
# Os 10 Ip's que mais fazem o download do aplicativo  
g6 <- sampleTrain %>%   
 filter(is\_attributed == 1) %>%  
 select(ip) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("IP") +  
 ylab("Frequência") +  
 ggtitle("Os 10 Ip's que mais fizeram downloads")  
  
# Os 10 App's que mais fazem o download do aplicativo  
g7 <- sampleTrain %>%   
 filter(is\_attributed == 1) %>%  
 select(app) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("App's") +  
 ylab("Frequência") +  
 ggtitle("Os 10 App's que mais fizeram downloads")  
  
# Os 10 Devices que mais fazem o download do aplicativo  
g8 <- sampleTrain %>%   
 filter(is\_attributed == 1) %>%  
 select(device) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("Devices") +  
 ylab("Frequência") +  
 ggtitle("Os 10 Devices que mais fizeram downloads")  
  
# Os 10 Os's que mais fazem o download do aplicativo  
g9 <- sampleTrain %>%   
 filter(is\_attributed == 1) %>%  
 select(os) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("Os'") +  
 ylab("Frequência") +  
 ggtitle("Os 10 Ip's que mais fizeram downloads")  
  
# Os 10 Channels que mais fazem o download do aplicativo  
g10 <- sampleTrain %>%   
 filter(is\_attributed == 1) %>%  
 select(channel) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("Channels") +  
 ylab("Frequência") +  
 ggtitle("Os 10 Channels que mais fizeram downloads")  
  
# Os 10 Ip's que menos fazem o download do aplicativo  
g11 <- sampleTrain %>%   
 filter(is\_attributed == 0) %>%  
 select(ip) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("IP") +  
 ylab("Frequência") +  
 ggtitle("Os 10 Ip's que menos fizeram downloads")  
  
# Os 10 App's que menos fazem o download do aplicativo  
g12 <- sampleTrain %>%   
 filter(is\_attributed == 0) %>%  
 select(app) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("App's") +  
 ylab("Frequência") +  
 ggtitle("Os 10 App's que menos fizeram downloads")  
  
# Os 10 Devices que menos fazem o download do aplicativo  
g13 <- sampleTrain %>%   
 filter(is\_attributed == 0) %>%  
 select(device) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("Devices") +  
 ylab("Frequência") +  
 ggtitle("Os 10 Devices que menos fizeram downloads")  
  
# Os 10 Os's que menos fazem o download do aplicativo  
g14 <- sampleTrain %>%   
 filter(is\_attributed == 0) %>%  
 select(os) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("Os'") +  
 ylab("Frequência") +  
 ggtitle("Os 10 Os's que menos fizeram downloads")  
  
# Os 10 Channels que menos fazem o download do aplicativo  
g15 <- sampleTrain %>%   
 filter(is\_attributed == 0) %>%  
 select(channel) %>%  
 table() %>%  
 sort(decreasing = TRUE) %>%  
 head(n = 10) %>%  
 data.frame() %>%  
 ggplot(aes(x = ., y = Freq)) +  
 geom\_bar(stat = "identity") +  
 xlab("Channels") +  
 ylab("Frequência") +  
 ggtitle("Os 10 Channels que menos fizeram downloads")  
  
# Plots  
grid.arrange(g1, g6, g11)



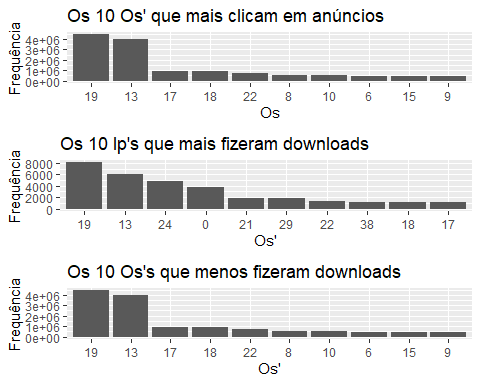
grid.arrange(g2, g7, g12)



