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

Sobre

Projeto com feedback 01 do curso Big Data Analytics com R e Microsoft Azure Machine da Formação Cientista de Dados da Data Science Academy.

Dataset disponibilizado no kaggle por uma plataforma de Big Data independente da China no desafio TalkingData AdTracking Fraud Detection Challenge. Link do dataset: https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data

Objetivo: Construir um modelo de machine learning para determinar se um clique é ou não fraudulento.

Baixando pacotes necessários

```
library(kableExtra)
library(lubridate)
library(Amelia)
library(dplyr)
library(ggplot2)
library(viridis)
library(gridExtra)
library(paletteer)
library(plotrix)
library(caTools)
library(DMwR)
library(randomForest)
library(forcats)
library(e1071)
library(caret)
library(ROCR)
```

Amostra Dataset

```
df <- read.csv('train_sample.csv', sep = ",", header = TRUE, stringsAsFactors = FALSE)

head(df) %>%
    kbl(caption = 'Amostra do Dataset') %>%
    kable_paper('striped',full_width = F) %>%
    row_spec(0, bold = T) %>%
    footnote('TalkingData - Detecção de cliques fraudulentos')
```

Table 1: Amostra do Dataset

ip	app	device	os	channel	click_time	attributed_time	is_attributed
87540	12	1	13	497	2017-11-07 09:30:38		0
105560	25	1	17	259	2017-11-07 13:40:27		0
101424	12	1	19	212	2017-11-07 18:05:24		0
94584	13	1	13	477	2017-11-07 04:58:08		0
68413	12	1	1	178	2017-11-09 09:00:09		0
93663	3	1	17	115	2017-11-09 01:22:13		0

Note:
TalkingData - Detecção de cliques fraudulentos

Dicionário de dados

```
dic <- data.frame(variavel = c('ip', 'app', 'device', 'os', 'channel',
                               'click_time', 'attributed_time', 'is_attributed'),
                  descricao = c('endereço de ip do clique',
                                'id do aplicativo (fins de marketing)',
                                'id do tipo de aparelho usado pelo usuário
                                (iphone, huawei, etc.)', 'id da versão do
                                sistema operacional do usuário', 'id do
                                canal publicador', 'horário do clique',
                                'quando o usuário faz o download do
                                aplicativo após um anúncio (ou seja,
                                is_attributed = 1), esse campo indica o
                                horário do download', 'se o aplicativo foi
                                ou não baixado (variável target)'),
                  tipo = c('integer', 'integer', 'integer', 'integer',
                           'integer', 'character', 'character', 'integer'),
                  valores_permitidos = c('números inteiros', 'números
                                         inteiros', 'números inteiros',
                                         'números inteiros', 'números
                                         inteiros', 'textos', 'textos',
                                         'números inteiros'))

dic %>%
    rename('Variável' = variavel,
           'Descrição' = descricao,
           'Tipo de dado' = tipo,
           'Valores permitidos' = valores_permitidos) %>%
    kbl() %>% kable_paper('striped',full_width = F) %>%
    row_spec(0, bold = T) %>%
    column_spec(2, width = '8cm')
```

Variável	Descrição	Tipo de dado	Valores permitidos
ip	endereço de ip do clique	integer	números inteiros
app	id do aplicativo (fins de marketing)	integer	números inteiros
device	id do tipo de aparelho usado pelo usuário (iphone, huawei, etc.)	integer	números inteiros
os	id da versão do sistema operacional do usuário	integer	números inteiros
channel	id do canal publicador	integer	números inteiros
click_time	horário do clique	character	textos
attributed_time	quando o usuário faz o download do aplicativo após um anúncio (ou seja, is_attributed = 1), esse campo indica o horário do download	character	textos
is_attributed	se o aplicativo foi ou não baixado (variável target)	integer	números inteiros

Pré processamento de dados

- Retirada da coluna ip tendo em vista que é um identificador único e pode atrapalhar o modelo de machine learning futuramente

```
df$ip <- NULL
```

- Alterar colunas identificadas como categóricas, porém que se encontram em outro formato

```
fatores <- c('app', 'device', 'os', 'channel', 'is_attributed')
for (i in colnames(df)) {
    if (i %in% fatores) {
        df[,i] <- as.factor(df[,i])
    }
}
```

- Alterar colunas de data, lidas como character

```
df$click_time <- ymd_hms(df$click_time)
df$attributed_time <- ymd_hms(df$attributed_time)
```

- Criando nova coluna para análise futura

```
df$duration <- as.numeric(df$attributed_time - df$click_time)
df$duration[is.na(df$duration)] <- 0
```

