Detecção de Fraudes em Cliques

Fábio Teixeira Trindade

15 de dezembro de 2018

Descrição do Projeto

Título: Detecção de Fraudes no Tráfego de Cliques em Propagandas de Aplicações Mobile utilizando Random Forest, Árvore de Decisão, SVM, Regressão Logística e Naive Bayes.

A TalkingData, a maior plataforma de Big Data independente da China, cobre mais de 70% dos dispositivos móveis ativos no 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 pediram a sua ajuda para 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.

Em resumo, neste projeto, você deverá construir um modelo de aprendizado de máquina para determinar se um clique é fraudulento ou não.

Os dados disponíveis são dados mascarados (somente podemos ver os códigos, não os dados reais), prática estabelecida no GDPR (Regulação de Proteção de dados Genéricos), isso deve ser adotado por todas as companhias que mantem base de dados.

Cada linha dos dados de treino contém o registro de um click, com as seguintes variáveis:

1. **ip**: endereço ip do click;
2. **app**: identificação do app para usada no marketing;
3. **device**: identificação do tipo de dispositivo do celular do usuário (por exemplo, iphone 6 plus, iphone 7, huawei mate 7, etc.);
4. **os**: versão do sistema operacional do celular;
5. **channel**: canal de quem faz a chamado do marketing no celular;
6. **click_time**: data e hora do click (UTC);
7. **attributed_time**: Se o usuário fizer o download após clicar na propaganda, essa é a hora de download do app;

8. **is_attributed**: variável target que será predita no modelo, que indica se a app foi baixada (downloaded).

Carregamento e Preparação dos Dados

```
library(lubridate)
library(caret)
library(dplyr)
library(DMwR)
library(ROSE)
library(ggplot2)
library(randomForest)
library(rpart)
library(rpart.plot)
library(data.table)
library(e1071)
library(gridExtra)
library(Amelia)
library(caTools)

train <-fread('train.csv', stringsAsFactors = FALSE, data.table = FALSE)
test <-fread('test.csv', stringsAsFactors = FALSE, data.table = FALSE)

str(train)

## 'data.frame':    100000 obs. of  8 variables:
##  $ ip             : int  87540 105560 101424 94584 68413 93663 17059
121505 192967 143636 ...
##  $ app            : int  12 25 12 13 12 3 1 9 2 3 ...
##  $ device         : int  1 1 1 1 1 1 1 1 2 1 ...
##  $ os             : int  13 17 19 13 1 17 17 25 22 19 ...
##  $ channel        : int  497 259 212 477 178 115 135 442 364 135 ...
##  $ click_time     : chr  "2017-11-07 09:30:38" "2017-11-07 13:40:27"
"2017-11-07 18:05:24" "2017-11-07 04:58:08" ...
##  $ attributed_time: chr  "" "" "" "" ...
##  $ is_attributed  : int  0 0 0 0 0 0 0 0 0 0 ...

str(test)

## 'data.frame':    100000 obs. of  7 variables:
##  $ click_id  : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ ip        : int  43570 80528 32323 42887 119289 49447 108881 36052
105475 56460 ...
##  $ app       : int  3 3 3 3 58 3 3 3 58 3 ...
##  $ device    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ os        : int  18 13 13 17 30 8 13 13 34 48 ...
##  $ channel   : int  379 379 379 379 120 379 379 379 120 379 ...
##  $ click_time: chr  "2017-11-09 14:23:39" "2017-11-09 14:23:51" "2017-
11-09 14:25:57" "2017-11-09 14:26:03" ...
```

Não há diferença entre dados de treino e teste, a não ser pela presença da variável target (is_attributed) nos dados de teste que devemos prever e da variável attributed_time (tempo levado para o download do app) que vem como 'NA' nos dados de teste).

Verificando e estimando valores missing nos dados de treino

```
colSums(is.na(train))
```

```
##               ip              app           device               os 
##                0                0                0                0 
##          channel      click_time attributed_time   is_attributed 
##                0                0                0                0
```

Não há valores missing, os dados estão limpos Attributed_time (Tempo levado para download) tem valores em branco.

```
colSums(train=='')
```

```
##               ip              app           device               os 
##                0                0                0                0 
##          channel      click_time attributed_time   is_attributed 
##                0                0            99773                0
```

Vamos verificar a variável target, quantos não foram baixados no dados de treino

```
table(train$is_attributed)
```

```
## 
##     0     1 
## 99773   227
```

Quando comparamos os valores em branco em 'Attributed_time' e a quantidade de 'is_attributed = 0' (quantidade de aplicações não baixadas) nos dados de treino, vemos que esses valores são iguais. Observa-se que a variável 'Attributed_time' não está presente nos dados de teste, portanto, não há motivos para mantê-la nos dados do treino também.

```
train$attributed_time=NULL
```

Data Munging e Feature Engineering

Feature engineering é o processo de determinar quais variáveis preditoras contribuirão para a capacidade preditiva do algoritmo de aprendizado de máquina.

Data Munging é o processo de transformar e mapear dados "bruto" em outro formato com a intenção de torná-lo mais apropriado e valioso para uma análise de dados.

Vamos utilizar esses dois conceitos, começando na conversão click_time para o formato data e hora

```
train$click_time<-as.POSIXct(train$click_time,
                             format = "%Y-%m-%d %H:%M",tz =
"America/Sao_Paulo")
```

Separando ano, mês, dia da semana e hora

```
train$year=year(train$click_time)
train$month=month(train$click_time)
train$days=weekdays(train$click_time)
train$hour=hour(train$click_time)
```

Depois de obter essas novas variáveis, vamos remover a variável original "click_time"

```
train$click_time=NULL
```

Verificando valores únicos para cada variável obtida de click_time

```
apply(train,2, function(x) length(unique(x)))
```

```
##            ip           app        device            os       channel
##         34857           161           100           130           161
## is_attributed          year         month          days          hour
##             2             1             1             4            24
```

Verificando os valores únicos, podemos ver que temos datas coletadas para um único mês, em um determinado ano, ou seja, o dados se repete a cada linha e então, não há necessidade de manter essas variáveis mês e ano.

```
train$month=NULL
train$year=NULL
```