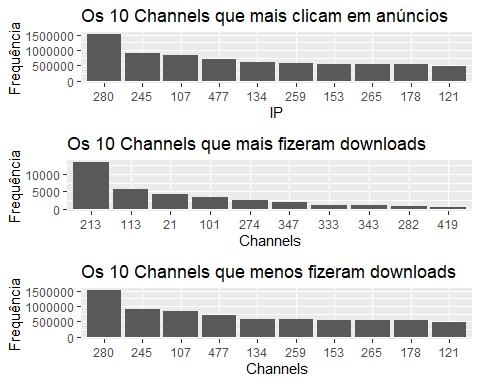
grid.arrange(g3, g8, g13)



grid.arrange(g4, g9, g14)



grid.arrange(g5, g10, g15)



# Removendo Variáveis do ambiente  
rm("graph1", "graph2", "graph3", "graph4")  
rm("g1", "g2", "g3", "g4", "g5", "g6", "g7", "g8", "g9", "g10", "g11", "g12", "g13", "g14", "g15")

Observa-se que os Ip’s 53454, 114276, 26995, 95766, 17149 e 105475 são ip’s que clicam bastante nos anúncios porém não realizam downloads na mesma proporção, o que pode indicar fraude, tipo cliques automatizados. Na verdade, os 10 primeiros que mais clicam em anúncios são os mesmos 10 primeiros que menos fazem downloads. Algo interessante a se destacar é que os 4 primeiros que mais clicam são os mesmos 4 primeiros que mais fazem downloads, o que faz sentido.

Em relação aos app’s podemos observar que quase nenhum dos 10 que mais clicam nos anúncios estão dentro da lista dos 10 que mais fazem download, com exceção do app 9 e 3, e que ainda assim não é na mesma proporção dos cliques. O app 19 por exemplo, não aparece na lista dos 10 que mais clicam, mas é o app que mais faz download. É necessário por parte da empresa observar e tirar os anúncios de alguns app’s, pois pode ter algum tipo de falha de autenticação de usuário que permita a operação com bots de maneira facilitada, nesses app’s, por exemplo.

No atributo device o que chama mais atenção é o de número 1, que é o que mais clica em anúncios, é o que mais baixa e o que menos baixa, o que seria um comportamento que considero como normal para um usuário real. O de número 0 é o terceiro que mais acessa anúncios e o segundo que mais baixa, sendo um bom indicador de que não há fraude nos cliques. Os outros devices que fazem mais downloads não aparecem na lista dos que mais clicam nos anúncios. Provavelmente o device 0, 1 e 2 são aparelhos mais populares, por terem uma maior incidência de cliques, o que não quer dizer que são os mais baratos.

Os Os’ (sistema operacional) com mais cliques provavelmente são as versões de sistemas operacionais mais populares no mercado de telefonia móvel. Observa-se que os que mais clicam são os que menos fazem downloads, porem o 19 e o 13 possuem uma boa taxa de conversão de download. O Os’ 24 tem uma boa taxa de download mesmo sem está no topo dos que mais clicam em anúncios. O Os’ 17 e 18 também aparecem na lista dos 10 que mais fazem downloads mas não na mesma proporção dos cliques.

Em relação aos channels temos que nenhum dos que mais clicam são os que mais fazem downloads. Isso mostra que há a possibilidade de ter bots clicando nos anúncios por esses meios.

De forma geral se vê que os que mais clicam sempre são os que menos fazem downloads, o que não seriaum comportamento anormal para um usuário real. Podemos sim clicar de forma errada quando temos aplicativos ou páginas web muito poluídas, por exemplo. Por vezes, alguns desses que mais clicam são os que mais fazem download, o que faria sentido, pois quantos mais cliques dados maior a probabilidade de achar algo que vale a pena de realizar o download, porém isso aconte numa proporção muito pequena.