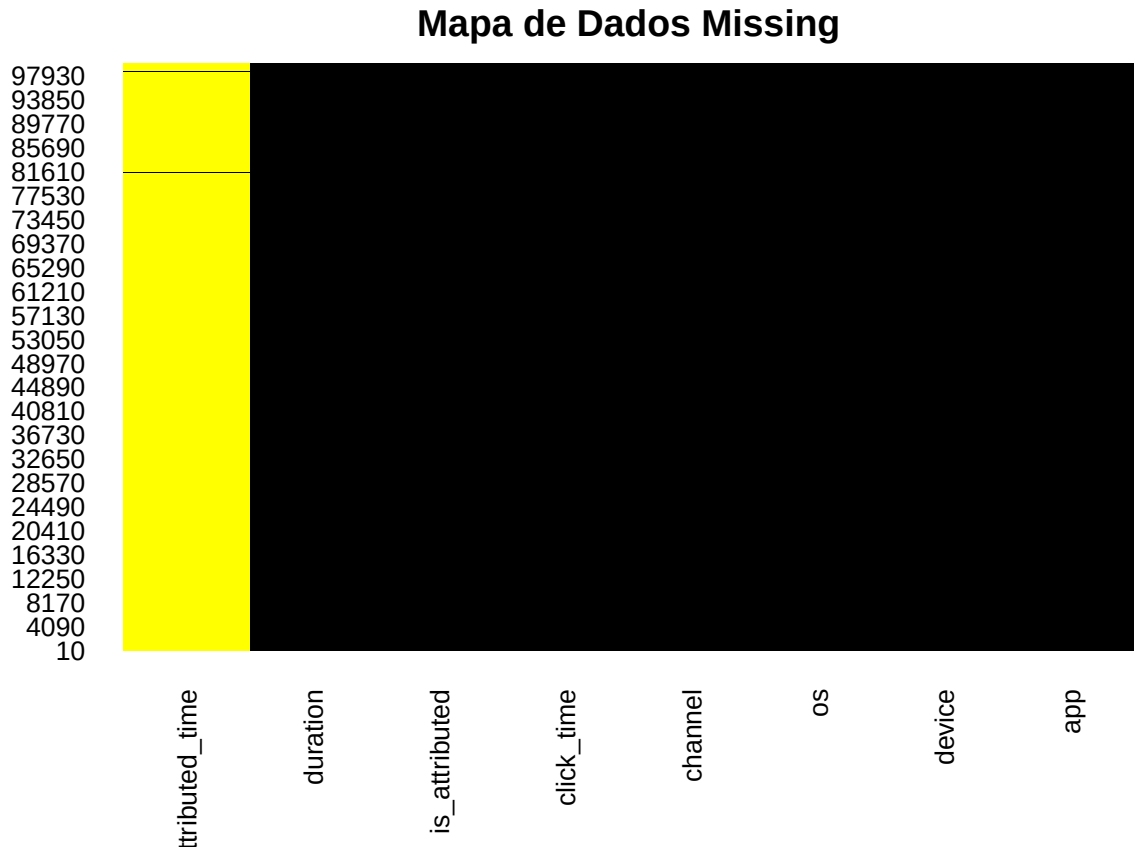
A nova coluna contém o tempo de conversão de um usuário, ou seja, o tempo em segundos que ele demorou após o clique para de fato realizar o download do app. Para não deixar dados missing no dataset, foi inerido o valor 0 (segundos) para os casos que o usuário não realizou o download.

- Verificando se o dataset possui dados faltantes

```
missmap(df,
        main = "Mapa de Dados Missing",
        col = c("yellow", "black"),
        legend = FALSE)
```

Table 2: Dataset final

app	device	os	channel	click_time	attributed_time	is_attributed	duration
12	1	13	497	2017-11-07 09:30:38	NA	0	0
25	1	17	259	2017-11-07 13:40:27	NA	0	0
12	1	19	212	2017-11-07 18:05:24	NA	0	0
13	1	13	477	2017-11-07 04:58:08	NA	0	0
12	1	1	178	2017-11-09 09:00:09	NA	0	0
3	1	17	115	2017-11-09 01:22:13	NA	0	0



Observamos missing values na coluna 'attributed_time'. Isso ocorre pois estes valores estão associados à variável 'is_attributed'. Se a última for 0, ou seja, o usuário não realizou o download do app após o anúncio, a primeira não é preenchida.

Dataset de trabalho:

```
head(df) %>%
    kbl(caption = 'Dataset final') %>%
    kable_paper('striped',full_width = F) %>%
    row_spec(0, bold = T)
```

