

Data Science Project

Click Traffic Fraud Detection in Mobile Application Advertisements

Fellipe Augusto Soares Silva

2019

Sumário

Summary	5
1 – Introduction	6
2 – Business Problem	7
3 – Dataset Used	8
1. train.csv	8
4 – Data Dictionary.....	9
5 – Libraries used	10
6 – Feature Engineering	11
7 – Exploratory Analysis I	15
1. Number of Accesses.....	16
2. Top 30 IP's.....	16
3. Top Used Devices	16
4. IP Filtering	17
5. Accesses by Hour	17
6. Accesses by Hour each Day.....	17
7. Accesses Indicator.....	17
8. Apps Downloaded or not	17
9. Percentage of Downloads	17
8 – Exploratory Analysis II	18
1. Most Accessed Time	18
2. Most Accessed Applications	19
→ App 19	19
→ App 3	20
3. Click Journey by IP.....	21
→ IP 5.348	22

→ IP 5.314	23
→ IP 53.454	24
→ IP 114.276	25
9 – Machine Learning Model Building	26
1. Cross – Validation.....	27
2. Evaluation Metric.....	28
3. Machine Learning Algorithms	29
3.1 CART	29
3.2 KNN	31
3.3 Random Forest	34
3.4 Choosing the Best Machine Learning Model	37
4. Making Predictions and Evaluating the Predictive Model	38
4.1 Confusion Matrix.....	38
10 – Final Considerations.....	40
Source Code	41

Lista de Figuras

Figura 1 - Dataset train.csv	8
Figura 2 – Data Dictionary.....	9
Figura 3 – Loading Dataset train.csv	9
Figura 4 - Loading libraries.....	10
Figura 5 - Table after feature engineering.....	12
Figura 6 - click_time before rescale	12
Figura 7 - click_time after rescale.....	13
Figura 8 - Feature Engineering Code Part I	13
Figura 9 - Feature Engineering Code Part II	14
Figura 10 - Dashboard IP Accesses.....	15
Figura 11 – Dashboard Analysis	16
Figura 12 – Most Accessed Time.....	18
Figura 13 – Most Accessed Apps - App19	19
Figura 14 - Most Accessed Apps - App03.....	20
Figura 15 - False Clicks App03	21
Figura 16 – Click Journey - IP 5.348.....	22
Figura 17 - Click Journey - IP 5.314	23
Figura 18 - Click Journey - IP 53.454	24
Figura 19 - Click Journey - IP 114.276	25
Figura 20 - Cross - Validation	27
Figura 21 – Parameterization of Cross - Validation and Accuracy.....	28
Figura 22 - CART train	29
Figura 23 – CART Modeling Status.....	30
Figura 24 - CART Model Accuracy Graph	30
Figura 25 - kNN and relation k	31
Figura 26 – Choosing Factor k.....	32
Figura 27 - kNN train.....	32
Figura 28 - kNN Modeling Status	33
Figura 29 - Accuracy Graph of the kNN Model	33
Figura 30 – Inverted tree	34
Figura 31 – randomForest.....	35
Figura 32 - Underfitting x Overfitting.....	35
Figura 33 - randomForest train	35
Figura 34 - randomForest Model Status	36
Figura 35 - Accuracy Graph of randomForest Model	36
Figura 36 - Model Accuracy	37
Figura 37 - Accuracy Variation	37
Figura 38 – Time to train models	38
Figura 39 - Confusion Matrix.....	38
Figura 40 - Confusion Matrix my model.....	39

Summary

This project was developed to detect fraudulent clicks on mobile app advertisements and to predict whether or not a user will download the app after clicking on the ad.

For this, a predictive model of supervised Machine Learning was implemented based on historical data. This data was provided by TalkingData (Chinese company) to Google LLC's Kaggle, an online community of data science and machine learning, as open data for the community to use.

With the data provided, exploratory analyzes were developed to identify higher access IPs, with more clicks on advertisements, more downloaded applications, more clicked and less downloaded applications (possible frauds), tracking IP's accesses listed by day and hour, and others analytics that provide business intelligence opportunities.