# Criação do modelo de machine learning

Nessa etapa há três coisas bastante importantes a serem feitas. Uma delas é a divisão do dataset em conjunto de treino e teste do modelo. Nesse caso o dataset a partir do qual faremos a divisão, não será o sampleTrain, pois devido as limitações computacionais, não foi possível treinar um modelo com tantas observações. Dessa forma o dataset criado possui 1/90 avos do tamanho do sampleTrain. Essa amostra terá 80% das observações direcionadas para treinamento e 20% de observações direcionadas para teste. A segunda coisa que tem de ser feita é o feature selection, escolhendo as variáveis que possuem maior significância para a construção do modelo. O terceiro fator a se considerar é o balanceamento da variável alvo. Ela está extremamente desbalanceada, e como se trata de problema de classificação, o desbalanceamento prejudica e compromete o resultado final do treinamento do modelo.

# Dividindo o dataset (amostra aleatória que representa 90 avos do dataset que já foi dividido inicialmente)  
trainingLines <- sampleTrain[sample(1:nrow(sampleTrain), round(nrow(sampleTrain)/90)),-c(6:7, 11:12)]  
sampleLines <- sample(1:nrow(trainingLines), 0.8 \* nrow(trainingLines))  
  
training <- trainingLines[sampleLines,]  
prop.table(table(training$is\_attributed))

##   
## 0 1   
## 0.997608893 0.002391107

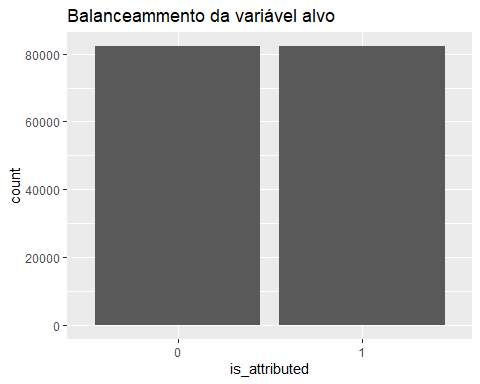
testing <-trainingLines[-sampleLines,]  
prop.table(table(testing$is\_attributed))

##   
## 0 1   
## 0.997517644 0.002482356

# Balanceando a variável alvo para treinar o modelo  
trainingRose <- ROSE(is\_attributed ~ ., data= training, seed = 1)$data  
prop.table(table(trainingRose$is\_attributed))

##   
## 0 1   
## 0.5003681 0.4996319

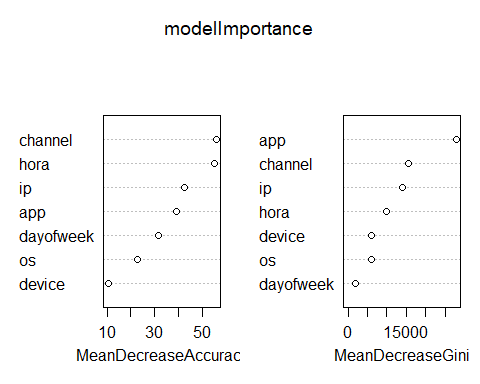
# Verificando graficamente se o metodo de balanceamento foi concluído  
ggplot(data = trainingRose, aes(x = is\_attributed)) +  
 geom\_bar() +   
 ggtitle("Balanceammento da variável alvo")



## Feature selection

Utilizaremos o algoritmo Random Forest para escolher as melhores variáveis para uma possível otimização do modelo de machine leaning. Antes disso vamos transformar as variáveis de classe factor com mais de 53 níveis para integer, pois se permanecer em factor o algoritmo não funciona.

