Projeto com Feedback 1

William Rappel

30/07/2020

Detecção de Fraudes no Tráfego de Cliques em Propagandas de Aplicações Mobile

Apresentação

Meu nome é William Edward Rappel de Amorim, sou estudante de Estatística da Universidade de Brasília (UnB), com provável conclusão no segundo semestre letivo de 2020. Estou cursando a Formação Cientista de Dados da Data Science Academy, que possui como primeiro curso da formação o de Big Data Analytics com R e Microsoft Azure Machine Learning. Um dos projetos desse curso é o de detecção de fraudes no tráfego de cliques em propagandas de aplicações Mobile, que será o objeto de estudo deste projeto.

Descrição do projeto

O risco de fraude está em toda parte, mas para as empresas que anunciam online, a fraude de cliques pode acontecer em um volume avassalador, resultando em dados de cliques enganosos e dinheiro desperdiçado. Os canais de anúncios podem aumentar os custos simplesmente quando pessoas ou bots clicam nos anúncios em grande escala, o que na prática não gera o resultado esperado. Com mais de 1 bilhão de dispositivos móveis em uso todos os meses, a China é o maior mercado móvel do mundo e, portanto, sofre com grandes volumes de tráfego fraudulento.

A TalkingData (https://www.talkingdata.com), a maior plataforma de Big Data independente 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 impedir fraudes de cliques para desenvolvedores de aplicativos é medir a jornada do clique de um usuário em todo o portfólio e sinalizar endereços IP que produzem muitos cliques, mas nunca acabam instalando aplicativos. Com essas informações, eles criaram uma lista negra de IPs e uma lista negra de dispositivos.

Embora bem-sucedidos, eles querem estar sempre um passo à frente dos fraudadores e propuseram uma competição no Kaggle para desenvolver ainda mais a solução. Os competidores foram desafiados a criar um algoritmo que possa prever se um usuário fará o download de um aplicativo depois de clicar em um anúncio de aplicativo para dispositivos móveis. Resumindo, neste projeto, será construído um modelo de aprendizado de máquina para determinar se um clique é fraudulento ou não.

Mais informações sobre o projeto e fontes de dados em: https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data

Leitura do Banco de Dados

Primeiramente, foi realizada a leitura do banco de dados de treinamento, que possui mais de 7 gb e está no formato .csv. Para isso, definiu-se o diretório de trabalho onde se encontram os arquivos do projeto. Além disso, carregou-se os pacotes data.table, dplyr e lubridate, que serão utilizados na leitura e manipulação do banco de dados.

```
# Diretório de trabalho
setwd("C:/FCD/BigDataRAzure/Cap20/Projeto01")
getwd()
```

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

```
# Pacotes de manipulação
library(data.table)
library(dplyr)
library(lubridate)

# Leitura do banco de dados completo
system.time(df <- fread("train.csv"))
```

```
##    user  system elapsed
##   59.10   16.74  404.77
```

Em seguida, estudou-se a estrutura do banco de dados e a tabela de frequências da variável target: *is_attributed*.

```
# Estrutura
str(df)
```

```
## Classes 'data.table' and 'data.frame':   184903890 obs. of  8 variables:
##  $ ip             : int  83230 17357 35810 45745 161007 18787 103022 114221 165970 74544 ...
##  $ app            : int  3 3 3 14 3 3 3 3 3 64 ...
##  $ device         : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ os             : int  13 19 13 13 13 16 23 19 13 22 ...
##  $ channel        : int  379 379 379 478 379 379 379 379 379 459 ...
##  $ click_time     : chr  "2017-11-06 14:32:21" "2017-11-06 14:33:34" "2017-11-06 14:34:12" "2017-11-0
##  $ attributed_time: chr  "" "" "" "" ...
##  $ is_attributed  : int  0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
head(df)
```

```
##        ip app device os channel          click_time attributed_time
## 1:  83230   3      1 13     379 2017-11-06 14:32:21
## 2:  17357   3      1 19     379 2017-11-06 14:33:34
## 3:  35810   3      1 13     379 2017-11-06 14:34:12
## 4:  45745  14      1 13     478 2017-11-06 14:34:52
## 5: 161007   3      1 13     379 2017-11-06 14:35:08
## 6:  18787   3      1 16     379 2017-11-06 14:36:26
##    is_attributed
## 1:             0
## 2:             0
## 3:             0
## 4:             0
## 5:             0
## 6:             0
```

```
# Variável resposta
(tab <- table(df$is_attributed))
```

```
##
##         0         1
## 184447044    456846
```

```r
prop.table(tab)
```

```
##
##           0           1
## 0.997529279 0.002470721
```

Com isso, concluiu-se que o banco de dados estava muito desbalanceado, o que impactaria na estimação dos modelos de aprendizado de máquina.

Rebalanceamento

Como forma de rebalancear e diminuir o tamanho do banco de dados para um que permitisse a execução dos modelos em uma máquina de 8 gb de memória RAM, utilizou-se a seguinte estratégia: dentre as 184.447.044 observações da classe 0, foram amostradas aleatoriamente 456.846 observações; já as observações da classe 1 foram todas mantidas (456.846), de forma que o banco de dados possuísse quantidades iguais de cada classe.

```r
# Rebalanceamento
set.seed(100)
ind_zero <- which(df$is_attributed == 0)
ind_zero_novo <- sample(ind_zero, tab[2])
df2 <- df[c(ind_zero_novo, which(df$is_attributed == 1)),]
rm(df, ind_zero, ind_zero_novo)
```

Em seguida, estudou-se a estrutura desse novo banco de dados, agora devidamente balanceado e nomeado de *df2*.

```r
# Novo banco de dados
str(df2)
```

```
## Classes 'data.table' and 'data.frame':   913692 obs. of  8 variables:
##  $ ip             : int  103555 78833 32976 15187 116881 61288 89103 99915 69886 275098 ...
##  $ app            : int  2 3 3 2 12 26 2 9 15 18 ...
##  $ device         : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ os             : int  18 13 18 18 22 3 32 19 17 17 ...
##  $ channel        : int  219 442 280 219 178 477 212 232 245 449 ...
##  $ click_time     : chr  "2017-11-09 00:31:43" "2017-11-08 23:32:55" "2017-11-07 14:06:35" "2017-11-0
##  $ attributed_time: chr  "" "" "" "" ...
##  $ is_attributed  : int  0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```r
head(df2)
```

```
##        ip app device os channel          click_time attributed_time
## 1: 103555   2      1 18     219 2017-11-09 00:31:43
## 2:  78833   3      1 13     442 2017-11-08 23:32:55
## 3:  32976   3      1 18     280 2017-11-07 14:06:35
## 4:  15187   2      1 18     219 2017-11-08 01:22:36
## 5: 116881  12      1 22     178 2017-11-09 06:29:21
## 6:  61288  26      1  3     477 2017-11-09 10:03:20
##    is_attributed
```

```
## 1:             0
## 2:             0
## 3:             0
## 4:             0
## 5:             0
## 6:             0
```

```
table(df2$is_attributed)
```

```
##
##      0      1
## 456846 456846
```

```
sapply(df2, function(x) length(unique(x)))
```

```
##              ip             app          device              os
##          253030             337            1878             190
##         channel      click_time attributed_time   is_attributed
##             179          229166          182058               2
```

É importante ressaltar que o último comando permite concluir que as variáveis: *app*, *device*, *os* e *channel* possuem muitos níveis diferentes, o que irá impactar no tempo de execução dos modelos.

Reorganizando as variáveis

Em seguida, excluiu-se as colunas *ip* e *attributed_time*, pois não seriam úteis para a predição de *is_attributed*.

```
# Removendo variáveis
df2 <- df2 %>%
  select(-ip, -attributed_time)

# Código para salvar e carregar o banco de dados modificado, respectivamente
# saveRDS(df2, file = "df2.rds")
# df2 <- readRDS("df2.rds")
```