After exploring all the information provided by the data, three predictive models were implemented in training data to analyze the accuracy of the model. By selecting the best machine learning model, new (previously unknown) data was introduced into the model to test its efficiency by presenting the results through a confusion matrix.

This project was developed in R programming language and the exploratory analysis will be presented through PowerBi with interactive graphics highlighting the trail that each IP leaves when accessing mobile applications, including business opportunities for digital marketing and application advertising companies.

Palavras – Chave: Data Science, Machine Learning, R, PowerBi, Exploratory Analysis, Business Intelligence, BI, caret, dplyr, Predictive model, Accuracy, Confusion Matrix, Fraud Detection, Mobile Applications, TalkingData, Kaggle.

1 – Introduction

This project was developed in R programming language using RStudio¹ as a script² development and testing platform.

To complement the knowledge of the data and provide a better view of IP clicks, PowerBi³ was also used to create a dashboard with interactive graphics for the Business Intelligence process.

Some libraries with essential packages for code development were also used, such as caret for machine learning, dplyr for data manipulation, lubrication for date and time manipulation, scale for variable normalization, among others which will be explained in the following chapters.

Like any Data Science project, before starting the parameterization of the predictive model it is necessary to perform exploratory analysis and to know the data set. During this procedure, some information that could present problems to the predictive model was transformed into new data and, as a result, Business Intelligence analysis presented several business opportunities, also highlighted in the chapter in question.

To conclude, we will present all script items, commented line by line and presenting graphical analysis to complement the understanding of the development of this Data Science project.

The beginning of business problem solving lies in understanding the problem itself which will be presented in the following chapter.

¹ Rstudio is a free integrated development environment software for R, a programming language for graphs and statistical calculations.

² Script is a set of code instructions written in computer language to perform various functions within a program.

³ Developed and powered by Microsoft, Power BI is a Business Intelligence toolkit used to create cloud dashboards for data and business analytics.

2 – Business Problem

TalkingData⁴ founded in 2011 is a Chinese data intelligence company empowering and helping its partner companies to transform the data-driven organizational culture.

TalkingData's role in addition to developing intelligent data-based applications is also to help people and the world improve through the technological evolution of the big data industry.

With this, the company has also developed open source projects cooperating with renowned universities such as MIT Media Laboratory, Stanford Artificial Intelligence Laboratory, and Caltech Engineering and Applied Science Department.

Covering over 70% of active mobile devices across China, TalkingData has become in 7 years the largest BigData platform in the country handling 3 billion mobile app ad clicks per day where 90% are potentially fraudulent.

While the risk of fraud is everywhere, the impact of massive online ad clicks can lead to the loss of money, containing misleading potential customer data, thereby increasing the cost of advertisers and operational staff as advertiser pays a fee for each click executed, even if the app was not downloaded.

China has the largest mobile market in the world with over 1 billion devices in use every month, thus suffering from click fraud. With that in mind TalkingData is asking for an improvement in its operation by creating an algorithm that can predict whether or not a user will download an app after clicking an ad on the mobile device.

In other words, what TalkingData wants is to build a machine learning model that can conclude whether a new user is prone to click fraud, be it a robot or a person, by analyzing the portfolio's click journey signaling IP addresses with many clicks but few downloads.

In addition to generating greater reliability for advertiser companies, this project seeks to introduce business intelligence by generating opportunities and increasing profitability not only for partner companies, but also for TalkingData.