## Tranformando novamente algumas variaveis de factor para integer, para poder utilizar o randomForest no feature selection  
trainingRose$ip <- as.integer(trainingRose$ip)   
trainingRose$app <- as.integer(trainingRose$app)   
trainingRose$device <- as.integer(trainingRose$device)   
trainingRose$os <- as.integer(trainingRose$os)   
trainingRose$channel <- as.integer(trainingRose$channel)   
  
modelImportance <- randomForest(is\_attributed ~ ., data = trainingRose, ntree = 40, nodesize = 2, importance = T)  
varImpPlot(modelImportance)



## Ajustando as variaveis de teste para integer, para realizar a predição no modeloo  
testing$ip <- as.integer(testing$ip)   
testing$app <- as.integer(testing$app)   
testing$device <- as.integer(testing$device)   
testing$os <- as.integer(testing$os)   
testing$channel <- as.integer(testing$channel)

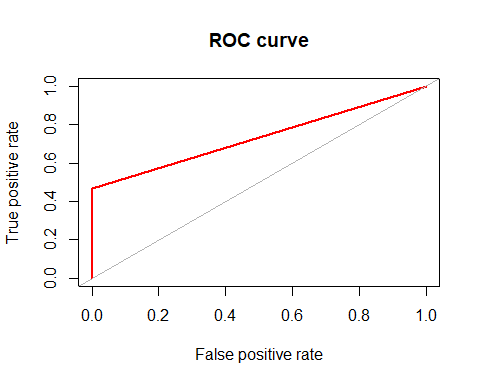
Nesse primeiro momento vamos criar um modelo com todas as variáveis, e após a verificação da sua acurácia, vamos alterando o algoritmo ou o dataset para tentar otimizar o modelo.

## Criação do 1º modelo

# Criando o modelo de machine learning  
model <- randomForest(is\_attributed ~ ., data = trainingRose)  
prediction <- predict(model, testing, type = 'class')  
  
# Visualizando a matriz de confusão  
confusionMatrix(table(dados = testing$is\_attributed,  
 pred = prediction), positive = '1')

## Confusion Matrix and Statistics  
##   
## pred  
## dados 0 1  
## 0 40894 94  
## 1 54 48  
##   
## Accuracy : 0.9964   
## 95% CI : (0.9958, 0.997)  
## No Information Rate : 0.9965   
## P-Value [Acc > NIR] : 0.710948   
##   
## Kappa : 0.3917   
##   
## Mcnemar's Test P-Value : 0.001347   
##   
## Sensitivity : 0.338028   
## Specificity : 0.998681   
## Pos Pred Value : 0.470588   
## Neg Pred Value : 0.997707   
## Prevalence : 0.003456   
## Detection Rate : 0.001168   
## Detection Prevalence : 0.002482   
## Balanced Accuracy : 0.668355   
##   
## 'Positive' Class : 1   
##

#Plot da Curva ROC  
roc.curve(testing$is\_attributed, prediction, plotit = T, col = "red")



## Area under the curve (AUC): 0.734

Com esse primeiro modelo temos uma acurácia de 0.9969, vamos alterar os algoritmos para observar se o random forest é realmente o melhor deles para a criação do modelo com esses dados de treino. Também não podemos ficar presos apenas ao valor da acurácia dos modelos, eles podem não representar de forma fidedigna o quão bom o modelo possa estar.

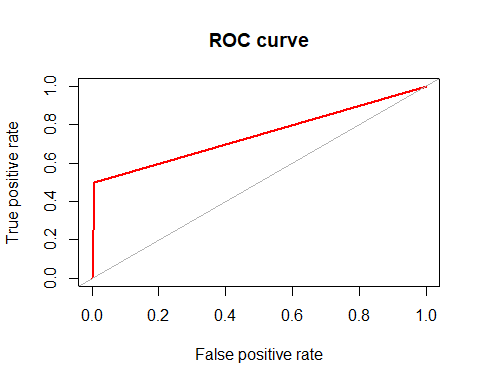
## Buscando otimização para o modelo

Primeiro realizaremos uma alteração para o algoritmo C5.0

#Alterando o algoritmo para o C5.0  
modelOptimized <- C5.0(is\_attributed ~ ., data = trainingRose)  
predictionOpitimized <- predict(modelOptimized, testing, type = 'class')  
  