Convertendo as variáveis "is_attributed" e "days" em variáveis do tipo fator

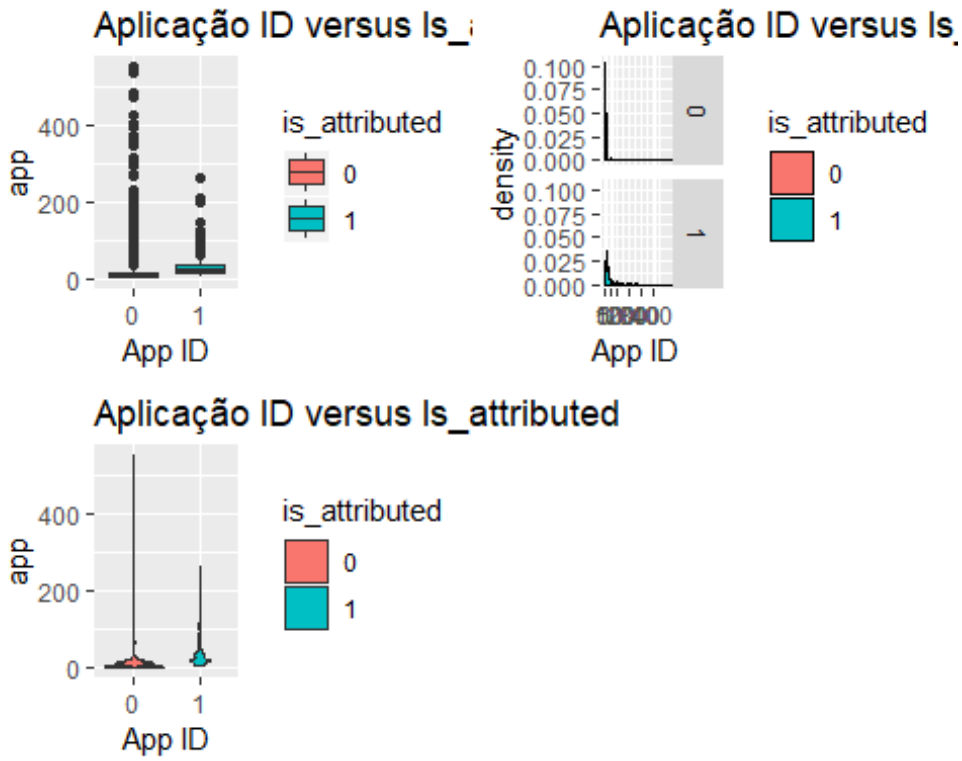
```
train$is_attributed=as.factor(train$is_attributed)
train$days=as.factor(train$days)
```

Análise exploratória de dados para verificar a importância das variáveis para a previsão

is_attributed (App foi baixado) versus App_id para marketing

```
p1=ggplot(train,aes(x=is_attributed,y=app,fill=is_attributed))+
  geom_boxplot()+
  ggtitle("Aplicação ID versus Is_attributed")+
  xlab("App ID") +
  labs(fill = "is_attributed")

p2=ggplot(train,aes(x=app,fill=is_attributed))+
  geom_density()+facet_grid(is_attributed~.)+
  scale_x_continuous(breaks = c(0,50,100,200,300,400))+
  ggtitle("Aplicação ID versus Is_attributed")+
  xlab("App ID") +
```

```
  labs(fill = "is_attributed")

p3=ggplot(train,aes(x=is_attributed,y=app,fill=is_attributed))+
  geom_violin()+
  ggtitle("Aplicação ID versus Is_attributed")+
  xlab("App ID") +
  labs(fill = "is_attributed")

grid.arrange(p1,p2, p3, nrow=2,ncol=2)
```

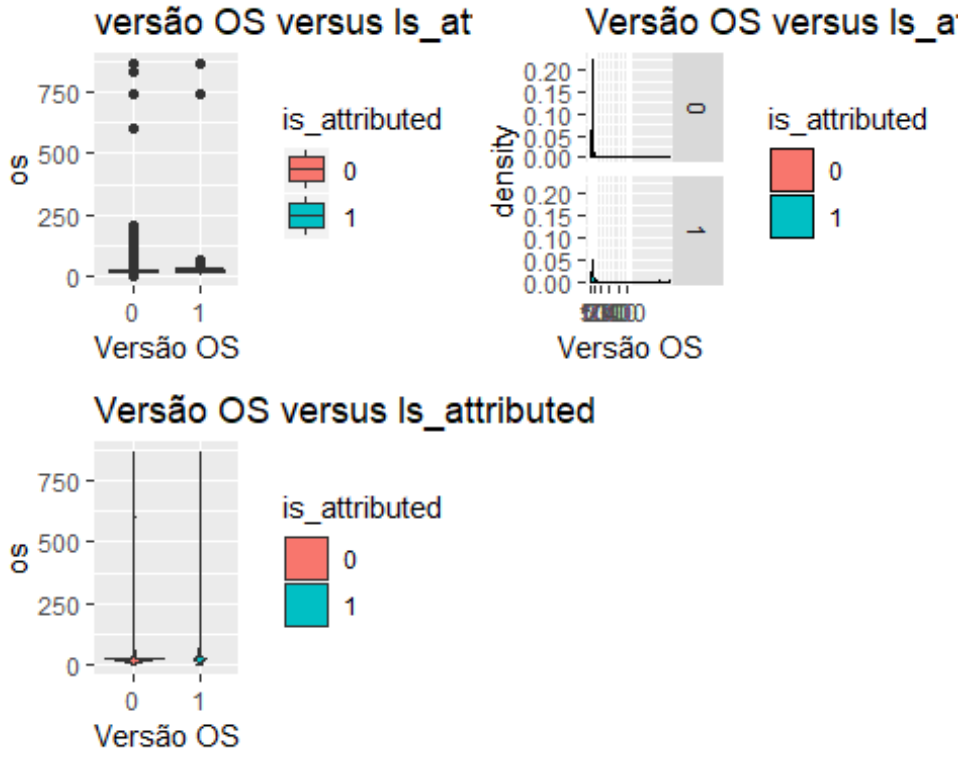


Observe o padrão e a forma diferente em todos os gráficos is_attributed (App foi baixado) versus App id no marketing, especialmente a diferenciação clara no Boxplot Isso definitivamente vai ser umas das variáveis importantes para diferenciar usuários que baixaram a aplicação ou não.