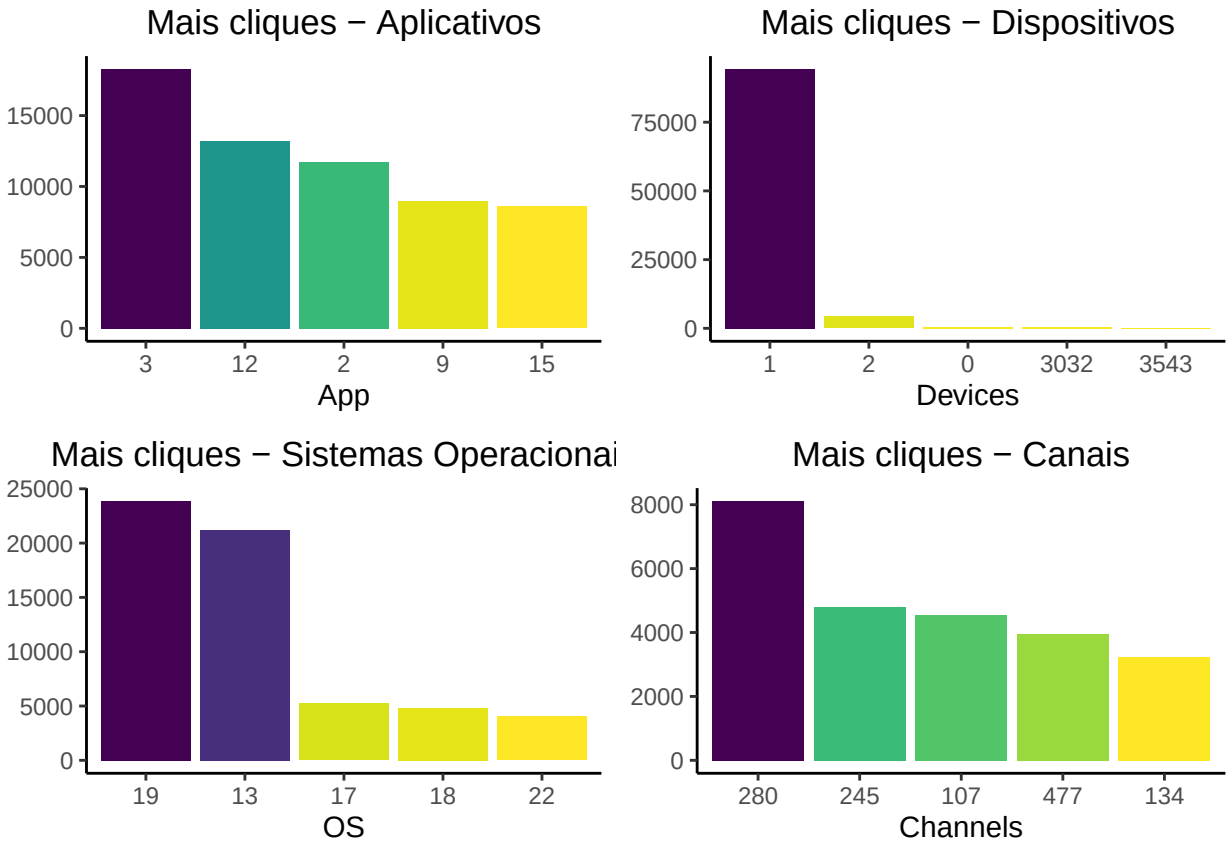
Análise Exploratória

Com o dataset tratado, vamos tirar alguns insights dos dados, entendendo quais as características dos cliques, quantos de fato se converteram em downloads e dias e horários mais badalados.

- Top cliques por categorias

```
plot_app <- df %>%
    group_by(app) %>%
    count(sort = T) %>%
    head(5) %>%
    ggplot(aes(x = reorder(app, -n), y = n, fill = -n)) +
    geom_col(show.legend = F) +
    labs(title = 'Mais cliques - Aplicativos',
         x = 'App',
         y = NULL) +
    scale_fill_viridis(option = "viridis",direction = 1) +
    theme_classic() +
    theme(plot.title = element_text(hjust = 0.5))

plot_device <- df %>%
    group_by(device) %>%
    count(sort = T) %>%
    head(5) %>%
    ggplot(aes(x = reorder(device, -n), y = n, fill = -n)) +
    geom_col(show.legend = F) +
    labs(title = 'Mais cliques - Dispositivos',
         x = 'Devices',
         y = NULL) +
    scale_fill_viridis(option = "viridis",direction = 1) +
    theme_classic() +
    theme(plot.title = element_text(hjust = 0.5))

plot_os <- df %>%
    group_by(os) %>%
    count(sort = T) %>%
    head(5) %>%
    ggplot(aes(x = reorder(os, -n), y = n, fill = -n)) +
    geom_col(show.legend = F) +
    labs(title = 'Mais cliques - Sistemas Operacionais',
         x = 'OS',
         y = NULL) +
    scale_fill_viridis(option = "viridis",direction = 1) +
    theme_classic() +
    theme(plot.title = element_text(hjust = 0.5))

plot_channel <- df %>%
    group_by(channel) %>%
    count(sort = T) %>%
    head(5) %>%
    ggplot(aes(x = reorder(channel, -n), y = n,fill = -n)) +
    geom_col(show.legend = F) +
    labs(title = 'Mais cliques - Canais',
         x = 'Channels',
```