# Visualizando a matriz de confusão  
confusionMatrix(table(dados = testing$is\_attributed,  
 pred = predictionOpitimized), positive = '1')

## Confusion Matrix and Statistics  
##   
## pred  
## dados 0 1  
## 0 40733 255  
## 1 51 51  
##   
## Accuracy : 0.9926   
## 95% CI : (0.9917, 0.9934)  
## No Information Rate : 0.9926   
## P-Value [Acc > NIR] : 0.5152   
##   
## Kappa : 0.2472   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.166667   
## Specificity : 0.998750   
## Pos Pred Value : 0.500000   
## Neg Pred Value : 0.993779   
## Prevalence : 0.007447   
## Detection Rate : 0.001241   
## Detection Prevalence : 0.002482   
## Balanced Accuracy : 0.582708   
##   
## 'Positive' Class : 1   
##

#Plot da Curva ROC  
roc.curve(testing$is\_attributed, predictionOpitimized, plotit = T, col = "red")



## Area under the curve (AUC): 0.747

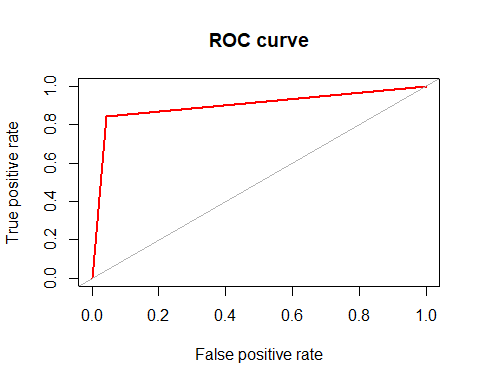
Esse modelo com o algoritmo C5.0 obteve uma acurácia de 0.9926, o que também é um valor extremamente interessante. Lembrando novamente de depois verifiar outros parâmetros importantes para um modelo de classificação

Por último temos um modelo feito com o algoritmo rpart, nos dando uma acurácia de 0.9664 como poderemos ver abaixo.

#Alterando o algoritmo para o rpart  
modelOptimized\_3 <- rpart(is\_attributed ~ ., data = trainingRose)  
predictionOpitimized\_3 <- predict(modelOptimized\_3, testing, type = 'class')  
  
# Visualizando a matriz de confusão  
confusionMatrix(table(dados = testing$is\_attributed,  
 pred = predictionOpitimized\_3), positive = '1')

## Confusion Matrix and Statistics  
##   
## pred  
## dados 0 1  
## 0 39181 1807  
## 1 16 86  
##   
## Accuracy : 0.9556   
## 95% CI : (0.9536, 0.9576)  
## No Information Rate : 0.9539   
## P-Value [Acc > NIR] : 0.05034   
##   
## Kappa : 0.0819   
##   
## Mcnemar's Test P-Value : < 2e-16   
##   
## Sensitivity : 0.045431   
## Specificity : 0.999592   
## Pos Pred Value : 0.843137   
## Neg Pred Value : 0.955914   
## Prevalence : 0.046070   
## Detection Rate : 0.002093   
## Detection Prevalence : 0.002482   
## Balanced Accuracy : 0.522511   
##   
## 'Positive' Class : 1   
##

#Plot da Curva ROC  
roc.curve(testing$is\_attributed, predictionOpitimized\_3, plotit = T, col = "red")



## Area under the curve (AUC): 0.900

Não podemos analisar esses modelos apenas pela acurácia de cada um deles. Vamos tentar nos aprofundar em outros parâmetros, pois em um dataset de teste tão desbalanceado apenas a acurácia não é um parâmetro muito balizador.

—> A partir desse momento vamos nos refirir as métricas de avaliação pelos nomes delas em inglês, para não ocorrer problemas com tradução (accuracy, recall, precision).