Agora, com o intuito de diminuir a quantidade de níveis das variáveis *app*, *device*, *os* e *channel*, utilizou-se a seguinte abordagem: para a 1ª, 3ª e 4ª, todas as categorias com frequência menor que 10.000 foram agrupadas em uma só categoria, nomeada como 999. Já para *device*, utilizou-se a mesma estratégia, porém com frequência de 1.000. A partir desse estratégia, criou-se 4 novas variáveis, que foram adicionadas ao banco de dados.

```
# Reorganizando variáveis
# app
tab_app <- sort(table(df2$app), decreasing = T)
app_mant <- as.numeric(names(tab_app)[tab_app >= 10000])
app_mod <- ifelse(df2$app %in% app_mant, df2$app, 999)
table(app_mod)
```

```
## app_mod
##     1     2     3     5     8     9    10    11    12    13
## 15626 59104 93809 28146 16178 59474 36247 13900 62182 11421
##    14    15    18    19    29    35    45    72   999
## 27599 43140 47160 134366 41470 64348 16485 11477 131560
```

```r
df2$app_mod <- app_mod
# app_mant: 1, 2, 3, 5, 8, 9, 10, 11, 12, 13, 14, 15, 18, 19, 29, 35, 45, 72

# device
tab_dev <- sort(table(df2$device), decreasing = T)
dev_mant <- as.numeric(names(tab_dev)[tab_dev >= 1000])
dev_mod <- ifelse(df2$device %in% dev_mant, df2$dev, 999)
table(dev_mod)
```

```
## dev_mod
##      0      1      2      6     16     18     21     33     40    999
## 104159 737314  22431   2954   2221   1484   1158   1108   2628  36489
##   3032
##   1746
```

```r
df2$device_mod <- dev_mod
# dev_mant: 0, 1, 2, 6, 16, 18, 21, 33, 40, 3032

# os
tab_os <- sort(table(df2$os), decreasing = T)
os_mant <- as.numeric(names(tab_os)[tab_os >= 10000])
os_mod <- ifelse(df2$os %in% os_mant, df2$os, 999)
table(os_mod)
```

```
## os_mod
##      0      6      8      9     10     13     15     16     17     18
##  38894  17029  18550  13835  19233 159949  15264  11906  35332  33905
##     19     20     21     22     24     25     27     29     32     37
## 189306  14956  19442  31273  48247  15174  10709  19715  11008  15588
##     38    999
##  12399 161978
```

```r
df2$os_mod <- os_mod
# os_mant: 0, 6, 8, 9, 10, 13, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 27, 29, 32, 37, 38

# channel
tab_cha <- sort(table(df2$channel), decreasing = T)
cha_mant <- as.numeric(names(tab_cha))[tab_cha >= 10000]
cha_mod <- ifelse(df2$channel %in% cha_mant, df2$channel, 999)
table(cha_mod)
```

```
## cha_mod
##     21    101    107    113    121    134    145    153    178    205
##  43150  38396  25320  56996  13644  18066  14403  14704  13558  11456
##    213    245    259    265    274    280    343    347    442    477
## 134704  22738  16783  14157  25930  41027  11256  21515  10086  18878
##    999
## 346925
```

```r
df2$channel_mod <- cha_mod
# cha_mant: 21, 101, 107, 113, 121, 134, 145, 153, 178, 205, 213, 245, 259, 265, 274, 280, 343, 347, 44
```

Em relação a variável *clicktime*, 2 novas variáveis foram criadas: uma contendo apenas o dia da semana e outra apenas a hora do click. Essas variáveis foram mantidas como numéricas.

```
df2$click_time <- ymd_hms(df2$click_time)
df2 <- df2 %>%
  mutate(weekday = as.factor(weekdays(click_time)), hour = hour(click_time))
str(df2)
```

```
## Classes 'data.table' and 'data.frame':   913692 obs. of  12 variables:
##  $ app          : int  2 3 3 2 12 26 2 9 15 18 ...
##  $ device       : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ os           : int  18 13 18 18 22 3 32 19 17 17 ...
##  $ channel      : int  219 442 280 219 178 477 212 232 245 449 ...
##  $ click_time   : POSIXct, format: "2017-11-09 00:31:43" "2017-11-08 23:32:55" ...
##  $ is_attributed: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ app_mod      : num  2 3 3 2 12 999 2 9 15 18 ...
##  $ device_mod   : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ os_mod       : num  18 13 18 18 22 999 32 19 17 17 ...
##  $ channel_mod  : num  999 442 280 999 178 477 999 999 245 999 ...
##  $ weekday      : Factor w/ 4 levels "quarta-feira",..: 2 1 4 1 2 2 1 2 4 1 ...
##  $ hour         : int  0 23 14 1 6 10 5 6 7 2 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
df2$weekday <- as.numeric(ordered(df2$weekday, levels = c("segunda-feira",
                                                          "terça-feira",
                                                          "quarta-feira",
                                                          "quinta-feira")))
```

Em seguida, criou-se o banco de dados final, contendo apenas a variável resposta e as 6 variáveis criadas acima.

```
final <- df2 %>%
  select(app_mod, device_mod, os_mod, channel_mod, weekday, hour, is_attributed) %>%
  mutate_at(c("app_mod", "device_mod", "os_mod", "channel_mod"), as.factor)
str(final)
```

```
## Classes 'data.table' and 'data.frame':   913692 obs. of  7 variables:
##  $ app_mod      : Factor w/ 19 levels "1","2","3","5",..: 2 3 3 2 9 19 2 6 12 13 ...
##  $ device_mod   : Factor w/ 11 levels "0","1","2","6",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ os_mod       : Factor w/ 22 levels "0","6","8","9",..: 10 6 10 10 14 22 19 11 9 9 ...
##  $ channel_mod  : Factor w/ 21 levels "21","101","107",..: 21 19 16 21 9 20 21 21 12 21 ...
##  $ weekday      : num  4 3 2 3 4 4 3 4 2 3 ...
##  $ hour         : int  0 23 14 1 6 10 5 6 7 2 ...
##  $ is_attributed: int  0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
# Código para salvar e carregar o banco de dados modificado, respectivamente
# saveRDS(final, file = "final.rds")
# final <- readRDS("final.rds")
```

Divisão do Banco de Dados

O banco de dados foi dividido em 2: um para treinamento e outro para teste. A proporção utilizada foi de 70% para treinamento e 30% para teste.

```
# Treino e Validação
set.seed(200)
treino <- sample(1:nrow(final), ceiling(0.7*nrow(final)))
df_treino <- final[treino,]
df_teste <- final[-treino,]
```

Em seguida, estudou-se a estrutura dos novos bancos, assim como a distribuição da variável target em cada um e a quantidade de níveis das variáveis explicativas.

```
str(df_treino)
```

```
## Classes 'data.table' and 'data.frame':   639585 obs. of  7 variables:
##  $ app_mod      : Factor w/ 19 levels "1","2","3","5",..: 19 12 14 14 12 14 17 14 13 16 ...
##  $ device_mod   : Factor w/ 11 levels "0","1","2","6",..: 2 2 10 8 2 1 2 1 2 2 ...
##  $ os_mod       : Factor w/ 22 levels "0","6","8","9",..: 11 6 13 18 6 15 22 15 11 11 ...
##  $ channel_mod  : Factor w/ 21 levels "21","101","107",..: 21 12 11 21 12 11 21 11 5 1 ...
##  $ weekday      : num  3 3 3 2 3 3 3 1 2 3 ...
##  $ hour         : int  0 16 11 14 0 7 5 21 13 4 ...
##  $ is_attributed: int  0 0 1 1 0 1 1 1 0 1 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
str(df_teste)
```

```
## Classes 'data.table' and 'data.frame':   274107 obs. of  7 variables:
##  $ app_mod      : Factor w/ 19 levels "1","2","3","5",..: 3 6 12 13 3 2 19 9 3 19 ...
##  $ device_mod   : Factor w/ 11 levels "0","1","2","6",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ os_mod       : Factor w/ 22 levels "0","6","8","9",..: 6 11 9 9 6 9 2 17 4 14 ...
##  $ channel_mod  : Factor w/ 21 levels "21","101","107",..: 19 21 12 21 10 21 2 21 21 13 ...
##  $ weekday      : num  3 4 2 3 3 3 4 4 2 3 ...
##  $ hour         : int  23 6 7 2 13 2 9 1 19 16 ...
##  $ is_attributed: int  0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
prop.table(table(df_treino$is_attributed))
```

```
##
##         0         1
## 0.5003526 0.4996474
```

```
prop.table(table(df_teste$is_attributed))
```

```
##
##         0         1
## 0.4991773 0.5008227
```

```r
sapply(final, function(x) length(unique(x)))
```

```
##       app_mod    device_mod        os_mod   channel_mod       weekday
##            19            11            22            21             4
##          hour is_attributed
##            24             2
```

```r
sapply(df_treino, function(x) length(unique(x)))
```

```
##       app_mod    device_mod        os_mod   channel_mod       weekday
##            19            11            22            21             4
##          hour is_attributed
##            24             2
```

```r
sapply(df_teste, function(x) length(unique(x)))
```

```
##       app_mod    device_mod        os_mod   channel_mod       weekday
##            19            11            22            21             4
##          hour is_attributed
##            24             2
```

```r
rm(list=ls()[!(ls() %in% c('df_treino','df_teste'))])
# Código para salvar e carregar os bancos de dados de treinamento e teste, respectivamente
# saveRDS(df_treino, file = "df_treino.rds")
# df_treino <- readRDS("df_treino.rds")
# saveRDS(df_teste, file = "df_teste.rds")
# df_teste <- readRDS("df_teste.rds")
```

Com isso, concluiu-se que os bancos de dados apresentavam propriedades semelhantes.