is_attributed (App foi baixada) versus versão do sistema operacional (OS) do celular:

```
p4=ggplot(train,aes(x=is_attributed,y=os,fill=is_attributed))+
  geom_boxplot()+
  ggtitle("versão OS versus Is_attributed")+
  xlab("Versão OS") +
  labs(fill = "is_attributed")


p5=ggplot(train,aes(x=os,fill=is_attributed))+
  geom_density()+facet_grid(is_attributed~.)+
  scale_x_continuous(breaks = c(0,50,100,200,300,400))+
```

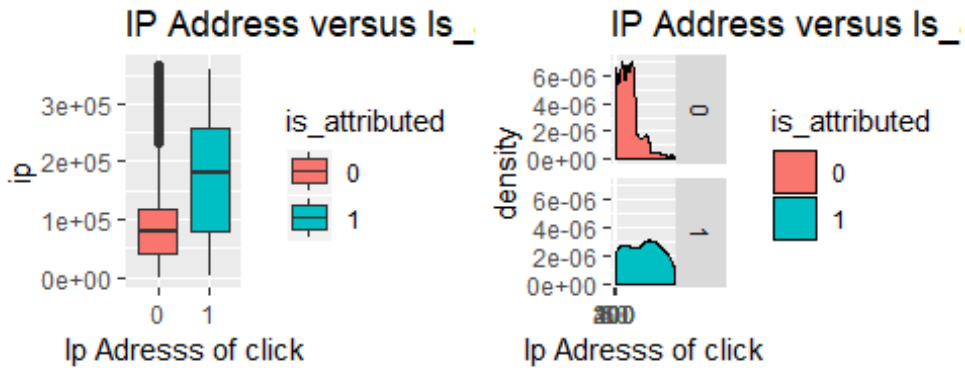
```
  ggtitle("Versão OS versus Is_attributed ")+
  xlab("Versão OS") +
  labs(fill = "is_attributed")


p6=ggplot(train,aes(x=is_attributed,y=os,fill=is_attributed))+
  geom_violin()+
  ggtitle("Versão OS versus Is_attributed")+
  xlab("Versão OS") +
  labs(fill = "is_attributed")


grid.arrange(p4,p5, p6, nrow=2,ncol=2)
```

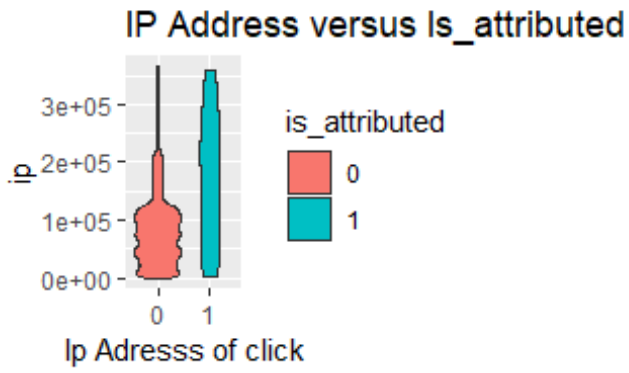


App foi baixada versus endereço ip do click.

```
p7=ggplot(train,aes(x=is_attributed,y=ip,fill=is_attributed))+
  geom_boxplot()+
  ggtitle("IP Address versus Is_attributed")+
  xlab("Ip Adresss of click") +
  labs(fill = "is_attributed")


p8=ggplot(train,aes(x=ip,fill=is_attributed))+
  geom_density()+facet_grid(is_attributed~.)+
  scale_x_continuous(breaks = c(0,50,100,200,300,400))+
  ggtitle("IP Address versus Is_attributed")+
```

```
  xlab("Ip Adresss of click") +
  labs(fill = "is_attributed")



p9=ggplot(train,aes(x=is_attributed,y=ip,fill=is_attributed))+
  geom_violin()+
  ggtitle("IP Address versus Is_attributed")+
  xlab("Ip Adresss of click") +
  labs(fill = "is_attributed")

grid.arrange(p7,p8, p9, nrow=2,ncol=2)
```