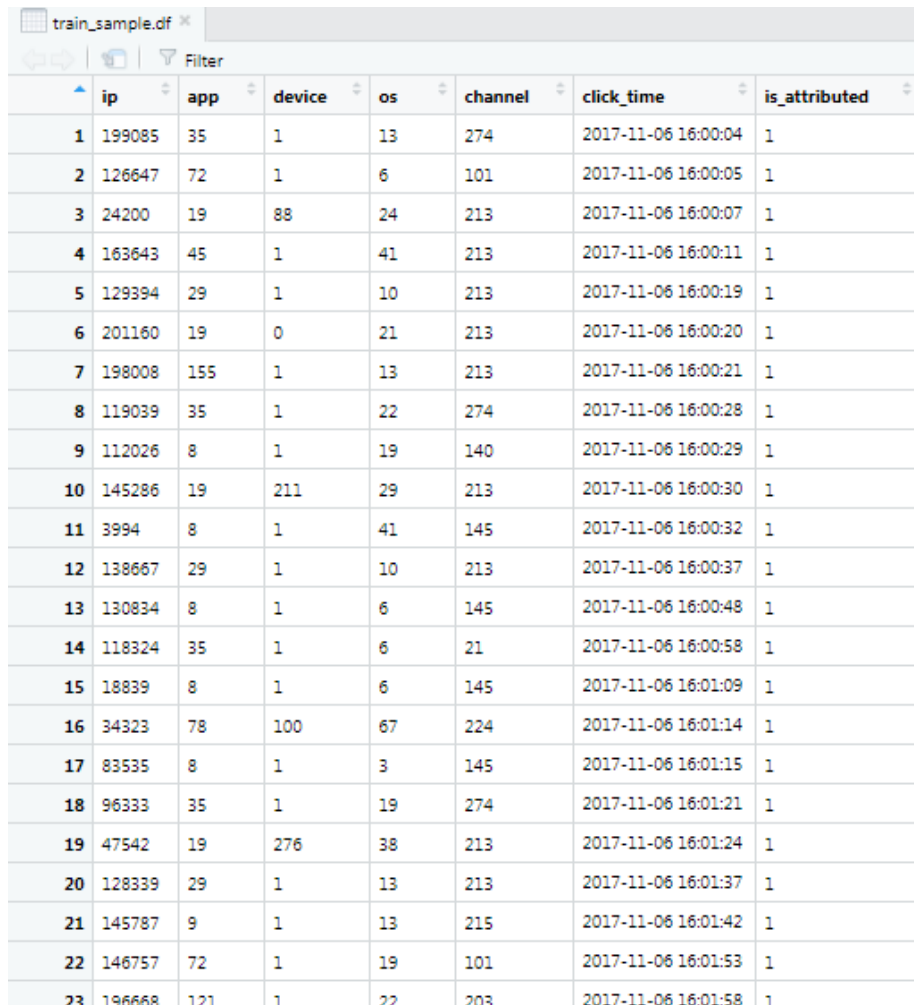
⁴ <https://www.talkingdata.com>

3 – Dataset Used

For this business problem TalkingData provided the following data set:

1. train.csv

This dataset is structured as follows:



	ip	app	device	os	channel	click_time	is_attributed
1	199085	35	1	13	274	2017-11-06 16:00:04	1
2	126647	72	1	6	101	2017-11-06 16:00:05	1
3	24200	19	88	24	213	2017-11-06 16:00:07	1
4	163643	45	1	41	213	2017-11-06 16:00:11	1
5	129394	29	1	10	213	2017-11-06 16:00:19	1
6	201160	19	0	21	213	2017-11-06 16:00:20	1
7	198008	155	1	13	213	2017-11-06 16:00:21	1
8	119039	35	1	22	274	2017-11-06 16:00:28	1
9	112026	8	1	19	140	2017-11-06 16:00:29	1
10	145286	19	211	29	213	2017-11-06 16:00:30	1
11	3994	8	1	41	145	2017-11-06 16:00:32	1
12	138667	29	1	10	213	2017-11-06 16:00:37	1
13	130834	8	1	6	145	2017-11-06 16:00:48	1
14	118324	35	1	6	21	2017-11-06 16:00:58	1
15	18839	8	1	6	145	2017-11-06 16:01:09	1
16	34323	78	100	67	224	2017-11-06 16:01:14	1
17	83535	8	1	3	145	2017-11-06 16:01:15	1
18	96333	35	1	19	274	2017-11-06 16:01:21	1
19	47542	19	276	38	213	2017-11-06 16:01:24	1
20	128339	29	1	13	213	2017-11-06 16:01:37	1
21	145787	9	1	13	215	2017-11-06 16:01:42	1
22	146757	72	1	19	101	2017-11-06 16:01:53	1
23	196668	121	1	22	203	2017-11-06 16:01:58	1

Figura 1 - Dataset train.csv

The dataset has 7.10 GB which corresponds to approximately 185 million rows of data divided into 7 variables, one of which being the predictor variable.