Análise Exploratória

Valores Faltantes (NAs)

Primeiro, verificou-se que nenhuma das variáveis apresentavam valores faltantes, tanto no banco de dados de treinamento como no de teste.

```r
# Análise Exploratória
# NA
sapply(df_treino, function(x) sum(is.na(x)))
```

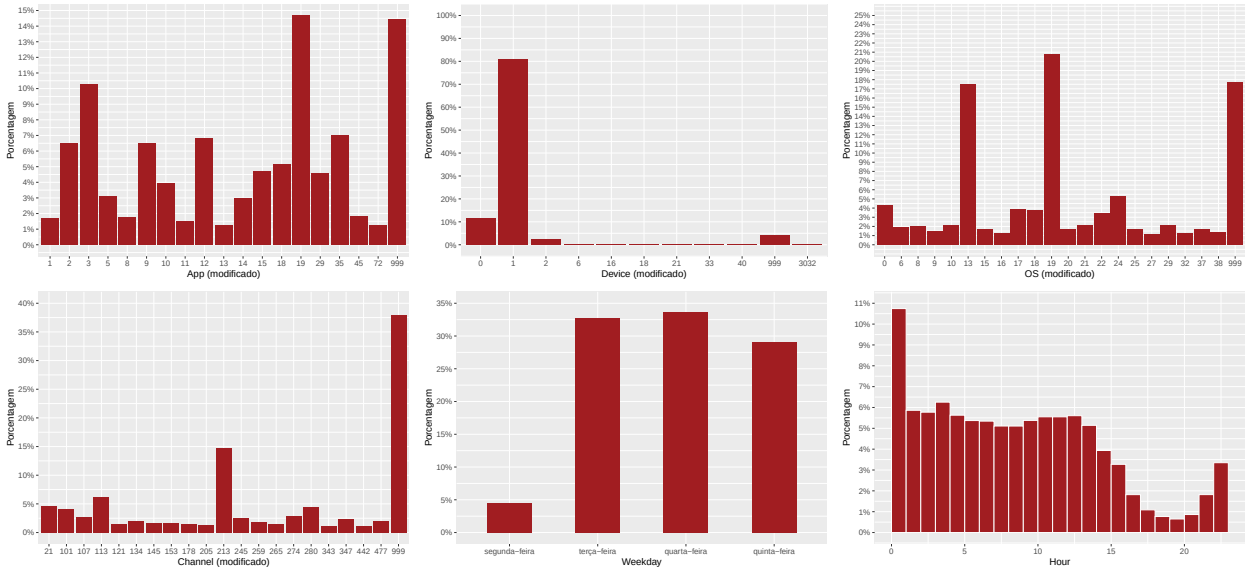
```
##       app_mod    device_mod        os_mod   channel_mod       weekday
##             0             0             0             0             0
##          hour is_attributed
##             0             0
```

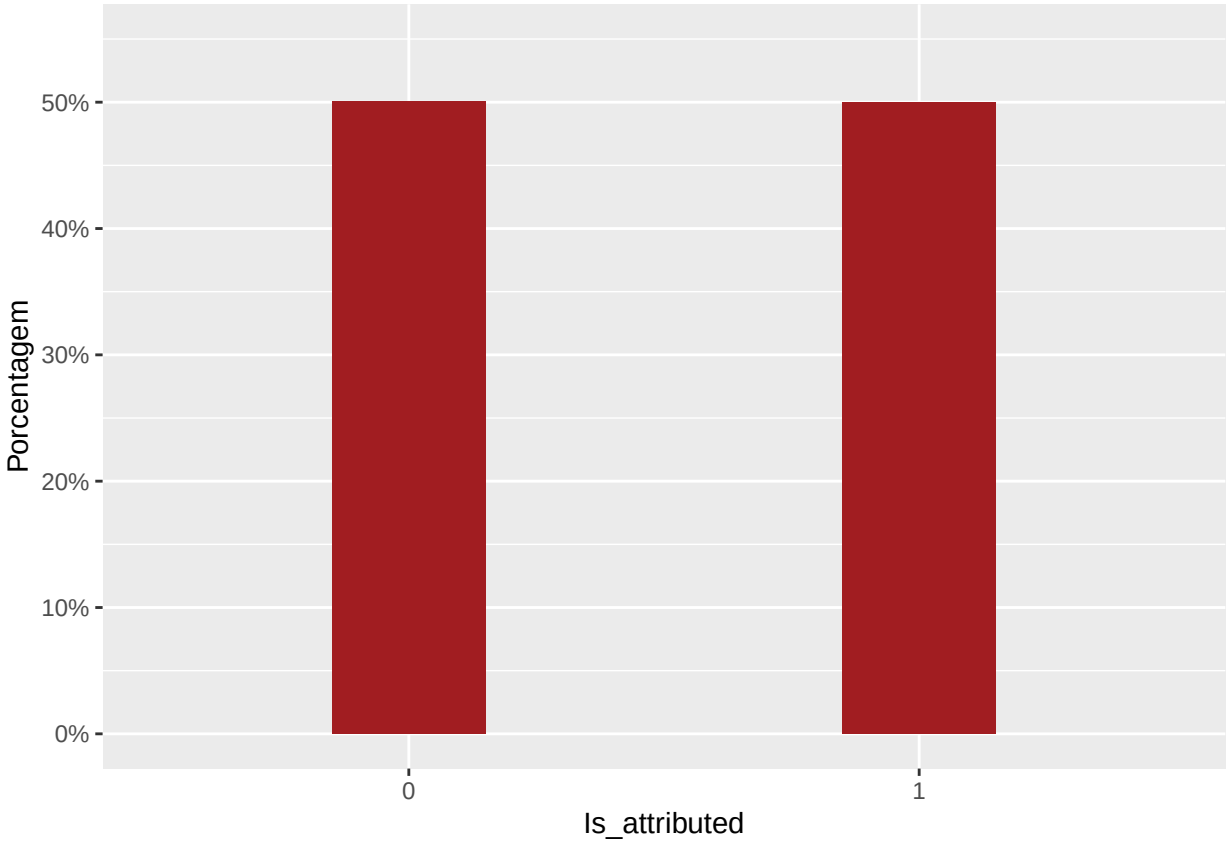
```r
sapply(df_teste, function(x) sum(is.na(x)))
```

```
##       app_mod    device_mod        os_mod   channel_mod       weekday
##             0             0             0             0             0
##          hour is_attributed
##             0             0
```

Individual

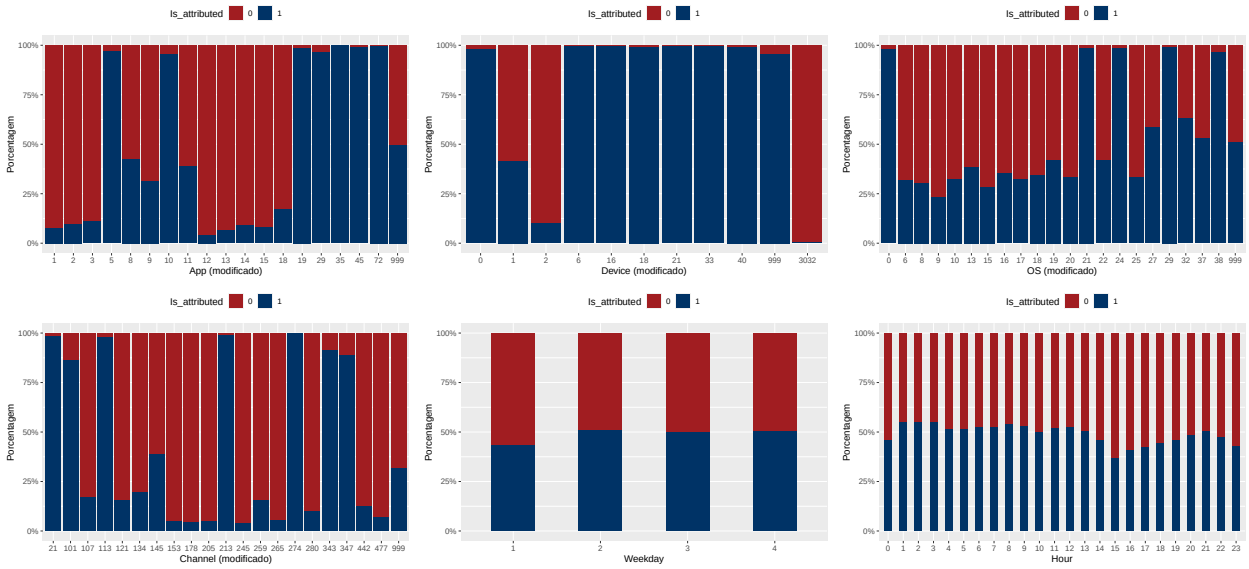
Em seguida, realizou-se uma análise exploratória individual das variáveis, com o objetivo de estudar o comportamento de cada atributo do banco de dados de treinamento, por meio de gráficos.





Bivariada

Por último, realizou-se uma análise bivariada envolvendo a variável target *is_attributed* em função de cada uma das variáveis explicativas. Para isso, elaborou-se gráficos ilustrando a distribuição da variável target para cada valor das variáveis explicativas, no banco de dados de treinamento.