Analisar apenas através da “accuracy” pode nos levar a uma falsa ideia de que o modelo está bem treinado em reconhecer o que foi definido no problema de negócio. Porém, como já dito, em datasets onde a variável alvo está desbalanceada de forma extrema, pois 0 representa 99,75% dos dados da coluna is\_attributed e 1 apenas 0,25%, temos que buscar outros parâmetros para melhorar nossa análise. Entre eles estão o “recall” e “precision”.

É solicitado que o algoritmo que preveja se um usuário fará o download de um aplicativo depois de clicar em um anúncio de aplicativo móvel. Então temos que entender que os falsos positivos são prejudiciais para o nosso modelo, pois não queremos que o modelo preveja que o usuário fará o download, quando na verdade ele não o fará. Já os falsos negativos (FN) não são tão importantes assim, pois se o algoritmo prever que o usuário não fará o download e ele fizer não é tão problemático quanto a situação anterior.

O “recall” nos diz quantos dos casos positivos reais fomos capazes de prever corretamente com nosso modelo. Sendo uma métrica útil nos casos em que o FN supera o falso positivo (FP). O que vem a ser o que ocorre nos modelos criados porque o dataset é desbalanceado.

E a “precision” nos diz quantos dos casos previstos corretamente acabaram sendo positivos. Sendo uma métrica útil nos casos em que os FP são uma preocupação mais importante do que os FN. O que também vem a ser o caso dos nossos modelos, como já foi dito mais acima.

Logo, pelo tipo do problema, preocupar-se com os FP é mais importante do levar em consideração a quantidade de FN da confusion matrix. Assim penso que um bom parâmetro a ser analisado seria a “precision”.

Por definição tem-se que: . Quanto menos FP tivermos, melhor será a predição para o problema que queremos, pois assim conseguisse evitar fraudes, já que o sistema não irá predizer muitos FP.

Dentro dos modelos criados com diferentes algoritmos, percebemos que o random forest foi o que apresentou o menor número de FP, o que imapctou na métrica “precision”, pois quanto menor o número de FP, melhor será o resultado da métrica. Sendo esse algoritmo, por motivos já explicados, o melhor entre os que testamos. Pois resultados de FP é ruim para a solução que queremos implementar.

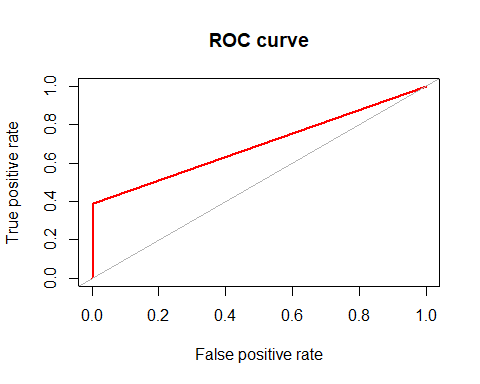
A partir desse ponto busca-se a partir de outras formas melhorar o modelo. Isso pode ser feito criando um modelo apenas com as variáveis mais importantes, conhecidas previamente, ou excluir alguma variável ou criar alguma. Dessa forma teremos modelos suficientes para fazer uma boa escolha.

## Modelo criado utilizando apenas os quatro atributos considerados os mais importantes de acordo com o random forest

## Primeiro vamos criar uma data frame onde estejam apenas as 4 melhores variaveis de acordo com o random forest  
trainingRose\_2 <- trainingRose[, c(1,2,6,7,8)]  
testing\_2 <- testing[, c(1,2,6,7,8)]  
  
# Criando o modelo de machine learning  
modelNewData <- randomForest(is\_attributed ~ ., data = trainingRose\_2)  
predictionNewData <- predict(modelNewData, testing\_2, type = 'class')  
  