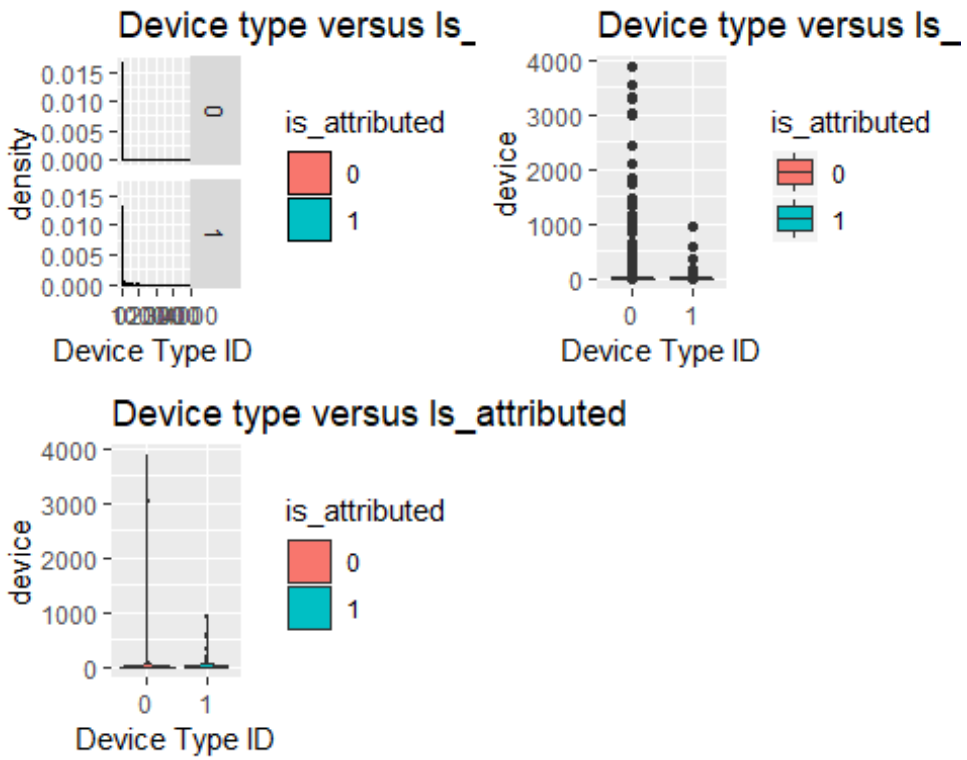




O endereço IP (IP Address) pode, muito bem, desempenhar um papel importante na previsão pois há diferenciação entre os dois grupos App foi baixada versus ID do tipo de dispositivo do usuário

```
p10=ggplot(train,aes(x=device,fill=is_attributed))+
  geom_density()+facet_grid(is_attributed~.)+
  ggtitle("Device type versus Is_attributed")+
  xlab("Device Type ID") +
  labs(fill = "is_attributed")



p11=ggplot(train,aes(x=is_attributed,y=device,fill=is_attributed))+
  geom_boxplot()+
```

```
  ggtitle("Device type versus Is_attributed")+
  xlab("Device Type ID") +
  labs(fill = "is_attributed")


p12=ggplot(train,aes(x=is_attributed,y=device,fill=is_attributed))+
  geom_violin()+
  ggtitle("Device type versus Is_attributed")+
  xlab("Device Type ID") +
  labs(fill = "is_attributed")

grid.arrange(p10,p11, p12, nrow=2,ncol=2)
```

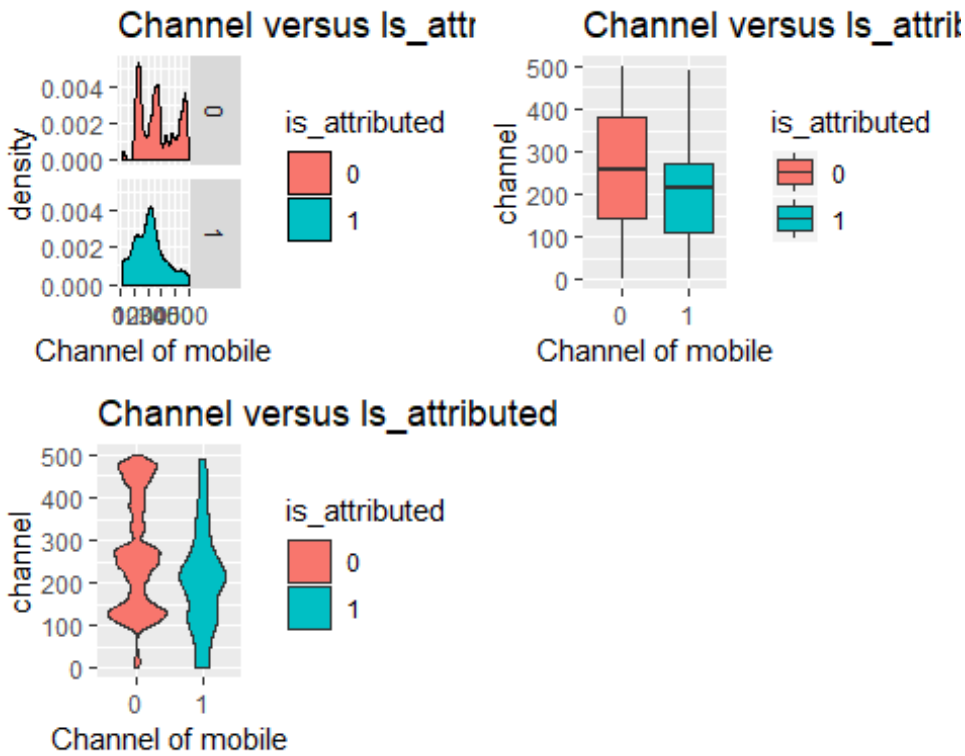


Não há diferenciação entre dispositivo (device) e is_attributed, não sendo importante para nossa análise

App foi baixada (is_attributed) versus ID do canal do editor de anúncios para celular

```
p13=ggplot(train,aes(x=channel,fill=is_attributed))+
  geom_density()+facet_grid(is_attributed~.)+
  ggtitle("Channel versus Is_attributed")+
  xlab("Channel of mobile") +
  labs(fill = "is_attributed")


p14=ggplot(train,aes(x=is_attributed,y=channel,fill=is_attributed))+
  geom_boxplot()+
```

```
  ggtitle("Channel versus Is_attributed")+
  xlab("Channel of mobile") +
  labs(fill = "is_attributed")

p15=ggplot(train,aes(x=is_attributed,y=channel,fill=is_attributed))+
  geom_violin()+
  ggtitle("Channel versus Is_attributed")+
  xlab("Channel of mobile") +
  labs(fill = "is_attributed")

grid.arrange(p13,p14, p15, nrow=2,ncol=2)
```