Estimação dos Modelos

Primeiro, carregou-se todos os pacotes que seriam utilizados. Além disso, carregou-se o script *plot_utils.R*, que fornecia algumas funções para construção de gráficos da curva ROC. Conforme descrito no site da competição, o critério de classificação seria a área abaixo da curva ROC entre a probabilidade predita e o valor observado. Por isso, esse será o critério principal utilizado para escolher o melhor modelo.

```
library(caret)
library(ROCR)
library(pROC)
library(randomForest)
library(class)
library(gbm)
source("plot_utils.R")
```

Devido ao tempo de execução muito elevado para estimação de alguns dos modelos, será apenas apresentado o código utilizado na primeira execução, porém os modelos já foram calculados e serão apenas carregados no script, por meio do formato RDS.

Regressão Logística

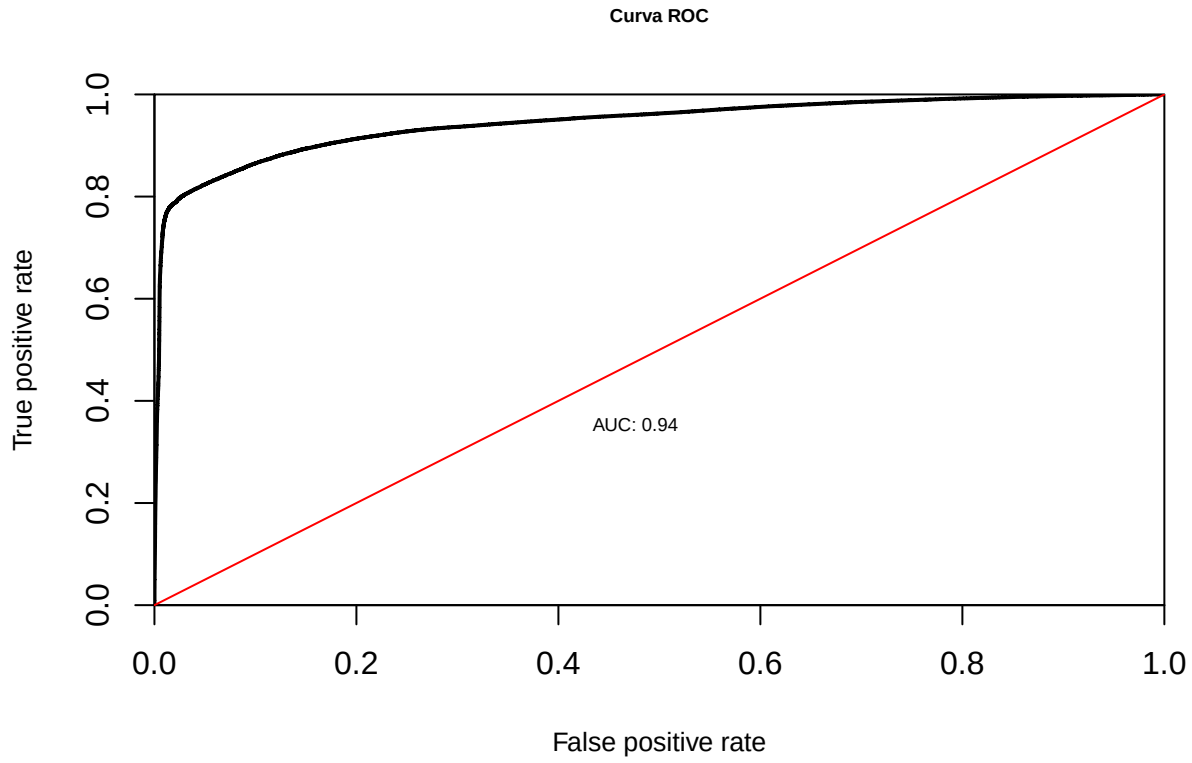
```
# Regressão Logística
# Completo
reg.log <- glm(is_attributed ~ app_mod + device_mod + os_mod + channel_mod + weekday + hour,
               data = df_treino, family = binomial(link = logit))
saveRDS(reg.log, "reg.log.rds")
```

```r
reg.log <- readRDS("reg.log.rds")
```

```r
reg.log.probs <- predict(reg.log, newdata = df_teste, type = "response")
reg.log.preds <- round(reg.log.probs)
confusionMatrix(table(observado = df_teste$is_attributed, predito = reg.log.preds), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##          predito
## observado      0      1
##         0 131799   5029
##         1  25967 111312
##
##                Accuracy : 0.8869
##                  95% CI : (0.8857, 0.8881)
##     No Information Rate : 0.5756
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7739
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9568
##             Specificity : 0.8354
##          Pos Pred Value : 0.8108
##          Neg Pred Value : 0.9632
##              Prevalence : 0.4244
##          Detection Rate : 0.4061
##    Detection Prevalence : 0.5008
##       Balanced Accuracy : 0.8961
##
##        'Positive' Class : 1
##
```

```r
predictions <- prediction(reg.log.probs, df_teste$is_attributed)
plot.roc.curve(predictions, title.text = "Curva ROC")
```




```
auc(roc(df_teste$is_attributed, reg.log.probs))
```

```
## Area under the curve: 0.9446
```

Esse modelo obteve AUC de 0.9446.

Random Forest

Para este método, foi necessário selecionar uma amostra de apenas 100.000 observações para o treinamento, pois se não o computador travava e o R era finalizado.

```
set.seed(300)
rf.mod <- randomForest(as.factor(is_attributed) ~ ., data = df_treino, importance = TRUE,
                       subset = sample(1:nrow(df_treino), 100000))
saveRDS(rf.mod, "rf.mod.rds")
```

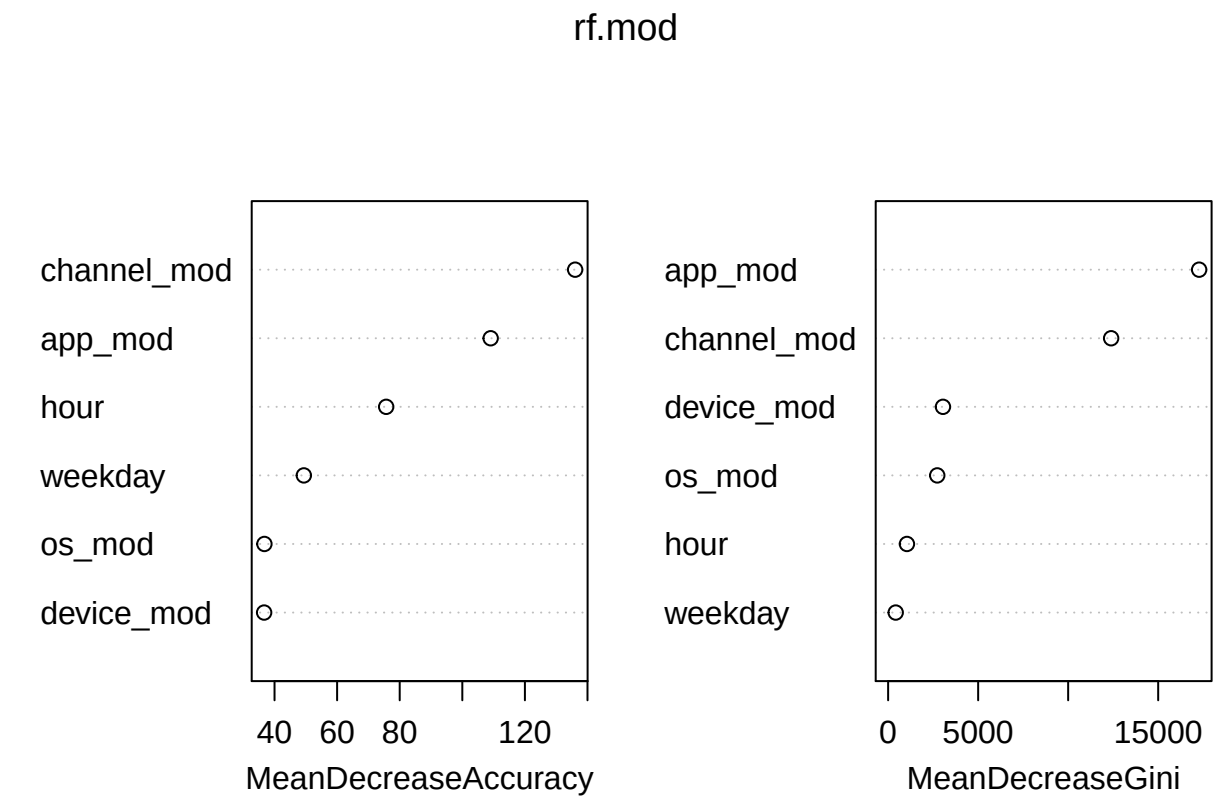
```
rf.mod <- readRDS("rf.mod.rds")
```

```
rf.mod
```

```
##
## Call:
##  randomForest(formula = as.factor(is_attributed) ~ ., data = df_treino,      importance = TRUE, subs
```

```
##                 Type of random forest: classification
##                       Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 10.76%
## Confusion matrix:
##       0     1 class.error
## 0 47786  2305  0.04601625
## 1  8459 41450  0.16948847
```

```
varImpPlot(rf.mod)
```