In the next chapter follows the Data Dictionary with the structuring of variables.

4 – Data Dictionary

Variável	Significado
ip	ip address of click
app	app id for marketing
device	device type id of user mobile phone (e.g., iphone 6 plus, iphone 7, huawei mate 7, etc.)
os	operational system version id of user mobile phone
channel	channel id of mobile ad publisher
click_time	timestamp of click (UTC)
attributed_time	time the user downloaded the app after clicking on the app advertisement
is_attributed	This is the predictor variable, the purpose of the study, indicating whether the app was downloaded or not.

Figura 2 – Data Dictionary

As noted, the dataset has a variable called “click_time” that presents unique information with the date and time the user clicked. This variable is in the Coordinated Universal Time (UTC) time zone and thus will remain for the creation of the predictive model, it will only be changed for the moments of exploratory analysis and presentation of results in PowerBi during the Business Intelligence process so that the understanding of the times with more and less acesses become clearer.

Code used to read datasets:

```
# Loading the dataset "train_sample.csv" with 100,000 rows
# Eliminating 'attributed_time' because it is the same information as 'is_attributed', avoiding Data Leakage to the predictive model
train <- "train.csv"
train.df <- fread(train, drop = c('attributed_time'))
train.df$n <- 1:nrow(train.df) # will be used to sample data and then eliminated
```

Figura 3 – Loading Dataset train.csv

The following chapters will present exploratory analysis and feature engineering adding relevant variables to the predictive model.

Next we will cover the libraries used.

5 – Libraries used

Using libraries is essential for the progress of a Data Science project. In this project the following libraries were used:

- `data.table`⁵: provides an improved version of `data.frame`;
- `dplyr`⁶: data manipulation facilitator;
- `readr`⁷: file reader;
- `caret`⁸: Machine Learning library;
- `lubridate`⁹: package with functions to manipulate dates and times;
- `scales`¹⁰: library used to scale data.

These libraries carry a series of packaged code providing more consistent and reliable results and agility as the code is ready, optimizing data analysis tasks.

Code used to load libraries:

```
## Library -----  
# IMPORTING NECESSARY LIBRARIES  
library(data.table)  
library(dplyr)  
# Using readr package  
#install.packages("readr")  
library(readr)           # Using to write_csv  
library(caret)  
library(lubridate)       # So I could work with date/time in a more intuitive syntax  
library(scales)
```

Figura 4 - Loading libraries

It is common to start a project with exploratory analysis and understand what the data provided are showing us and then draw a feature engineering strategy, but here we will reverse the order since the data has few variables and the date / time field is with more than one information in the same column, which is even be the subject of the next chapter.

⁵ <https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html>

⁶ <https://www.rdocumentation.org/packages/dplyr/versions/0.7.8>

⁷ <https://cran.r-project.org/web/packages/readr/readr.pdf>

⁸ <http://topepo.github.io/caret/index.html>

⁹ <https://cran.r-project.org/web/packages/lubridate/lubridate.pdf>

¹⁰ <https://cran.r-project.org/web/packages/scales/scales.pdf>

6 – Feature Engineering

Feature Engineering is the process of handling, adding, and removing variables. This process consists of finding out which data columns create the most useful attributes for improving the accuracy of the machine learning model.

During the loading of the variables I performed an analysis on the dataset and got an interesting result: the data distribution of the predictor variable (`is_attributed`) is totally unbalanced. This is a big problem, not now, but at the end of the Data Science process, that is, when creating the machine learning model.

Statistically we have in the variable “`is_attributed`”:

- 00.25% of data with “1” as attribute and;
- 99.75% of data with “0” as attribute.