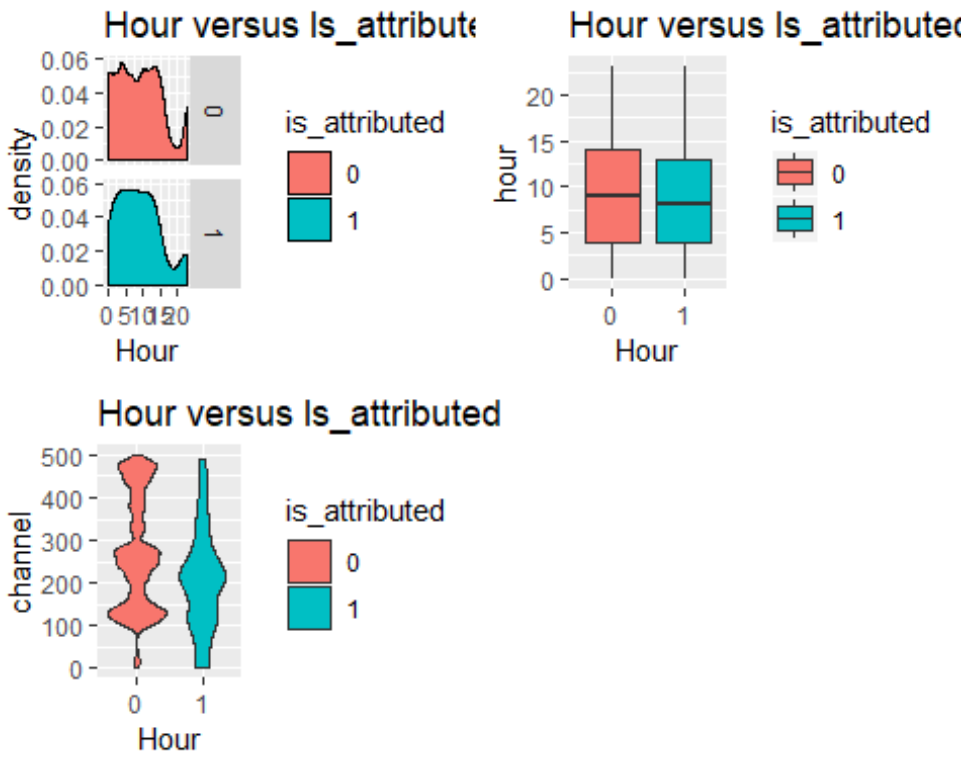


O canal do editor tem possibilidades de ajudar na previsão,podemos usar essa variável na análise de variáveis (feature)

A hora específica tem alguma relação com o download do app

```
p16=ggplot(train,aes(x=hour,fill=is_attributed))+
  geom_density()+facet_grid(is_attributed~.)+
  ggtitle("Hour versus Is_attributed ")+
  xlab("Hour") +
  labs(fill = "is_attributed")

p17=ggplot(train,aes(x=is_attributed,y=hour,fill=is_attributed))+
  geom_boxplot()+
  ggtitle("Hour versus Is_attributed")+
  xlab("Hour") +
```

```
  labs(fill = "is_attributed")

p18=ggplot(train,aes(x=is_attributed,y=channel,fill=is_attributed))+
  geom_violin()+
  ggtitle("Hour versus Is_attributed")+
  xlab("Hour") +
  labs(fill = "is_attributed")

grid.arrange(p16,p17, p18, nrow=2,ncol=2)
```

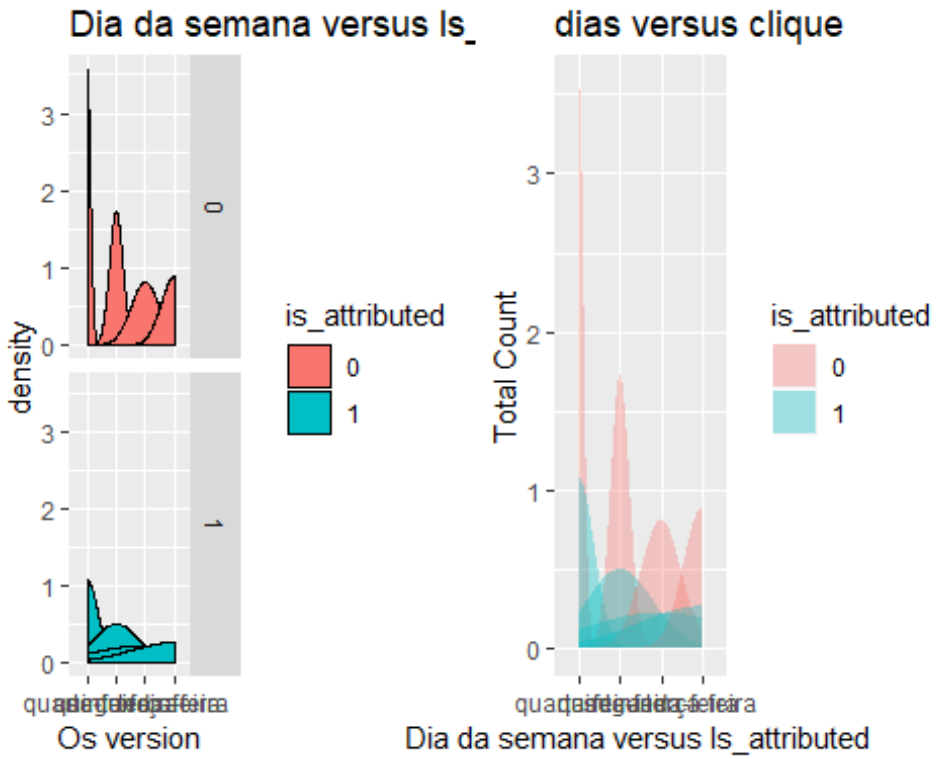


Há uma leve diferenciação em ambas as distruibuição, podemos dizer que é uma variável menos importante

Um dia específico tem algo a ver com o download da aplicação?

```
p19=ggplot(train,aes(x=days,fill=is_attributed))+
  geom_density()+facet_grid(is_attributed~.)+
  ggtitle("Dia da semana versus Is_attributed ")+
  xlab("Os version") +
  labs(fill = "is_attributed")


p20=ggplot(train,aes(x=days,fill=is_attributed))+geom_density(col=NA,alph
a=0.35)+
  ggtitle("dias versus clique")+
  xlab("Dia da semana versus Is_attributed ") +
  ylab("Total Count") +
```

```
  labs(fill = "is_attributed")

grid.arrange(p19,p20, ncol=2)
```



Parece que não há relação entre a variável dia e attributed_id

Aplicação dos Modelos

Validação sobre a análise das variáveis

1. para todas as variáveis
2. para variáveis selecionadas por meio da análise exploratória de dados

1. Modelo para todas as variáveis

Utilizando o pacote caret para particionar os dados de treino para aplicar ao modelo

```
set.seed(1234)
cv.10 <- createMultiFolds(train$is_attributed, k = 10, times = 10)
```