# Visualizando a matriz de confusão  
confusionMatrix(table(dados = testing\_2$is\_attributed,  
 pred = predictionNewData), positive = '1')

## Confusion Matrix and Statistics  
##   
## pred  
## dados 0 1  
## 0 40851 137  
## 1 62 40  
##   
## Accuracy : 0.9952   
## 95% CI : (0.9944, 0.9958)  
## No Information Rate : 0.9957   
## P-Value [Acc > NIR] : 0.9528   
##   
## Kappa : 0.2845   
##   
## Mcnemar's Test P-Value : 1.557e-07   
##   
## Sensitivity : 0.2259887   
## Specificity : 0.9984846   
## Pos Pred Value : 0.3921569   
## Neg Pred Value : 0.9966576   
## Prevalence : 0.0043076   
## Detection Rate : 0.0009735   
## Detection Prevalence : 0.0024824   
## Balanced Accuracy : 0.6122366   
##   
## 'Positive' Class : 1   
##

#Plot da Curva ROC  
roc.curve(testing\_2$is\_attributed, predictionNewData, plotit = T, col = "red")



## Area under the curve (AUC): 0.694

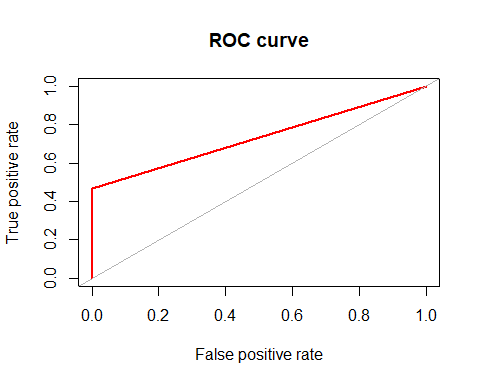
Mesmo criando um dataset com as 4 melhores variaveis escolhidas pelo random forest e ainda assim tendo uma “accuracy” permanecendo na faixa de 0.9952, temos um aumento de FP mostrados da confusion matrix. Como se trata de algoritmo de classificação, não há muito a possibilidade de melhora no resultado com a normalização dos dados.

Realizando um teste, criando um modelo sem a variavel IP.

## Retirando a variavel IP do data set  
trainingRose\_3 <- trainingRose[, c(2,5,6,7)]  
testing\_3 <- testing[, c(2,5,6,7)]  
  
# Criando o modelo de machine learning  
modelNewData <- randomForest(is\_attributed ~ ., data = trainingRose\_3)  
predictionNewData <- predict(modelNewData, testing\_3, type = 'class')  
  
# Visualizando a matriz de confusão  
confusionMatrix(table(dados = testing\_3$is\_attributed,  
 pred = predictionNewData), positive = '1')

## Confusion Matrix and Statistics  
##   
## pred  
## dados 0 1  
## 0 39750 1238  
## 1 29 73  
##   
## Accuracy : 0.9692   
## 95% CI : (0.9674, 0.9708)  
## No Information Rate : 0.9681   
## P-Value [Acc > NIR] : 0.1106   
##   
## Kappa : 0.0992   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.055683   
## Specificity : 0.999271   
## Pos Pred Value : 0.715686   
## Neg Pred Value : 0.969796   
## Prevalence : 0.031906   
## Detection Rate : 0.001777   
## Detection Prevalence : 0.002482   
## Balanced Accuracy : 0.527477   
##   
## 'Positive' Class : 1   
##

#Plot da Curva ROC  
roc.curve(testing$is\_attributed, prediction, plotit = T, col = "red")



## Area under the curve (AUC): 0.734

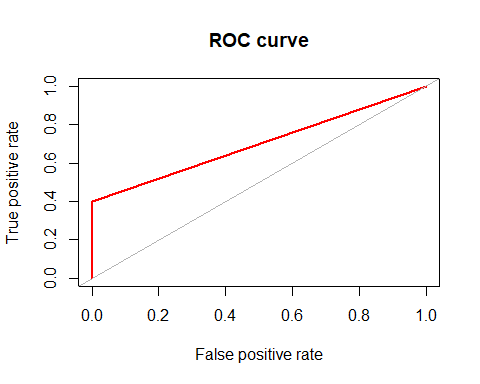
Com os resultados obtidos com o modelo sem a variável IP, observa-se que ela é uma das mais importantes para uma boa construção do modelo de classificação através do algoritmo random forest. Pois sem ela o aumento de FP é significativo, e isso não é bom para o nosso modelo.