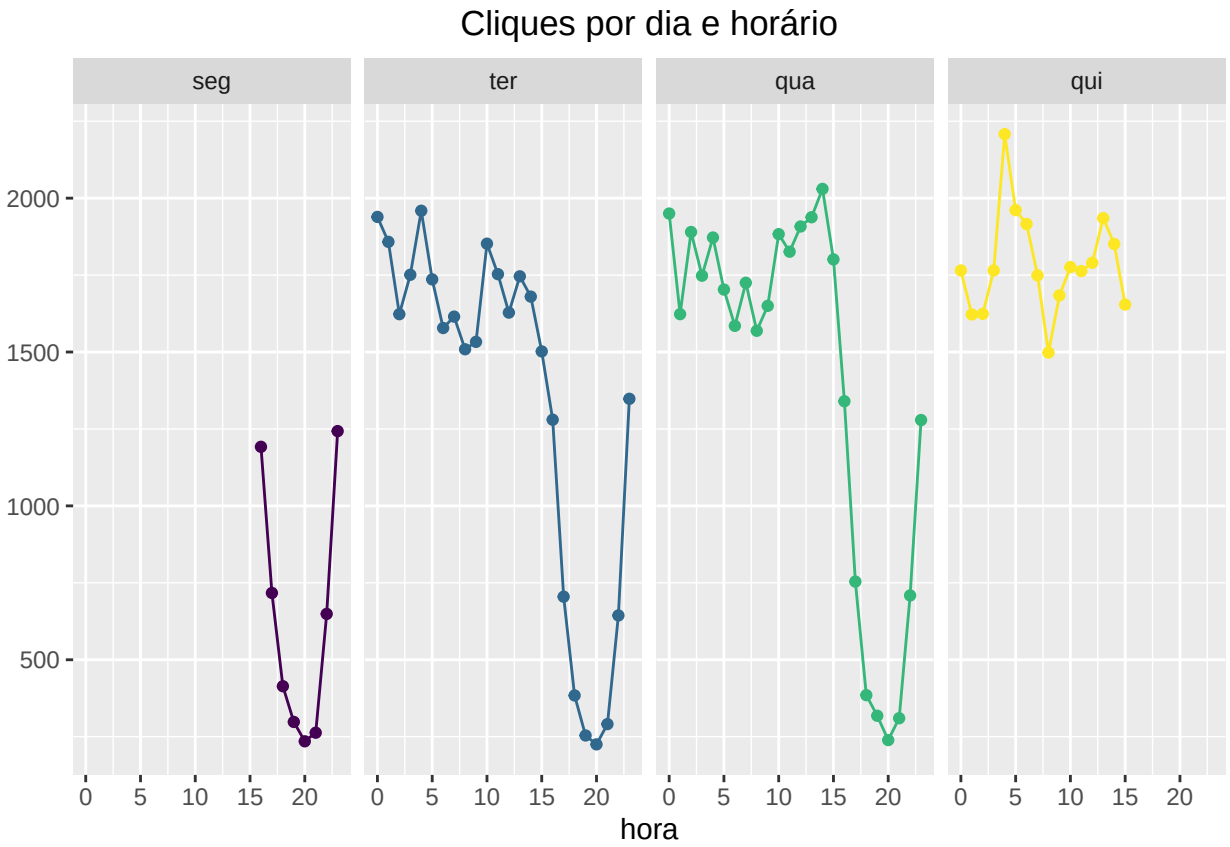
```
         y = NULL) +
    scale_fill_viridis(option = "viridis",direction = 1) +
    theme_classic() +
    theme(plot.title = element_text(hjust = 0.5))

grid.arrange(plot_app, plot_device, plot_os, plot_channel,
             ncol = 2,
             nrow = 2)
```



Nesses gráficos podemos ver os aplicativos (3 e 12), dispositivos (1 e 2), sistemas operacionais (19 e 13) e canais (280 e 245) dos quais mais vieram cliques.