The ideal is to generate a machine learning model that can predict both one attribute and another, but if I use the predictor variable data in this way I will generate bias and very possibly my model will fail.

In order to prevent the failure from happening I decided to assemble a training dataset that had exactly 50% of the data with “1” as an attribute and 50% of the data with “0” and after training acquire new, random and unknown data to test if the model was successful in this way. The result was very satisfactory and I describe all the steps in the following chapters.

After splitting the data back to feature engineering identifying good and bad attributes reflecting the final result, another possibility is to add relevant variables based on the data provided, which was the approach taken.

As the “`click_time`” variable has a date and time in only one piece of information, I converted this data to new variables using the `lubridate` library, namely:

- `year`
- `month`
- `day`
- `hour`
- `minutes`
- `second`
- `yday`
- `wday`
- `week`

Each new variable above, added to the dataset, corresponds to a portion of the information that the “click_time” item carries internally. Follows dataset after addition:

	ip	app	device	os	channel	click_time	target	year	month	day	hour	minutes	second	yday	wday	week
1	199085	35	1	13	274	0.003192087	1	2017	11	6	16	0	4	310	2	45
2	126647	72	1	6	101	0.003195933	1	2017	11	6	16	0	5	310	2	45
3	24200	19	88	24	213	0.003203624	1	2017	11	6	16	0	7	310	2	45
4	163643	45	1	41	213	0.003219008	1	2017	11	6	16	0	11	310	2	45
5	129394	29	1	10	213	0.003249775	1	2017	11	6	16	0	19	310	2	45
6	201160	19	0	21	213	0.003253621	1	2017	11	6	16	0	20	310	2	45
7	198008	155	1	13	213	0.003257467	1	2017	11	6	16	0	21	310	2	45
8	119039	35	1	22	274	0.003284388	1	2017	11	6	16	0	28	310	2	45
9	112026	8	1	19	140	0.003288234	1	2017	11	6	16	0	29	310	2	45

Figura 5 - Table after feature engineering

In addition to the added variables, for good practice the name of the predictor variable “is_attributed” was replaced by “target”, as indicated by the red arrow in the figure above.

After the addition of the new variables, the column “click_time” had the scale changed already preparing for the interpretation of the machine learning model, that is, the distribution of the values began to occupy a variation between 0 and 1. An important observation to do is that there was no exchange of information present in the variable, only the scale has been moved to values where the machine learning model can understand and estimate new values.

Using 2 charts for better understanding:

The following is the distribution of the “click_time” data in its original form, represented by figure 6, and the data distribution of this variable after scaling, represented by figure 7:

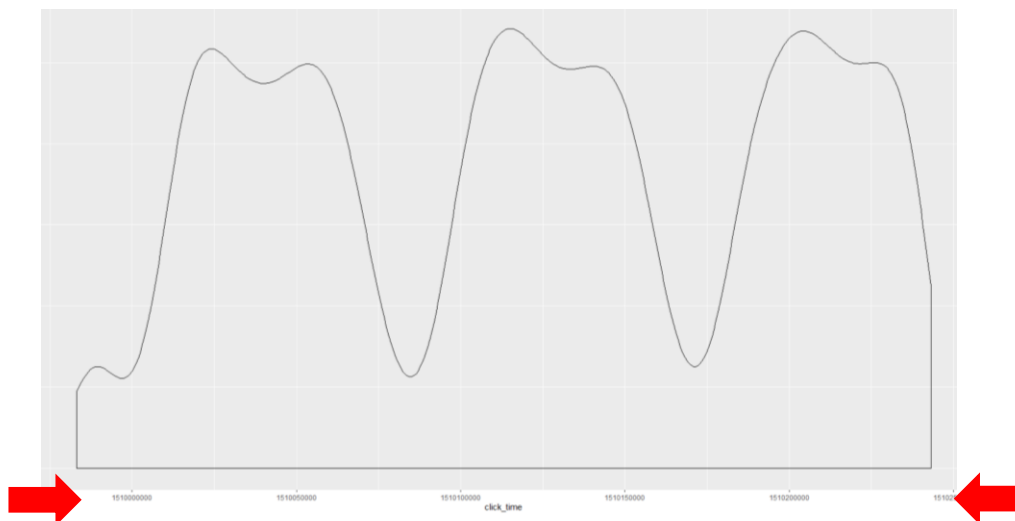


Figura 6 - click_time before rescale

Figures 6 and 7 indicate data distribution, note the variation in scale indicated by the red and yellow arrows respectively. The scale changed but the distribution remained the same.