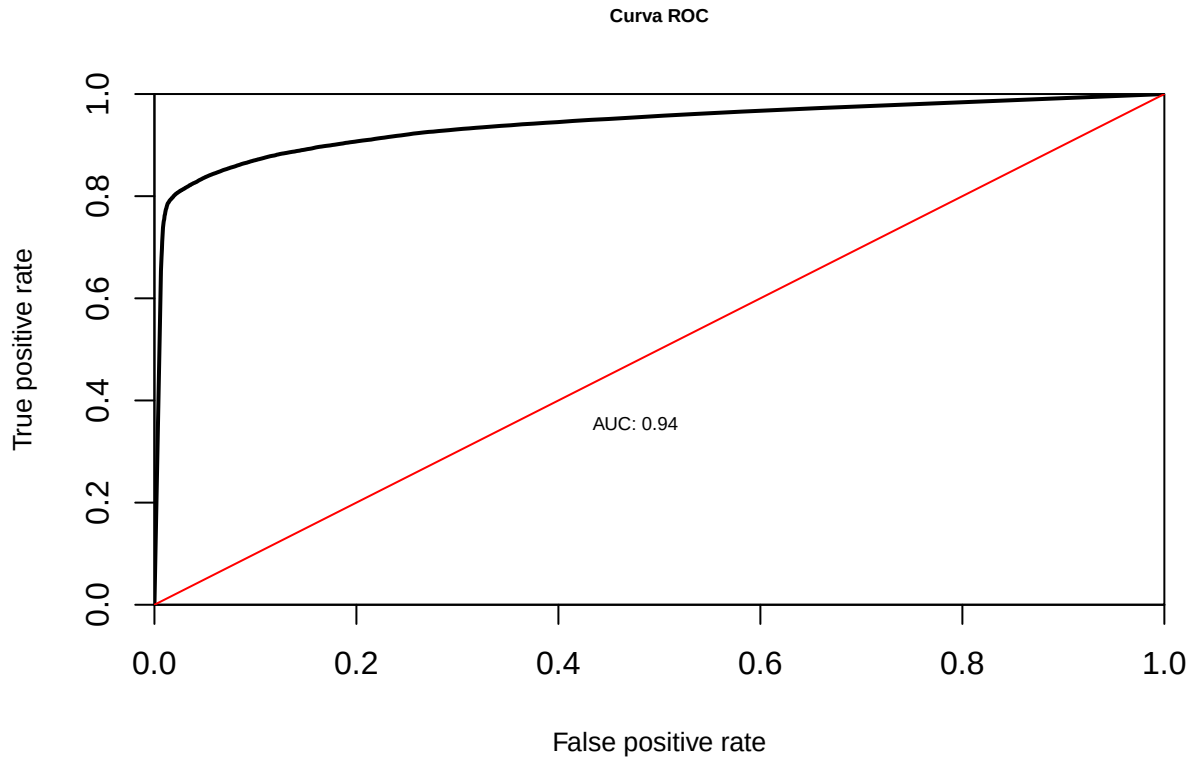



```
rf.probs <- predict(rf.mod, newdata = df_teste, type = "prob")[,2]
rf.preds <- predict(rf.mod, newdata = df_teste, type = "response")
confusionMatrix(table(observado = df_teste$is_attributed, predito = rf.preds), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##          predito
## observado      0      1
##         0 130767   6061
##         1  23186 114093
##
##                Accuracy : 0.8933
##                  95% CI : (0.8921, 0.8945)
```

```
##     No Information Rate : 0.5617
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7866
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9496
##             Specificity : 0.8494
##          Pos Pred Value : 0.8311
##          Neg Pred Value : 0.9557
##              Prevalence : 0.4383
##          Detection Rate : 0.4162
##    Detection Prevalence : 0.5008
##       Balanced Accuracy : 0.8995
##
##        'Positive' Class : 1
##
```

```
predictions <- prediction(rf.probs, df_teste$is_attributed)
plot.roc.curve(predictions, title.text = "Curva ROC")
```




```
auc(roc(df_teste$is_attributed, rf.probs))
```

```
## Area under the curve: 0.9398
```

Esse modelo obteve AUC de 0.9398.

KNN

Para este método, também foi necessário selecionar uma amostra de apenas 100.000 observações.

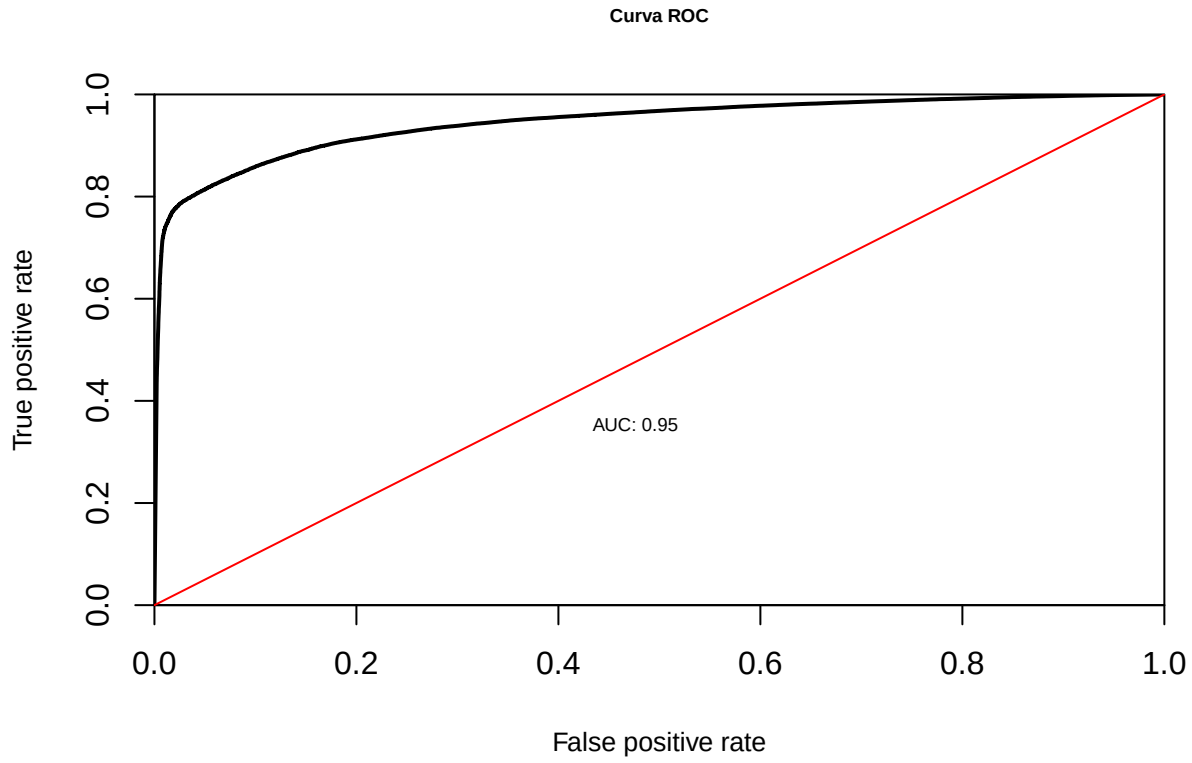
```
set.seed(400)
ind <- sample(1:nrow(df_treino), 100000)
knn.preds <- knn(train = df_treino[ind,-c('is_attributed')],
                 test = df_teste[,-c('is_attributed')],
                 cl = as.factor(df_treino$is_attributed[ind]), k = 50, prob = TRUE)
saveRDS(knn.preds, "knn.preds.rds")
```

```
knn.preds <- readRDS("knn.preds.rds")
```

```
knn.probs <- ifelse(knn.preds == 1, attributes(knn.preds)$prob, 1-attributes(knn.preds)$prob)
confusionMatrix(table(observado = df_teste$is_attributed, predito = knn.preds), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##          predito
## observado      0      1
##         0 128257   8571
##         1  23812 113467
##
##                Accuracy : 0.8819
##                  95% CI : (0.8806, 0.8831)
##     No Information Rate : 0.5548
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7638
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9298
##             Specificity : 0.8434
##          Pos Pred Value : 0.8265
##          Neg Pred Value : 0.9374
##              Prevalence : 0.4452
##          Detection Rate : 0.4140
##    Detection Prevalence : 0.5008
##       Balanced Accuracy : 0.8866
##
##        'Positive' Class : 1
##
```

```
predictions <- prediction(knn.probs, df_teste$is_attributed)
plot.roc.curve(predictions, title.text = "Curva ROC")
```




```
auc(roc(df_teste$is_attributed, knn.probs))
```

```
## Area under the curve: 0.945
```

Esse modelo obteve AUC de 0.945.