- Horários mais clicados por dia

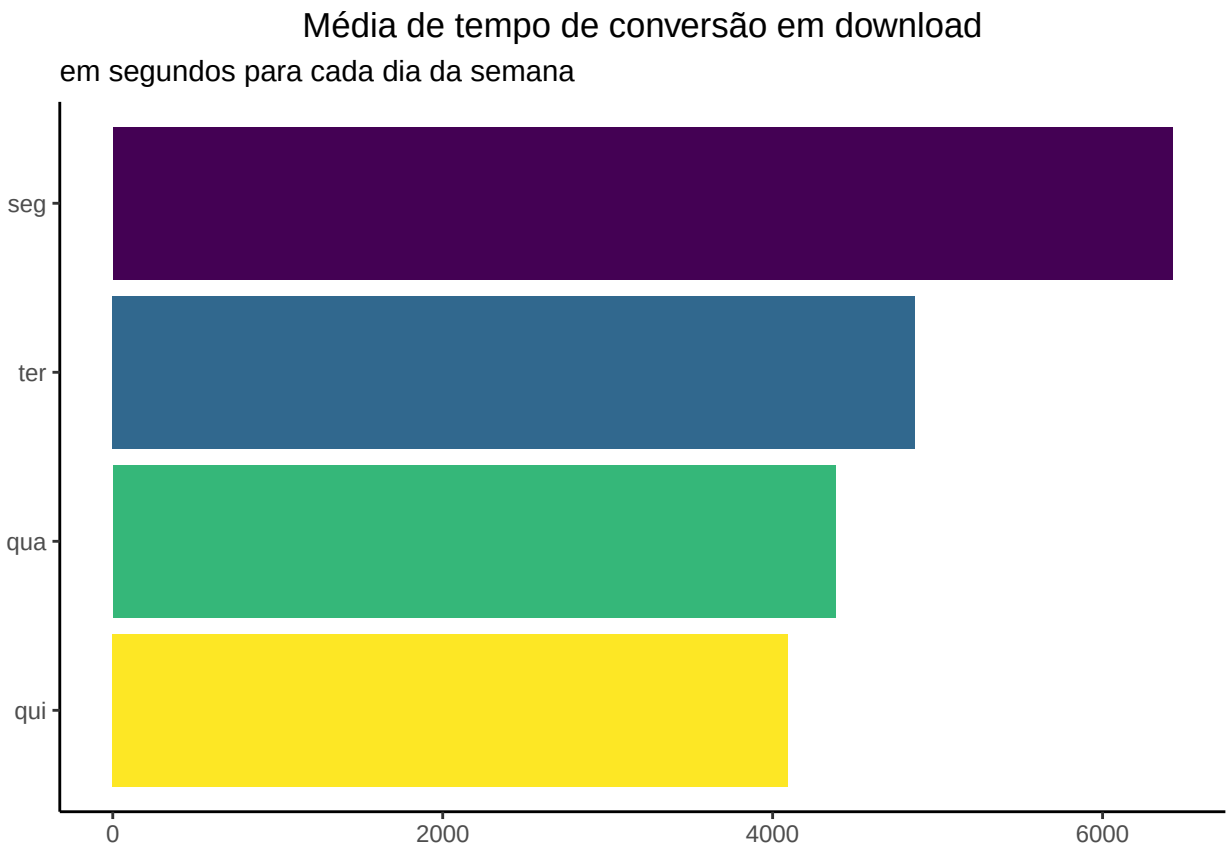
```
df %>%
  mutate(dia_da_semana = wday(click_time, label = T)) %>%
  mutate(hora = hour(click_time)) %>%
  group_by(hora, dia_da_semana) %>%
  count() %>%
  ungroup() %>%
  ggplot(aes(x = hora, y = n, color = dia_da_semana)) +
  geom_point(show.legend = F) + geom_line(show.legend = F) +
  facet_grid(~ dia_da_semana) +
  labs(title = 'Cliques por dia e horário',
       y = NULL) +
  theme(plot.title = element_text(hjust = 0.5))
```



Observamos que a maior parte dos cliques é realizada nas primeiras horas do dia (terça, quarta e quinta às 4h), enquanto ao final do dia, menos cliques ocorrem (segunda, terça e quarta às 20h).

- Tempo de conversão do usuário

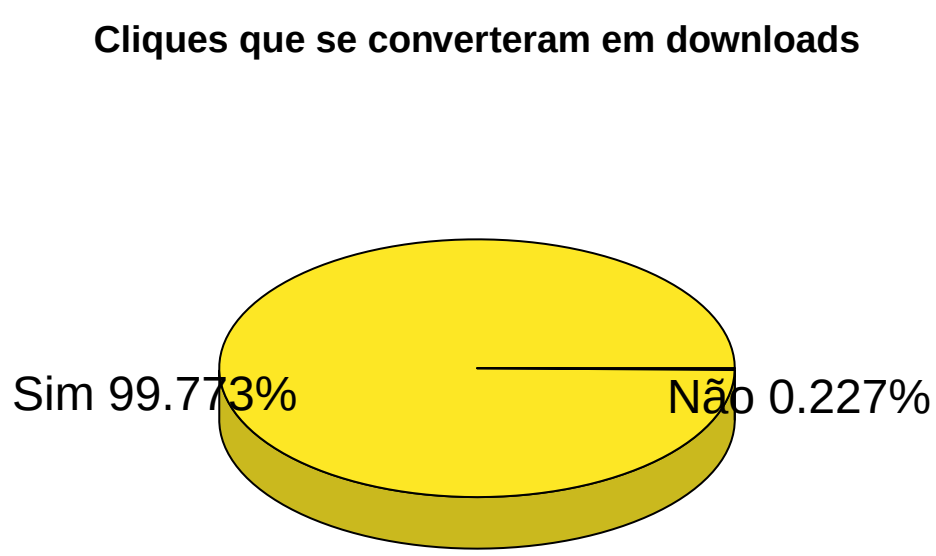
```
df %>%
  filter(is_attributed == 1) %>%
  mutate(dia_da_semana = wday(attributed_time, label = T)) %>%
  group_by(dia_da_semana) %>%
  summarise(media = mean(duration)) %>%
  ungroup() %>%
  ggplot(aes(x = reorder(dia_da_semana, media), y = media, fill = dia_da_semana)) +
  geom_col(show.legend = F) + coord_flip() +
  labs(title = 'Média de tempo de conversão em download',
       subtitle = 'em segundos para cada dia da semana',
       x = NULL, y = NULL) +
  theme_classic() +
  theme(plot.title = element_text(hjust = 0.5))
```