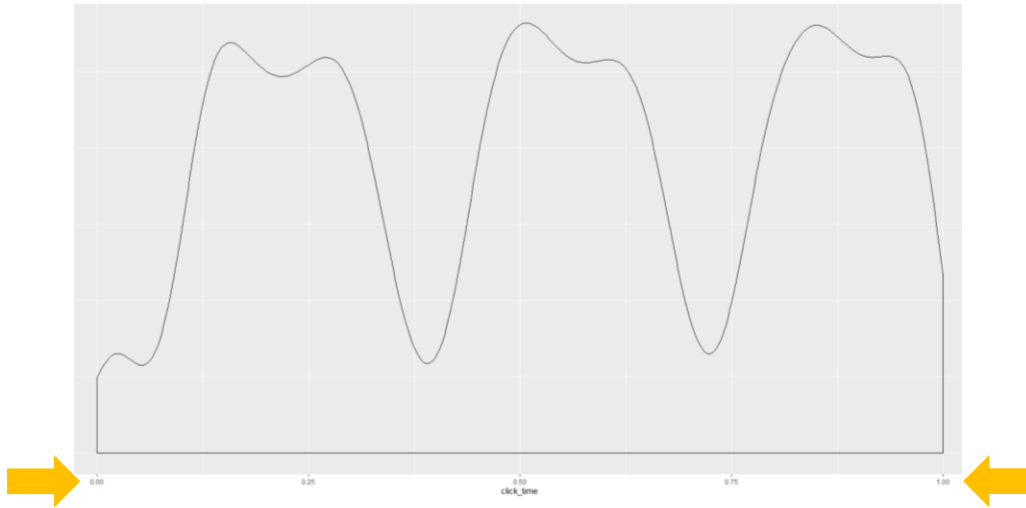


Figura 7 - click_time after rescale

Follows code used in this process:

```
# Checking if we have balanced data between 1 and 0 of target variable so it'll be a fair prediction model
nrow(filter(train.df, is_attributed == 1))/nrow(train.df)*100
nrow(filter(train.df, is_attributed == 0))/nrow(train.df)*100

# Unfortunately not, in fact we have:
# ~ 00.25% in 1 and
# ~ 99.75% in 0
# That is a real problem because I have to create an unbiased model. I'll have to deal with it.

# First let's separate all 1 data and 0 data
onedata <- filter(train.df, is_attributed == 1)
zerdata <- filter(train.df, is_attributed == 0)

# Now, as we have too many data and too less memory, let's get a sample of 50.000 of both files
set.seed(123)
samp <- createDataPartition(y = onedata$n, p=0.1096, list = FALSE)
set.seed(123)
onedata_train <- onedata[samp,]
onedata_train <- onedata_train[1:50000,]
set.seed(123)
onedata_test <- sample_n(onedata[-samp,], 50000)

set.seed(321)
samp <- createDataPartition(y = zerdata$n, p=0.0002711, list = FALSE)
set.seed(321)
zerdata_train <- zerdata[samp,]
zerdata_train <- zerdata_train[1:50000,]
set.seed(321)
zerdata_test <- sample_n(zerdata[-samp,], 50000)

# Binding balanced data
train_sample.df <- rbind(onedata_train, zerdata_train)
test_sample.df <- rbind(onedata_test, zerdata_test)
train_sample.df$n <- NULL
test_sample.df$n <- NULL

# Checking again if we have balanced data between 1 and 0 of target variable so it'll be a fair prediction model
nrow(filter(train_sample.df, is_attributed == 1))/nrow(train_sample.df)*100
nrow(filter(train_sample.df, is_attributed == 0))/nrow(train_sample.df)*100

nrow(filter(test_sample.df, is_attributed == 1))/nrow(test_sample.df)*100
nrow(filter(test_sample.df, is_attributed == 0))/nrow(test_sample.df)*100

# Now we have 100.000 data:
# - 50.00% in 1 and
# - 50.00% in 0
```