Bagging

Este método foi composto de *bagging* de árvores de decisão, sendo muito similar ao Random Forest, porém com o parâmetro m igual ao número de variáveis explicativas (nesse caso, 6). Também selecionou-se uma amostra de 100.000 observações.

```
set.seed(500)
bag.mod <- randomForest(as.factor(is_attributed) ~ ., data = df_treino, importance = TRUE,
                        subset = sample(1:nrow(df_treino), 100000), mtry = 6)
saveRDS(bag.mod, "bag.mod.rds")
```

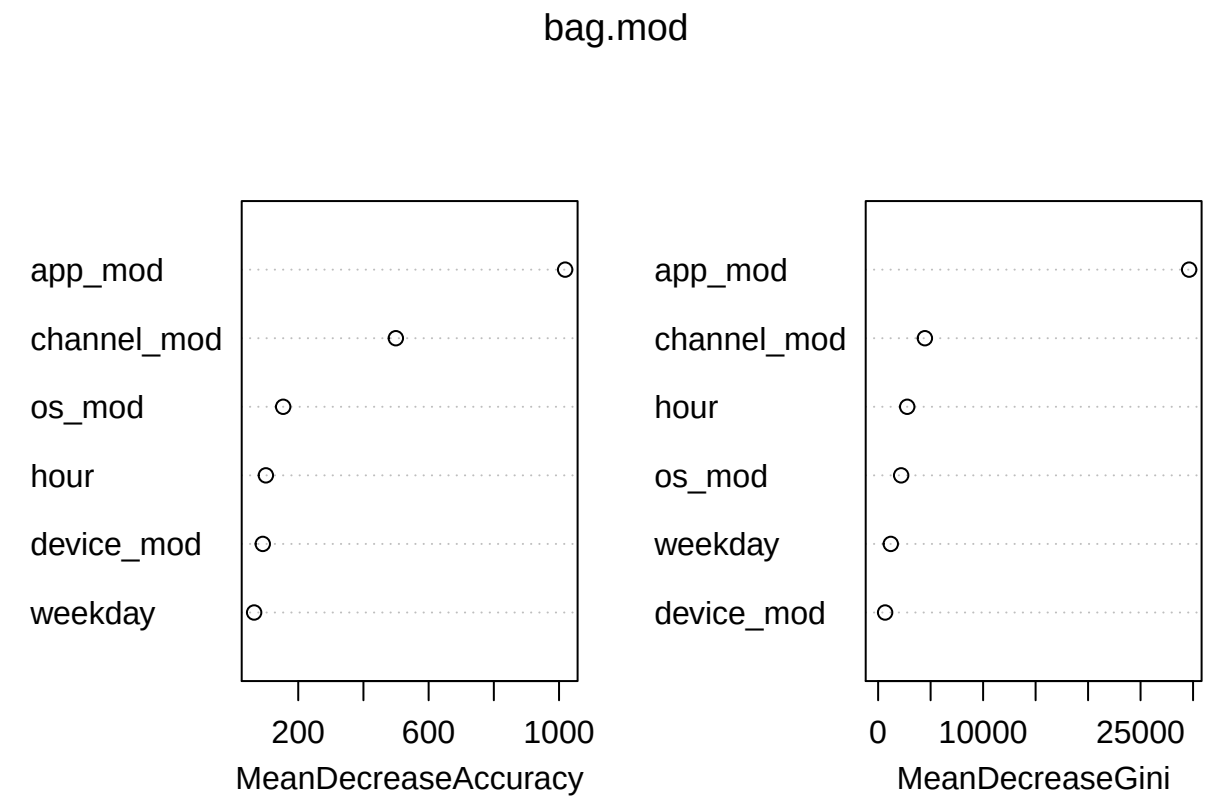
```
bag.mod <- readRDS("bag.mod.rds")
```

```
bag.mod
```

```
##
## Call:
```

```
##  randomForest(formula = as.factor(is_attributed) ~ ., data = df_treino,      importance = TRUE, mtry
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 11.87%
## Confusion matrix:
##       0     1 class.error
## 0 46398  4019  0.07971518
## 1  7850 41733  0.15832039
```

```
varImpPlot(bag.mod)
```

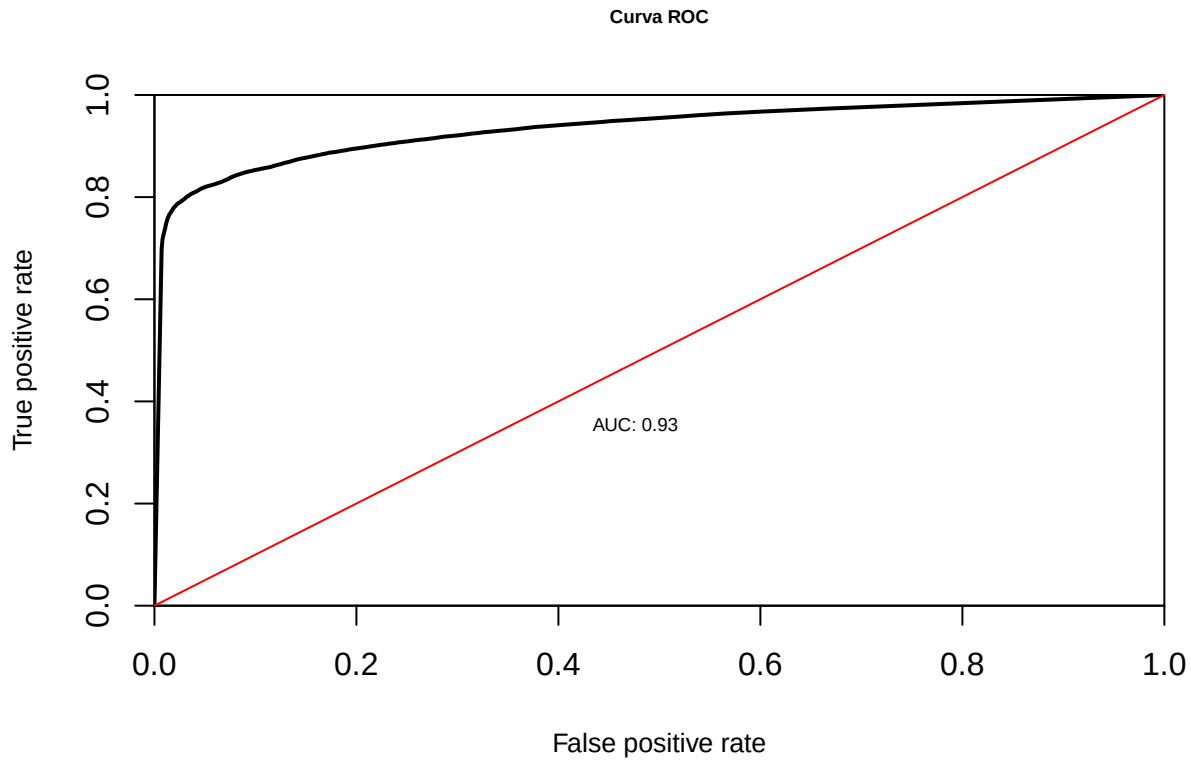



```
bag.probs <- predict(bag.mod, newdata = df_teste, type = "prob")[,2]
bag.preds <- predict(bag.mod, newdata = df_teste, type = "response")
confusionMatrix(table(observado = df_teste$is_attributed, predito = bag.preds), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##          predito
## observado      0      1
##         0 126141  10687
##         1  21886 115393
##
##                Accuracy : 0.8812
```

```
##                  95% CI : (0.8799, 0.8824)
##     No Information Rate : 0.54
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7624
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9152
##             Specificity : 0.8521
##          Pos Pred Value : 0.8406
##          Neg Pred Value : 0.9219
##              Prevalence : 0.4600
##          Detection Rate : 0.4210
##    Detection Prevalence : 0.5008
##       Balanced Accuracy : 0.8837
##
##        'Positive' Class : 1
##
```

```
predictions <- prediction(bag.probs, df_teste$is_attributed)
plot.roc.curve(predictions, title.text = "Curva ROC")
```




```
auc(roc(df_teste$is_attributed, bag.probs))
```

```
## Area under the curve: 0.9343
```

Esse modelo obteve AUC de 0.9343.