Criando um outro modelo onde acrescentaremos variaveis criadas a partir de outras variaveis. Apenas como teste.

## Novamente vamos criar um novo data frame onde vamos acrescentar outras variaveis  
trainingRose\_4 <- trainingRose[, c(1,2,3,4,5,6,7)]  
testing\_4 <- testing[, c(1,2,3,4,5,6,7)]  
  
# Modificação da variável hora para integer  
trainingRose\_4$hora <- as.integer(trainingRose\_4$hora)  
testing\_4$hora <- as.integer(testing\_3$hora)  
  
# Criação de novas variáveis  
# Nesse caso temos variaveis que representam a multiplicação de um dos atributos pela a hora do clique no anuncio  
trainingRose\_4$appHora <- trainingRose\_4$app \* trainingRose\_4$hora   
trainingRose\_4$deviceHora <- trainingRose\_4$device \* trainingRose\_4$hora  
trainingRose\_4$osHora <- trainingRose\_4$os \* trainingRose\_4$hora  
trainingRose\_4$channelHora <- trainingRose\_4$channel \* trainingRose\_4$hora  
  
testing\_4$appHora <- testing\_4$app \* testing\_4$hora   
testing\_4$deviceHora <- testing\_4$device \* testing\_4$hora  
testing\_4$osHora <- testing\_4$os \* testing\_4$hora  
testing\_4$channelHora <- testing\_4$channel \* testing\_4$hora  
  
# Criando o modelo de machine learning  
modelNewData\_2 <- randomForest(is\_attributed ~ ., data = trainingRose\_4)  
predictionNewData\_2 <- predict(modelNewData\_2, testing\_4, type = 'class')  
  
# Visualizando a matriz de confusão  
confusionMatrix(table(dados = testing\_4$is\_attributed,  
 pred = predictionNewData\_2), positive = '1')

## Confusion Matrix and Statistics  
##   
## pred  
## dados 0 1  
## 0 40903 85  
## 1 61 41  
##   
## Accuracy : 0.9964   
## 95% CI : (0.9958, 0.997)  
## No Information Rate : 0.9969   
## P-Value [Acc > NIR] : 0.96384   
##   
## Kappa : 0.3579   
##   
## Mcnemar's Test P-Value : 0.05698   
##   
## Sensitivity : 0.3253968   
## Specificity : 0.9985109   
## Pos Pred Value : 0.4019608   
## Neg Pred Value : 0.9979262   
## Prevalence : 0.0030664   
## Detection Rate : 0.0009978   
## Detection Prevalence : 0.0024824   
## Balanced Accuracy : 0.6619539   
##   
## 'Positive' Class : 1   
##

#Plot da Curva ROC  
roc.curve(testing\_4$is\_attributed, predictionNewData\_2, plotit = T, col = "red")



## Area under the curve (AUC): 0.700

Apesar desse último modelo ser o que melhor performa em relação ao número de FP, ele possui uma “precision” um pouco menor que o primeiro modelo criado, pois esse ultimo modelo possui uma quantidade de TP (true positives) menor que o primeiro.

Dessa forma levando em consideração as dificuldades computacionais encontradas durante a elaboração do projeto, o primeiro modelo de machine learning criado, utilizando todas as variáveis do dataset e o algoritmo random forest, seria o que melhor se adapta a situação proposta pelo projeto.