Utilização da validação cruzada (cross-validation) que divide os dados em 10 partes e roda o modelo 10 vezes, cada vez usando uma das partes diferentes como validação. O método repeatedcv é um bom começo.

```
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10,
                     index = cv.10)
```

Esta função configura conjuntos de dados de treino para uma série de classificações e regressões, ajusta os modelos e calcula uma medida de desempenho baseada em reamostragem (partições dos dados de treino).

```
set.seed(1234)
Model_CDT <- train(x = train[,-6], y = train[,6], method = "rpart",
tuneLength = 30,
                   trControl = ctrl)

PRE_VDTS=predict(Model_CDT$finalModel,data=train,type="class")
confusionMatrix(PRE_VDTS,train$is_attributed)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 99763   176
##          1    10    51
##
##                Accuracy : 0.9981
##                  95% CI : (0.9979, 0.9984)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 0.002836
##
##                   Kappa : 0.3535
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9999
##             Specificity : 0.2247
##          Pos Pred Value : 0.9982
##          Neg Pred Value : 0.8361
##              Prevalence : 0.9977
##          Detection Rate : 0.9976
##    Detection Prevalence : 0.9994
##       Balanced Accuracy : 0.6123
##
##        'Positive' Class : 0
##
```

Verificando a acurácia, apesar da acurácia está muito alta, mas a especificidade é muito baixa

2. Modelo para variáveis selecionadas

```
train$days=NULL
train$os=NULL
train$device=NULL

set.seed(1234)

Model_CDT1 <- train(x = train[,-4], y = train[,4], method = "rpart",
```

```
tuneLength = 30,
                    trControl = ctrl)

PRE_VDTS1=predict(Model_CDT1$finalModel,data=train,type="class")
confusionMatrix(PRE_VDTS1,train$is_attributed)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 99754   167
##          1    19    60
##
##                Accuracy : 0.9981
##                  95% CI : (0.9979, 0.9984)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 0.002836
##
##                   Kappa : 0.3914
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9998
##             Specificity : 0.2643
##          Pos Pred Value : 0.9983
##          Neg Pred Value : 0.7595
##              Prevalence : 0.9977
##          Detection Rate : 0.9975
##    Detection Prevalence : 0.9992
##       Balanced Accuracy : 0.6321
##
##        'Positive' Class : 0
##
```

Nesse segundo modelo, chega-se a mesma acurácia, no entanto há uma mudança drástica na especificidade. Então, iniciamos usando somente variáveis selecionadas para o nosso modelo atual Partição dos dados. Antes de fazer qualquer coisa, Vamos dividir os dados em dados de treino e dados de testes usando pacote caret.

```
set.seed(5000)
ind=createDataPartition(train$is_attributed,times=1,p=0.7,list=FALSE)
train_val=train[ind,]
test_val=train[-ind,]
```

Verificar a proporção:

```
round(prop.table(table(train$is_attributed)*100),digits = 3)
```

```
##
##     0     1
## 0.998 0.002
```

```
round(prop.table(table(train_val$is_attributed)*100),digits = 3)

##
##     0     1
## 0.998 0.002

round(prop.table(table(test_val$is_attributed)*100),digits = 3)

##
##     0     1
## 0.998 0.002
```

Observe que o Caret divide os dados na taxa de 70% e 30% e de que não há variação na proporção da variável target

Balanceando os dados usando o Smote

No final, verifica-se que todas as técnicas tais como up sampling, down sampling, Rose and smote, e entre esses, o Smote se sobressaiu com boa acurária

Vamos aplicar o smote e tentar equilibrar os dados:

```
set.seed(1234)
smote_train = SMOTE(is_attributed ~ ., data  = train_val)
table(smote_train$is_attributed)

##
##   0   1
## 636 477
```

Algoritmo de Aprendizado de Máquina e Validação Cruzada

Árvore de Decisão

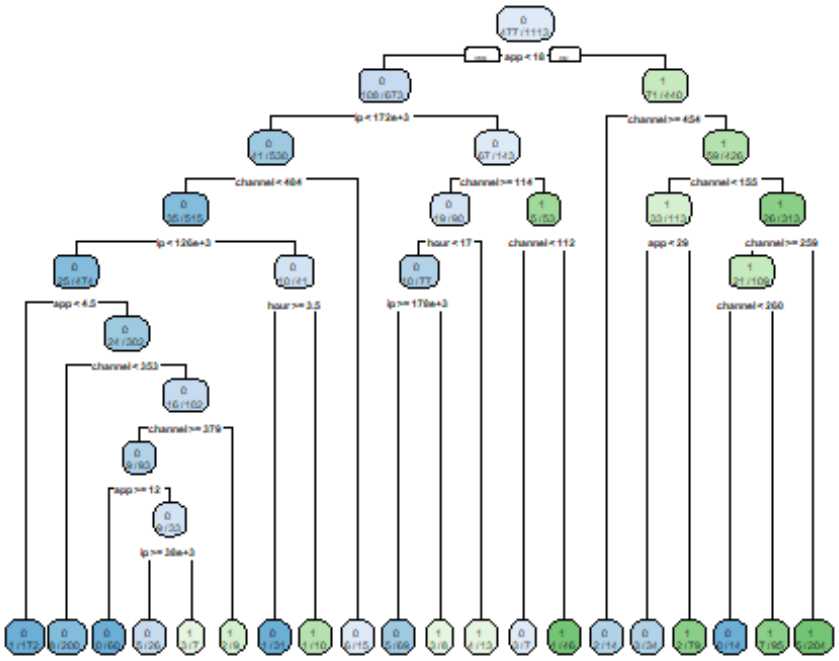
```
set.seed(1234)
cv.10 <- createMultiFolds(smote_train$is_attributed, k = 10, times = 10)
```

Controle

```
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10,
                     index = cv.10)
set.seed(1234)
```