Pecebemos que o usuário demora mais para realizar um download às segundas, podendo ser uma indicação de necessidade de ações de marketing para esse dia especificamente.

- Downloads

```
cores_2 <- as.vector(paletteer_c("viridis::viridis", n = 2, direction = -1))

tabela <- table(df$is_attributed)
tabela <- prop.table(tabela) * 100

download <- c('Sim', 'Não')
labels <- paste(download, tabela)
labels <- paste(labels, '%', sep = '')

pie3D(tabela, labels = labels,
      col = cores_2,
      main = 'Cliques que se converteram em downloads',
      theta = pi/4)
```



Notamos aqui a baixa porcentagem de registros de cliques que efetivamente realizaram download. Isso significa que nosso dataset não está equilibrado e vai gerar problemas futuramente no processo de machine learning, fazendo com que durante o treinamento, o algoritmo aprenda mais sobre dados fraudulentos que não fraudulentos.

Porcesso de Machine Learning

Após tratamento e análise exploratória dos dados, vamos iniciar o procedimento de aprendizado de máquina para determinar se um clique é ou não fraudulento.

- Separando dados de treino e teste

Para treinar nosso algoritmo, vamos separar 70% do nosso dataset, deixando 30% para fazer os testes. Vamos fazer isso com a função sample.split do pacote caTools

```
df_split <- df[, -c(5, 6, 8)]

indice <- sample.split(df_split$is_attributed, SplitRatio = .7)

treino <- subset(df_split, indice == T)
teste <- subset(df_split, indice == F)
```

obs: as colunas de datas precisaram ser retiradas para que o balanceamento pudesse ser feito posteriormente.

- Balanceamento dos dados

Table 3: Desbalanceamento - Treino e Teste

	treino	**teste**
is_attributed = 0 (apps não baixados)	99.7729 %	99.7733 %
is_attributed = 1 (apps baixados)	0.2271 %	0.2267 %

Table 4: Dataset Treino Balanceado

	Proporção
is_attributed = 0 (apps não baixados)	57.1429 %
is_attributed = 1 (apps baixados)	42.8571 %

Conforme observamos na análise exploratória, existe uma grande discrepância entre a existência de dados de cliques fraudulentos em detrimento de dados de cliques não fraudulentos (99,8% e 0,2%, respectivamente). Neste sentido, vamos utilizar uma ténica chamada SMOTE para balancear os dados e evitar modelos tendenciosos.