Boosting

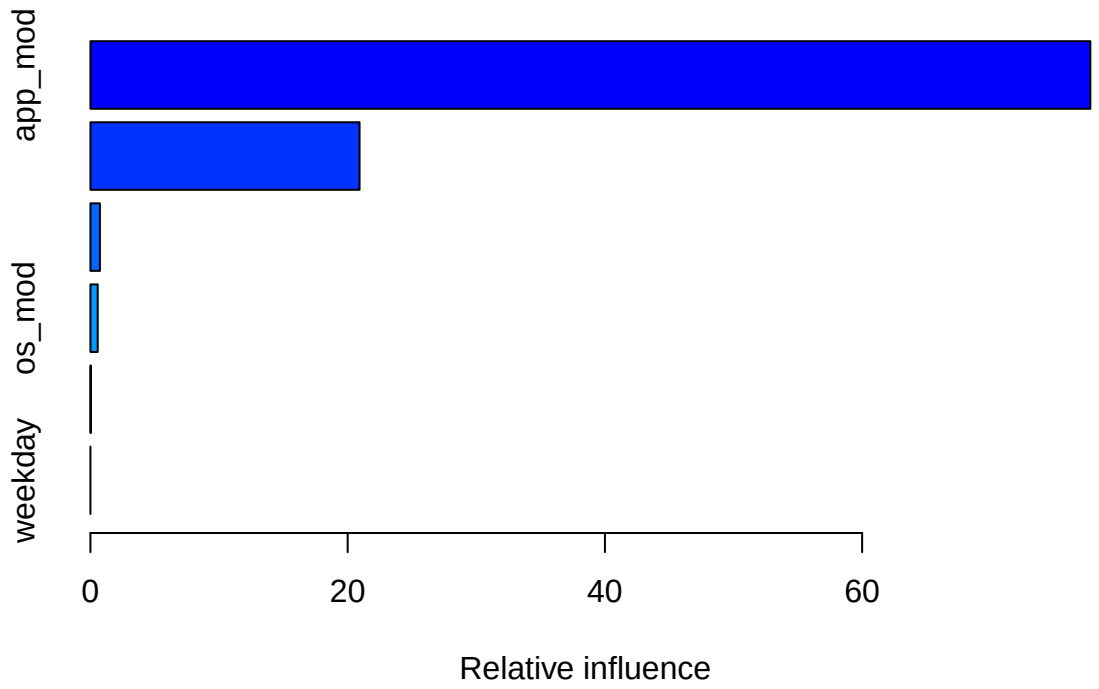
```
set.seed(600)
boost.mod <- gbm(is_attributed ~ ., data = df_treino,
                 distribution = "bernoulli", n.trees = 100)
saveRDS(boost.mod, "boost.mod.rds")
```

```
boost.mod <- readRDS("boost.mod.rds")
```

```
boost.mod
```

```
## gbm(formula = is_attributed ~ ., distribution = "bernoulli",
##     data = df_treino, n.trees = 100)
## A gradient boosted model with bernoulli loss function.
## 100 iterations were performed.
## There were 6 predictors of which 5 had non-zero influence.
```

```
summary(boost.mod)
```



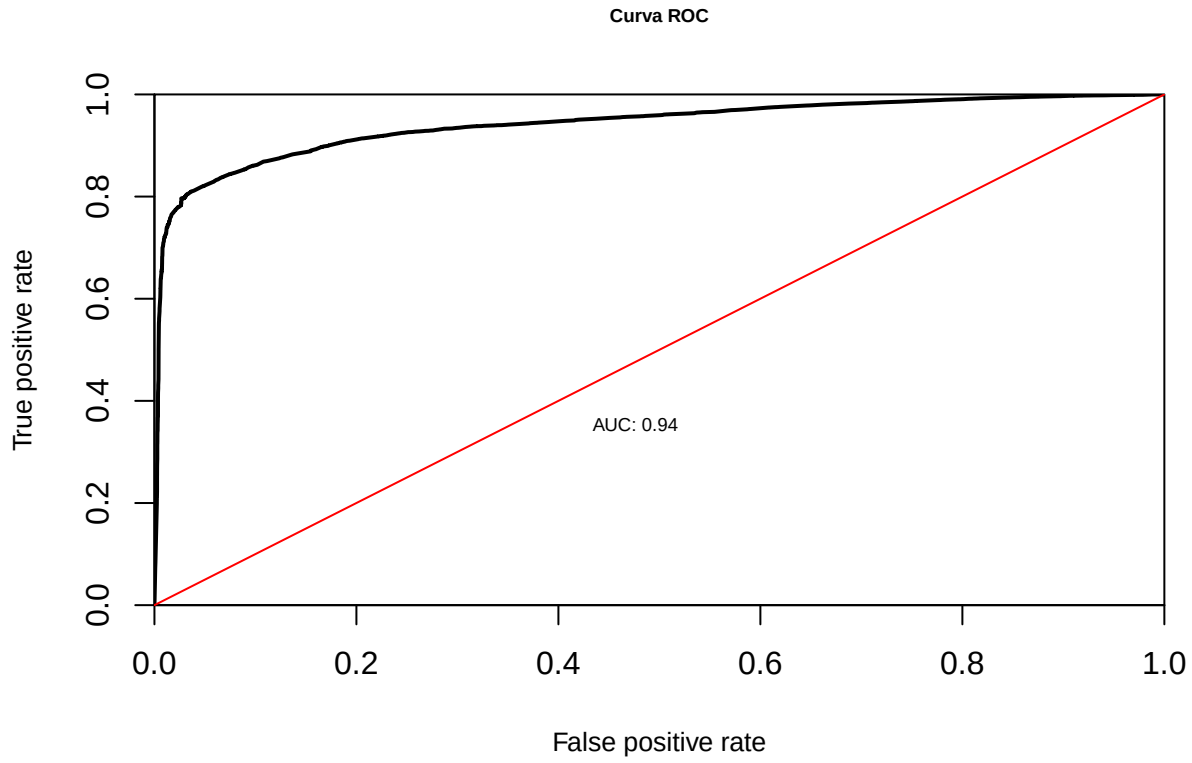

```
##                     var    rel.inf
```

```
## app_mod         app_mod 77.7434449
## channel_mod channel_mod 20.9235825
## device_mod   device_mod  0.7357831
## os_mod           os_mod  0.5669123
## hour               hour  0.0302771
## weekday         weekday  0.0000000
```

```
boost.probs <- predict(boost.mod, newdata = df_teste, n.trees = 100, type = "response")
boost.preds <- round(boost.probs)
confusionMatrix(table(observado = df_teste$is_attributed, predito = boost.preds), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##          predito
## observado      0      1
##         0 132599   4229
##         1  27276 110003
##
##                Accuracy : 0.8851
##                  95% CI : (0.8839, 0.8863)
##     No Information Rate : 0.5833
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7702
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9630
##             Specificity : 0.8294
##          Pos Pred Value : 0.8013
##          Neg Pred Value : 0.9691
##              Prevalence : 0.4167
##          Detection Rate : 0.4013
##    Detection Prevalence : 0.5008
##       Balanced Accuracy : 0.8962
##
##        'Positive' Class : 1
##
```

```
predictions <- prediction(boost.probs, df_teste$is_attributed)
plot.roc.curve(predictions, title.text = "Curva ROC")
```




```
auc(roc(df_teste$is_attributed, boost.probs))
```

```
## Area under the curve: 0.9416
```

Esse modelo obteve AUC de 0.9416.

Com isso, obteve-se a seguinte classificação de modelos mediante a AUC:

1. KNN: 0.945;
2. Regressão Logística: 0.9446;
3. Boosting: 0.9416;
4. Random Forest: 0.9398;
5. Bagging: 0.9343.

Avaliação na Competição Kaggle e Otimização

Em seguida, os cinco modelos acima foram utilizados para fazer as previsões para os dados de teste presentes no site da competição no Kaggle e enviados para avaliação, obtendo os seguintes resultados:

```
tabela <- data.frame(Modelo = c("Regressão Logística", "KNN", "Boosting", "Random Forest", "Bagging"),
                     `Private Score` = c(0.93269, 0.93104, 0.92852, 0.92194, 0.91783),
                     `Public Score` = c(0.93678, 0.92906, 0.93124, 0.92398, 0.90934))
tabela %>% kable() %>% kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive
```

Modelo	Private.Score	Public.Score
Regressão Logística	0.93269	0.93678
KNN	0.93104	0.92906
Boosting	0.92852	0.93124
Random Forest	0.92194	0.92398
Bagging	0.91783	0.90934

Com isso, conclui-se que o modelo com melhor desempenho foi o de regressão logística. Por isso, ele será o modelo otimizado. A partir dele, foram criados mais dois modelos: um contendo as mesmas variáveis explicativas, porém retirando *weekday* e o outro foi obtido retirando *weekday* e tratando *hour* como um fator.