Treina o modelo

```
Model_CDT <- train(x = smote_train[,-4], y = smote_train[,4], method =
"rpart", tuneLength = 30,
                   trControl = ctrl)

rpart.plot(Model_CDT$finalModel,extra =  3,fallen.leaves = T)
```



```
PRE_VDTS=predict(Model_CDT$finalModel,newdata=test_val,type="class")
confusionMatrix(PRE_VDTS,test_val$is_attributed)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 27554    10
##          1  2377    58
##
##                Accuracy : 0.9204
##                  95% CI : (0.9173, 0.9235)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0421
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.92058
##             Specificity : 0.85294
##          Pos Pred Value : 0.99964
##          Neg Pred Value : 0.02382
##              Prevalence : 0.99773
##          Detection Rate : 0.91850
##    Detection Prevalence : 0.91883
##       Balanced Accuracy : 0.88676
##
```

```
##        'Positive' Class : 0
##
```

Somos capazes de completar Árvore de Decisão com 0,94% de acurácia, e especificidade aumentada para 0,78% (Lembre-se, aumento drástico em especificidade depois do balanceamento dos dados)

Random forest

```
cv.10 <- createMultiFolds(smote_train$is_attributed, k = 10, times = 10)
```

Controle

```
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10,
                     index = cv.10)
set.seed(1234)
set.seed(1234)
rf.5<- train(x = smote_train[,-4], y = smote_train[,4], method = "rf",
tuneLength = 3,
             ntree = 100, trControl =ctrl)

rf.5
```

```
## Random Forest
##
## 1113 samples
##    4 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 1002, 1003, 1002, 1001, 1001, 1003, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9421422  0.8812016
##   3     0.9443848  0.8859062
##   4     0.9438418  0.8849253
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 3.
```

```
pr.rf=predict(rf.5,newdata = test_val)

confusionMatrix(pr.rf,test_val$is_attributed)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
```

```
##          0 28886     8
##          1  1045    60
##
##                Accuracy : 0.9649
##                  95% CI : (0.9628, 0.967)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0985
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9651
##             Specificity : 0.8824
##          Pos Pred Value : 0.9997
##          Neg Pred Value : 0.0543
##              Prevalence : 0.9977
##          Detection Rate : 0.9629
##    Detection Prevalence : 0.9632
##       Balanced Accuracy : 0.9237
##
##        'Positive' Class : 0
##
```

O modelo Random forest nos dá 95% accuracy, 1% melhor que decision tree, mas observe que não há muita mudança na especificidade

Support Vector Machine (SVM): Linear Support vector Machine (LSVM)

Antes de entrar no modelo, vamos ajustar o parâmetro custo (pacote e1071)

```
set.seed(1234)
liner.tune=tune.svm(is_attributed~.,data=smote_train,kernel="linear",cost
=c(0.1,0.5,1,5,10,50))

liner.tune
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.1851512
```

Vamos pegar o melhor modelo linear

```
best.linear=liner.tune$best.model
```

Dados de previsão

```
best.test=predict(best.linear,newdata=test_val,type="class")
confusionMatrix(best.test,test_val$is_attributed)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 25948    26
##          1  3983    42
##
##                Accuracy : 0.8664
##                  95% CI : (0.8625, 0.8702)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0161
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.86693
##             Specificity : 0.61765
##          Pos Pred Value : 0.99900
##          Neg Pred Value : 0.01043
##              Prevalence : 0.99773
##          Detection Rate : 0.86496
##    Detection Prevalence : 0.86583
##       Balanced Accuracy : 0.74229
##
##        'Positive' Class : 0
##
```

A Acurácia diminui no modelo Linear SVM, SVM não é um bom modelo para esses dados

Radial Support vector Machine

Vamos aplicar o SVM não linear, Radial Kernel

```
set.seed(1234)
rd.poly=tune.svm(is_attributed~.,data=smote_train,kernel="radial",gamma=seq(0.1,5))

summary(rd.poly)
```

```
##
## Parameter tuning of 'svm':
##
```

```
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma
##    4.1
##
## - best performance: 0.08984878
##
## - Detailed performance results:
##   gamma      error dispersion
## 1   0.1 0.18064672 0.05366200
## 2   1.1 0.11053732 0.03514339
## 3   2.1 0.09975869 0.03948812
## 4   3.1 0.09791667 0.03514135
## 5   4.1 0.08984878 0.03561188

best.rd=rd.poly$best.model
```

Vamos fazer previsões nos dados de teste

```
pre.rd=predict(best.rd,newdata = test_val)

confusionMatrix(pre.rd,test_val$is_attributed)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 27449    22
##          1  2482    46
##
##                Accuracy : 0.9165
##                  95% CI : (0.9133, 0.9196)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0312
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9171
##             Specificity : 0.6765
##          Pos Pred Value : 0.9992
##          Neg Pred Value : 0.0182
##              Prevalence : 0.9977
##          Detection Rate : 0.9150
##    Detection Prevalence : 0.9157
##       Balanced Accuracy : 0.7968
##
##        'Positive' Class : 0
##
```

```
pre.rd=predict(best.rd,newdata = test_val)

confusionMatrix(pre.rd,test_val$is_attributed)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 27449    22
##          1  2482    46
##
##                Accuracy : 0.9165
##                  95% CI : (0.9133, 0.9196)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0312
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9171
##             Specificity : 0.6765
##          Pos Pred Value : 0.9992
##          Neg Pred Value : 0.0182
##              Prevalence : 0.9977
##          Detection Rate : 0.9150
##    Detection Prevalence : 0.9157
##       Balanced Accuracy : 0.7968
##
##        'Positive' Class : 0
##
```

Embora o Radial faz melhor que o linear, no geral, a precisão não é boa.

Conclusão: poderíamos ter alcançado 99% de acurácia simplesmente usando os dados sem fazer “class balance”.