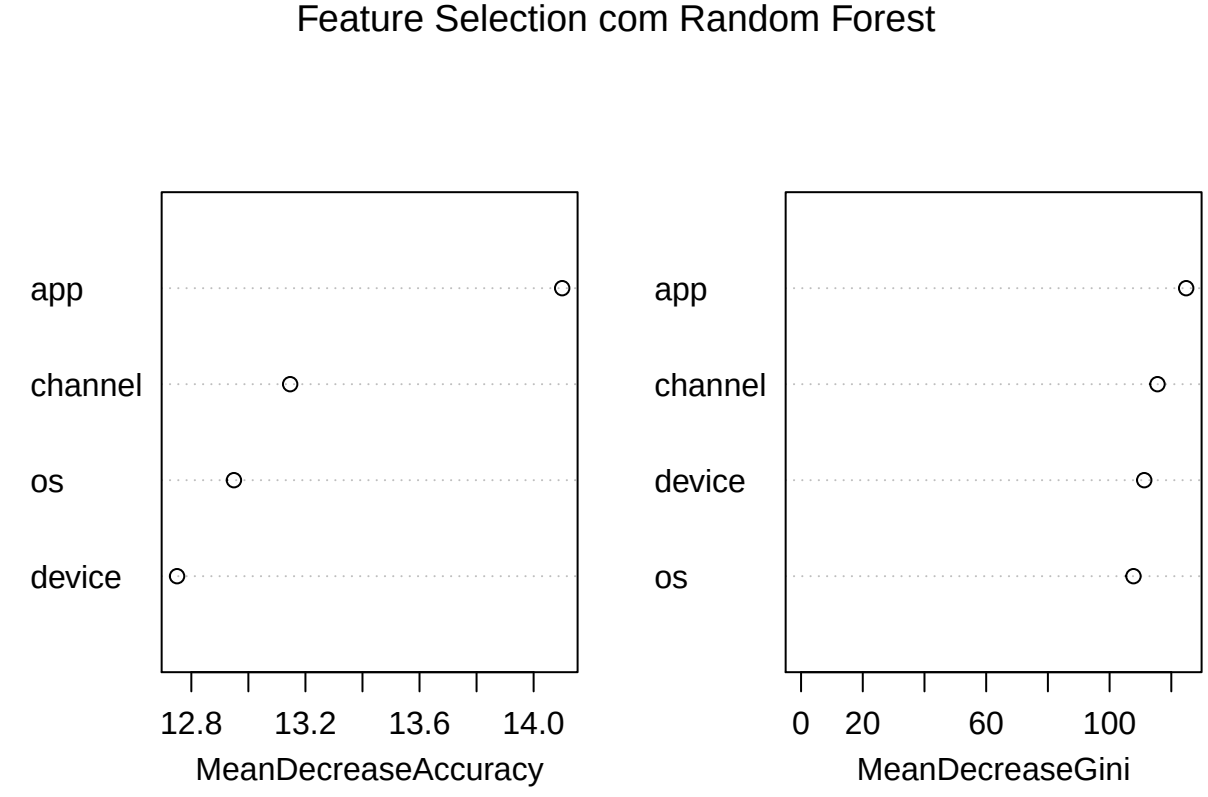
```
d_treino <- as.vector(round(prop.table(table(treino$is_attributed)) * 100 ,4))
d_teste <- as.vector(round(prop.table(table(teste$is_attributed)) * 100, 4))

desbalanceados <- data.frame(treino = paste(d_treino, '%'),
           teste = paste(d_teste, '%'),
           row.names = c('is_attributed = 0 (apps não baixados)',
                         'is_attributed = 1 (apps baixados)'))

desbalanceados %>%
  kbl(caption = 'Desbalanceamento - Treino e Teste') %>%
    kable_paper('striped',full_width = F) %>%
    row_spec(0, bold = T)
```

```
treino_smote <- SMOTE(is_attributed ~ ., data = treino)

b_treino <- as.vector(round(prop.table(table(treino_smote$is_attributed)) * 100, 4))

balanceados <- data.frame(proporcao = paste(b_treino, '%'),
                          row.names = c('is_attributed = 0 (apps não baixados)',
                                        'is_attributed = 1 (apps baixados)'))

balanceados %>%
  rename('Proporção' = proporcao) %>%
  kbl(caption = 'Dataset Treino Balanceado') %>%
    kable_paper('striped',full_width = F) %>%
    row_spec(0, bold = T)
```

- Feature Selection

Agora que os dados já foram separados em treino e teste e já encontram-se balanceados, precisamos identificar as variáveis mais relevantes para que o algoritmo de machine learning proceda da melhor maneira possível. Para isso, vamos utilizar o algoritmo Random Forest e visualizar as variáveis mais relevantes em um gráfico de acordo com dois métodos (Accuracy e Gini):

```r
aux1 <- treino_smote %>%
  select(-is_attributed) %>%
  mutate(app = fct_lump(app, n = 40),
         device = fct_lump(app, n = 40),
         os = fct_lump(app, n = 40),
         channel = fct_lump(app, n = 40))

aux2 <- treino_smote %>%
  select(is_attributed)

treino_fs <- cbind(aux1, aux2)

set.seed(1234)
modelo_fs <- randomForest(is_attributed ~ .,
                          data = treino_fs,
                          importance = T)
varImpPlot(modelo_fs,
           main = 'Feature Selection com Random Forest')
```



obs: antes de rodar o modelo, reduzi os levels das variáveis fatores tendo em vista que o radom forest não roda com variáveis categóricas que contenham mais de 53 níveis. Foi necessário, ainda, reduzir um pouco mais q auntidade de categorias (de 53 para 40) para que a função pudesse fazer os agrupamentos de forma correta.

Notamos que os dois métodos colocam a variável app como bastante relevante para o modelo, seguido da channel. Para o método Accuracy, as variáveis OS e device não parecem ser tão importantes, enquanto para

o Gini, apesar de ocuparem o terceiro e quato lugar no ranking, demonstram maio participação para boa performance do algoritmo.

Seguindo para a construção do modelo e tendo em vista que o exposto acima, vou seguir com o algoritmo apenas com as variáveis app, channel e OS.