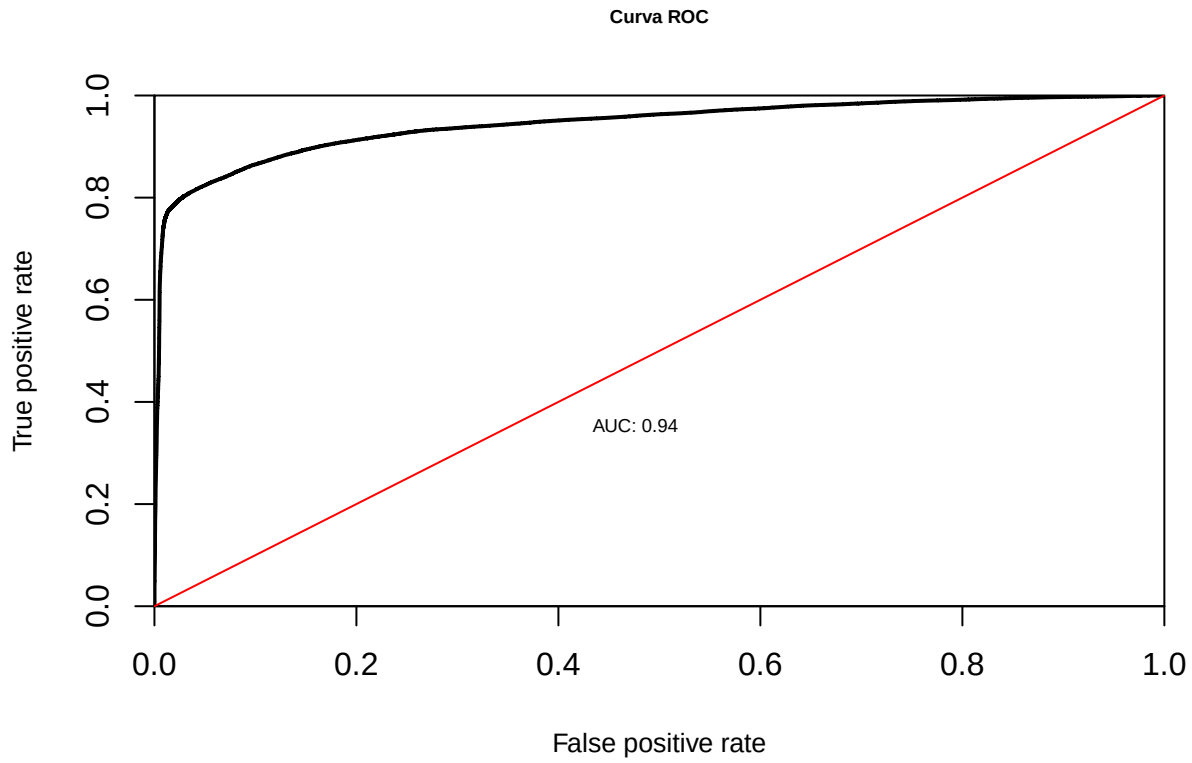
```
# Otimização Regressão Logística
# Sem weekday
reg.log2 <- glm(is_attributed ~ app_mod + device_mod + os_mod + channel_mod + hour,
                data = df_treino, family = binomial(link = logit))
saveRDS(reg.log2, "reg.log2.rds")
```

```
reg.log2 <- readRDS("reg.log2.rds")
```

```
reg.log2.probs <- predict(reg.log2, newdata = df_teste, type = "response")
reg.log2.preds <- round(reg.log2.probs)
confusionMatrix(table(observado = df_teste$is_attributed, predito = reg.log2.preds), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##          predito
## observado      0      1
##         0 132200   4628
##         1  26320 110959
##
##                Accuracy : 0.8871
##                  95% CI : (0.8859, 0.8883)
##     No Information Rate : 0.5783
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7742
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9600
##             Specificity : 0.8340
##          Pos Pred Value : 0.8083
##          Neg Pred Value : 0.9662
##              Prevalence : 0.4217
##          Detection Rate : 0.4048
##    Detection Prevalence : 0.5008
##       Balanced Accuracy : 0.8970
##
##        'Positive' Class : 1
##
```

```
predictions <- prediction(reg.log2.probs, df_teste$is_attributed)
plot.roc.curve(predictions, title.text = "Curva ROC")
```




```
auc(roc(df_teste$is_attributed, reg.log2.probs))
```

```
## Area under the curve: 0.9444
```

Esse modelo obteve AUC de 0.9444.

```
# Sem weekday e hour como fator
reg.log3 <- glm(is_attributed ~ app_mod + device_mod + os_mod + channel_mod + as.factor(hour),
                data = df_treino, family = binomial(link = logit))
saveRDS(reg.log3, "reg.log3.rds")
```

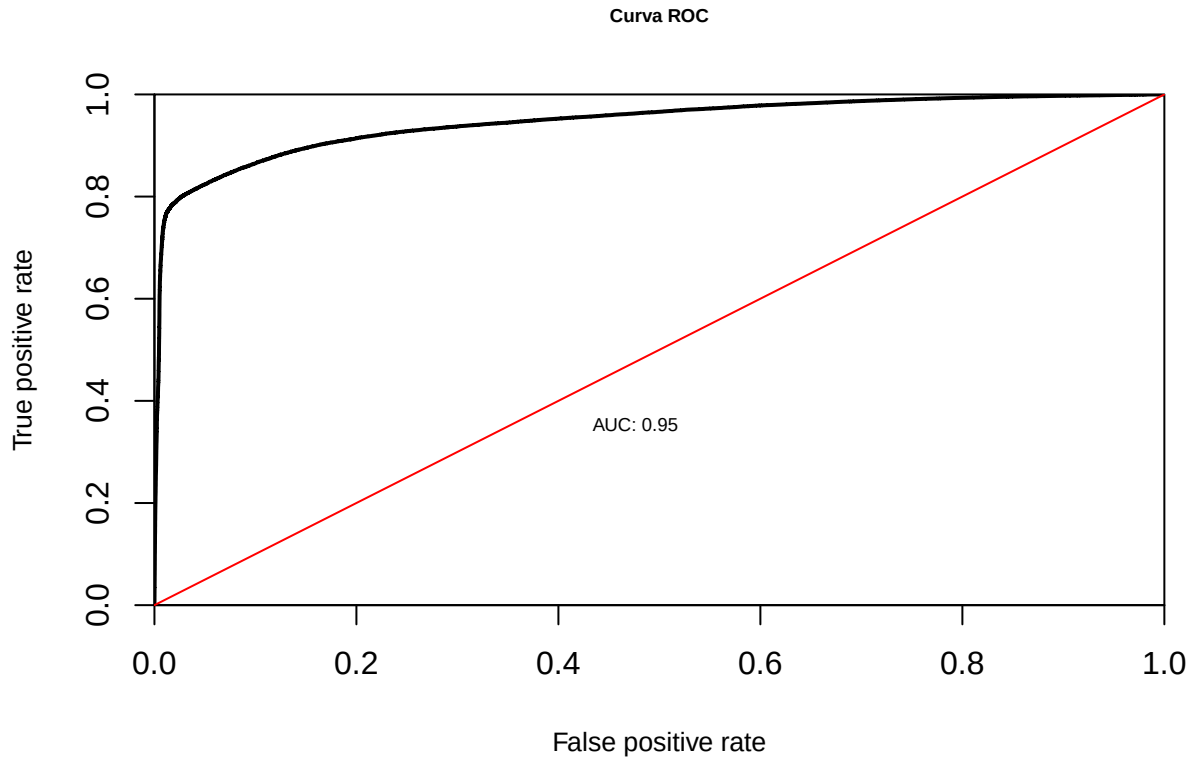
```
reg.log3 <- readRDS("reg.log3.rds")
```

```
reg.log3.probs <- predict(reg.log3, newdata = df_teste, type = "response")
reg.log3.preds <- round(reg.log3.probs)
confusionMatrix(table(observado = df_teste$is_attributed, predito = reg.log3.preds), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##          predito
```

```
## observado      0      1
##         0 131178   5650
##         1  25319 111960
##
##                Accuracy : 0.887
##                  95% CI : (0.8858, 0.8882)
##     No Information Rate : 0.5709
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7741
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9520
##             Specificity : 0.8382
##          Pos Pred Value : 0.8156
##          Neg Pred Value : 0.9587
##              Prevalence : 0.4291
##          Detection Rate : 0.4085
##    Detection Prevalence : 0.5008
##       Balanced Accuracy : 0.8951
##
##        'Positive' Class : 1
##
```

```
predictions <- prediction(reg.log3.probs, df_teste$is_attributed)
plot.roc.curve(predictions, title.text = "Curva ROC")
```




```
auc(roc(df_teste$is_attributed, reg.log3.probs))
```

```
## Area under the curve: 0.9459
```

Esse modelo obteve AUC de 0.9459.

Com isso, obteve-se a seguinte classificação de modelos mediante a AUC:

1. Regressão Logística 3: 0.9459;
2. KNN: 0.945;
3. Regressão Logística: 0.9446;
4. Regressão Logística 2: 0.9444;
5. Boosting: 0.9416;
6. Random Forest: 0.9398;
7. Bagging: 0.9343.

Esses dois modelos também foram enviados para avaliação no Kaggle, obtendo os seguintes resultados finais.

```
tabela <- data.frame(Modelo = c("Regressão Logística 3", "Regressão Logística", "Regressão Logística 2"
                     `Private Score` = c(0.93295, 0.93269, 0.93257, 0.93104, 0.92852, 0.92194, 0.91783)
                     `Public Score` = c(0.93678, 0.93678, 0.93687, 0.92906, 0.93124, 0.92398, 0.90934))
tabela %>% kable() %>% kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive
```

Modelo	Private.Score	Public.Score
Regressão Logística 3	0.93295	0.93678
Regressão Logística	0.93269	0.93678
Regressão Logística 2	0.93257	0.93687
KNN	0.93104	0.92906
Boosting	0.92852	0.93124
Random Forest	0.92194	0.92398
Bagging	0.91783	0.90934

Conclusão

Com essas informações, conclui-se que o melhor modelo para o objetivo proposto é o modelo de regressão logística sem a variável *weekday* e com *hour* tratada como fator. Esse modelo levou ao melhor valor do critério de seleção (AUC) tanto no banco de dados de teste elaborado no projeto, como na avaliação oficial da competição no site Kaggle. Por isso, ele deve ser o modelo utilizado para realizar as previsões tratadas no objetivo do projeto, pois é o modelo mais preciso.