REGRESSÃO LOGÍSTICA

Treinando o modelo

```
log.model <- glm(formula = is_attributed ~ . , family = binomial(link =
'logit'), data = train)
```

Podemos ver que as variaveis Sex, Age e Pclass sao as variaveis mais significantes

```
summary(log.model)

##
## Call:
## glm(formula = is_attributed ~ ., family = binomial(link = "logit"),
```

```
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3216  -0.0659  -0.0488  -0.0362   4.1220
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -6.510e+00  2.075e-01 -31.380  < 2e-16 ***
## ip           1.052e-05  6.559e-07  16.043  < 2e-16 ***
## app          1.416e-02  1.174e-03  12.062  < 2e-16 ***
## channel     -4.314e-03  5.928e-04  -7.276 3.43e-13 ***
## hour        -1.386e-02  1.108e-02  -1.251    0.211
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3217.4  on 99999  degrees of freedom
## Residual deviance: 2852.7  on 99995  degrees of freedom
## AIC: 2862.7
##
## Number of Fisher Scoring iterations: 9
```

Fazendo as previsoes nos dados de teste

Split dos dados

```
set.seed(101)
split = sample.split(train$is_attributed, SplitRatio = 0.70)
```

Datasets de treino e de teste

```
dados_treino_final = subset(train, split == TRUE)
dados_teste_final = subset(test, split == FALSE)
```

Gerando o modelo com a versao final do dataset

```
final.log.model <- glm(formula = is_attributed ~ . , family =
binomial(link='logit'), data = dados_treino_final)
```

Resumo

```
summary(final.log.model)
```

```
##
## Call:
## glm(formula = is_attributed ~ ., family = binomial(link = "logit"),
##     data = dados_treino_final)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -2.2666  -0.0646  -0.0470  -0.0343   4.1701
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -6.541e+00  2.496e-01 -26.207  < 2e-16 ***
## ip           1.114e-05  7.759e-07  14.351  < 2e-16 ***
## app          1.375e-02  1.326e-03  10.365  < 2e-16 ***
## channel     -4.544e-03  7.155e-04  -6.351 2.14e-10 ***
## hour        -1.691e-02  1.331e-02  -1.270    0.204
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2253.4  on 69999  degrees of freedom
## Residual deviance: 1971.6  on 69995  degrees of freedom
## AIC: 1981.6
##
## Number of Fisher Scoring iterations: 9
```

Prevendo a acurácia

```
fitted.probabilities <- predict(final.log.model, newdata =
dados_treino_final, type = 'response')
```

Calculando os valores

```
fitted.results <- ifelse(fitted.probabilities > 0.5, 1, 0)
```

Conseguimos 99% de acurácia

```
misClasificError <- mean(fitted.results !=
dados_treino_final$is_attributed)
print(paste('Acuracia', (1-misClasificError)*100))

## [1] "Acuracia 99.7685714285714"
```

Criando a confusion matrix

```
table(dados_treino_final$is_attributed, fitted.probabilities > 0.5)

##
##     FALSE  TRUE
##   0 69838     3
##   1   159     0

confusionMatrix(factor(fitted.results),dados_treino_final$is_attributed)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 69838   159
```

```
##          1     3     0
##
##                Accuracy : 0.9977
##                  95% CI : (0.9973, 0.998)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 0.6141
##
##                   Kappa : -1e-04
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9977
##          Neg Pred Value : 0.0000
##              Prevalence : 0.9977
##          Detection Rate : 0.9977
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

Modelo NAIVE BAYES

```
nb_model <- naiveBayes(is_attributed ~ ., data = train)
```

Visualizando o resultado

```
nb_model
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##       0       1
## 0.99773 0.00227
##
## Conditional probabilities:
##    ip
## Y        [,1]      [,2]
##   0  91072.82  69624.46
##   1 171715.15 106667.46
##
##    app
## Y       [,1]     [,2]
##   0 12.00196 14.84064
```

```
##   1 32.22907 33.76940
##
##    channel
## Y       [,1]     [,2]
##   0 268.9769 129.7153
##   1 205.3656 117.7590
##
##    hour
## Y       [,1]     [,2]
##   0 9.329458 6.180442
##   1 8.947137 6.245482
```

```
summary(nb_model)
```

```
##         Length Class  Mode
## apriori 2      table  numeric
## tables  4      -none- list
## levels  2      -none- character
## call    4      -none- call
```

```
str(nb_model)
```

```
## List of 4
##  $ apriori: 'table' int [1:2(1d)] 99773 227
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ Y: chr [1:2] "0" "1"
##  $ tables :List of 4
##   ..$ ip     : num [1:2, 1:2] 91073 171715 69624 106667
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ Y : chr [1:2] "0" "1"
##   .. .. ..$ ip: NULL
##   ..$ app    : num [1:2, 1:2] 12 32.2 14.8 33.8
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ Y  : chr [1:2] "0" "1"
##   .. .. ..$ app: NULL
##   ..$ channel: num [1:2, 1:2] 269 205 130 118
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ Y      : chr [1:2] "0" "1"
##   .. .. ..$ channel: NULL
##   ..$ hour   : num [1:2, 1:2] 9.33 8.95 6.18 6.25
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ Y   : chr [1:2] "0" "1"
##   .. .. ..$ hour: NULL
##  $ levels : chr [1:2] "0" "1"
##  $ call   : language naiveBayes.default(x = X, y = Y, laplace =
laplace)
##  - attr(*, "class")= chr "naiveBayes"
```

Previsões

```
nb_test_predict <- predict(nb_model, train)
```

Confusion matrix

```
table(pred = nb_test_predict, true = train$is_attributed)
```

```
##     true
## pred     0     1
##    0 99135   201
##    1   638    26
```

```
confusionMatrix(nb_test_predict,train$is_attributed)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 99135   201
##          1   638    26
##
##                Accuracy : 0.9916
##                  95% CI : (0.991, 0.9922)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0552
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99361
##             Specificity : 0.11454
##          Pos Pred Value : 0.99798
##          Neg Pred Value : 0.03916
##              Prevalence : 0.99773
##          Detection Rate : 0.99135
##    Detection Prevalence : 0.99336
##       Balanced Accuracy : 0.55407
##
##        'Positive' Class : 0
##
```

Chega-se a uma acurácia de 99% nesse modelo!

Este trabalho foi um esforço de revisão dos capítulos do curso de R, principalmente o capítulo 8, e a composição de alguns trabalhos disponíveis no site do kaggle. Percebe-se agora a dimensão da aplicabilidade dos conceitos vistos no curso com os trabalhos do mundo profissional. Valeu a pena pesquisar a análise dos dados e a comparação entre os cinco modelos de aprendizagem de máquina para um entendimento melhor da área de Ciência de Dados.