- Criando o modelo

Escolhi usar o algoritmo naive Bayes:

```
set.seed(1234)
modelo <- naiveBayes(is_attributed ~ app
                     + channel
                     + os,
                     data = treino_smote)

previsao <- predict(modelo, teste)
```

- Avaliando o modelo

Com a função de matriz de confusão do pacote caret, analisei se o modelo teve ou não boa performance

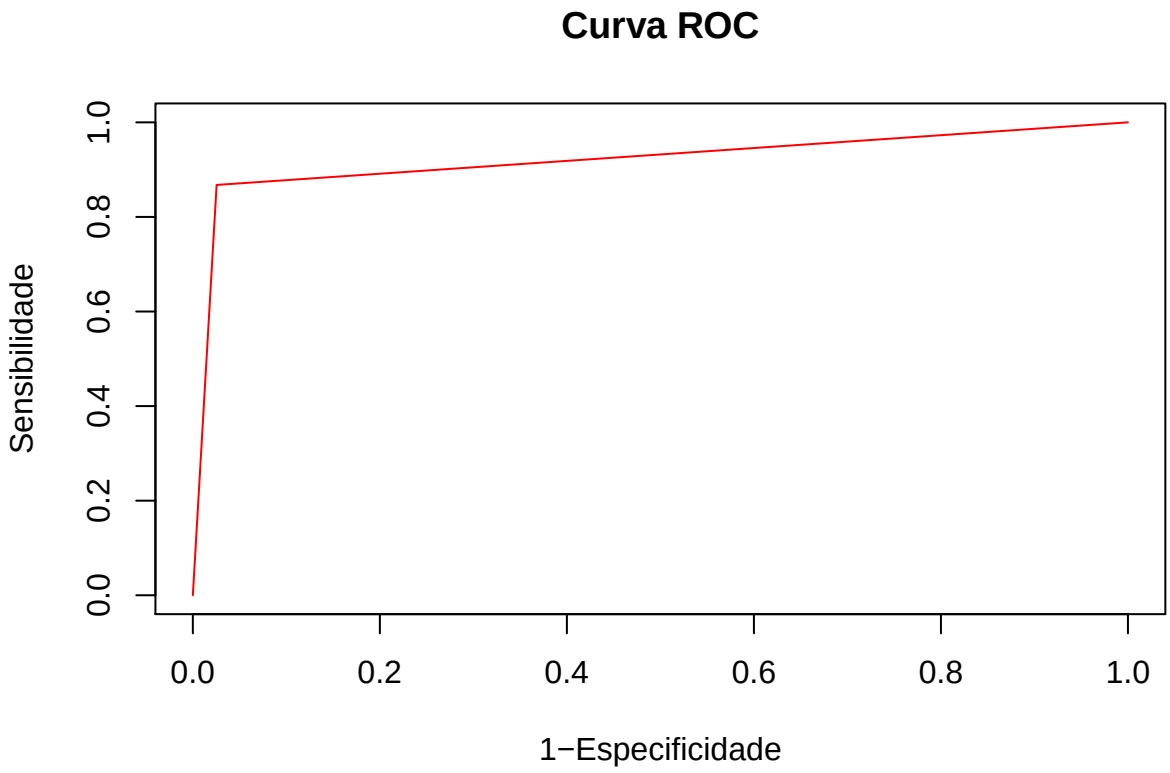
```
confusionMatrix(teste$is_attributed, previsao)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 29171   761
##          1     9    59
##
##                Accuracy : 0.9743
##                  95% CI : (0.9725, 0.9761)
##     No Information Rate : 0.9727
##     P-Value [Acc > NIR] : 0.03881
##
##                   Kappa : 0.1292
##
##  Mcnemar's Test P-Value : < 2e-16
##
##             Sensitivity : 0.99969
##             Specificity : 0.07195
##          Pos Pred Value : 0.97458
##          Neg Pred Value : 0.86765
##              Prevalence : 0.97267
##          Detection Rate : 0.97237
##    Detection Prevalence : 0.99773
##       Balanced Accuracy : 0.53582
##
##        'Positive' Class : 0
##
```

Na matriz de confusão podemos observar número alto de verdadeiros positivos e negativos, bem como ótima acurácia.

Para ilustrar, podemos ainda analisar a curva ROC (Receiver Operating Characteristic)

```
pred <- prediction(as.numeric(previsao), teste$is_attributed)
perf <- performance(pred, "tpr","fpr")
plot(perf, col = 'red',
     main = 'Curva ROC',
     ylab = 'Sensibilidade',
     xlab = '1-Especificidade')
```



Analisando...

Objetivo = detectar o máximo possível de verdadeiros positivos, enquanto minimiza os falsos positivos

Taxa verdadeiro positivo = sensibilidade = quantas vezes o modelo acertou para a opção positiva do classificador (0 - não fez o download, clique fraudulento)

Taxa falso positivo = especificidade (1) = quantas vezes o modelo errou, classificando como clique fraudulento, porém, era um download realizado

Quanto maior a taxa de verdadeiro positivo e menor a taxa de falso positivo, melhor para o modelo. Ou seja: o canto superior esquerdo é ponto ótimo.

Como a acurácia do modelo foi alta e vimos a representação da alta performance através do gráfico da curva ROC, optei por não treinar novamente o modelo com outro algoritmo.