Figura 8 - Feature Engineering Code Part I

```

# First of all I will unite both datas to one data and deal with all Feature Engineering
# Control Variable
train_sample.df$control <- 1
test_sample.df$control <- 0

# Binding
datatemp <- rbind(train_sample.df, test_sample.df)

# Just for convenience I will rename the predictor variable 'is_attributed' to 'target'
datatemp$target <- datatemp$is_attributed
datatemp$is_attributed <- NULL

# CHECKING FOR MISSING VALUES: NO!
any(is.na(datatemp))

# CHECKING VARIABLE TYPES: ALL ARE int TYPE EXCEPT click_time THAT IS char
glimpse(datatemp)

# As we can see, all variables are 'int' type, but 'click_time' that is char
# click_time should be in Date-Time format, let's make some changes:

# Transforming to dttm
datatemp$click_time <- ymd_hms(datatemp$click_time)

# Creating New Variables just by separating the date and time of 'click_time'
# Year
datatemp$year <- year(datatemp$click_time)

# Month
datatemp$month <- month(datatemp$click_time)
#datatemp$Month <- month(datatemp$click_time, label = TRUE) # if month labeled wanted

# Day
datatemp$day <- day(datatemp$click_time)

# Hour
datatemp$hour <- hour(datatemp$click_time)

# Minutes
datatemp$minutes <- minute(datatemp$click_time)

# Second
datatemp$second <- second(datatemp$click_time)

# yday
datatemp$yday <- yday(datatemp$click_time)

# mday
#datatemp$mday <- mday(datatemp$click_time)

# wday
datatemp$wday <- wday(datatemp$click_time)
#datatemp$wday <- wday(datatemp$click_time, label = TRUE) # if wday labeled wanted

# week
datatemp$week <- week(datatemp$click_time)

# Internal integer representation of click_time
datatemp$click_time <- unclass(datatemp$click_time)

# Separating train and test again
train_sample.df <- filter(datatemp, control == 1)
train_sample.df$control <- NULL
test_sample.df <- filter(datatemp, control == 0)
test_sample.df$control <- NULL

# Rescaling click_time from 0 to 1 to get better results
# Note: I'm not changing the values, just the scale
ggplot(data = train_sample.df, aes(click_time)) +
  geom_density(kernel = 'gaussian')
#hist(train_sample.df$click_time)
train_sample.df$click_time <- rescale(train_sample.df$click_time)
test_sample.df$click_time <- rescale(test_sample.df$click_time)

# Plotting Rescaled click_time histogram
ggplot(data = train_sample.df, aes(click_time)) +
  geom_density(kernel = 'gaussian')
ggplot(data = test_sample.df, aes(click_time)) +
  geom_density(kernel = 'gaussian')

# Cleaning Memory
rm(datatemp)

# CSV to Power BI
#write_csv(train_sample.df, "20A-PowerBiData.csv")

```

Figura 9 - Feature Engineering Code Part II

With feature engineering finished, we can move to the beginning of the exploratory analysis process.

7 – Exploratory Analysis I

Exploratory analysis is critical to understanding where we have the best variables, where we have problems, where we have business opportunities, and in this way, we can generate a number of important conclusions to continue the machine learning process.

For this project, PowerBi was chosen as a graphical viewer because it offers a platform with varied features assisting in the company's Business Intelligence.

PowerBi brings a professional way to create high-level interactive dashboards assisting data analysis enabling you to monitor any business need in real time.

After handling variables in the previous chapter, the following dashboard was created:

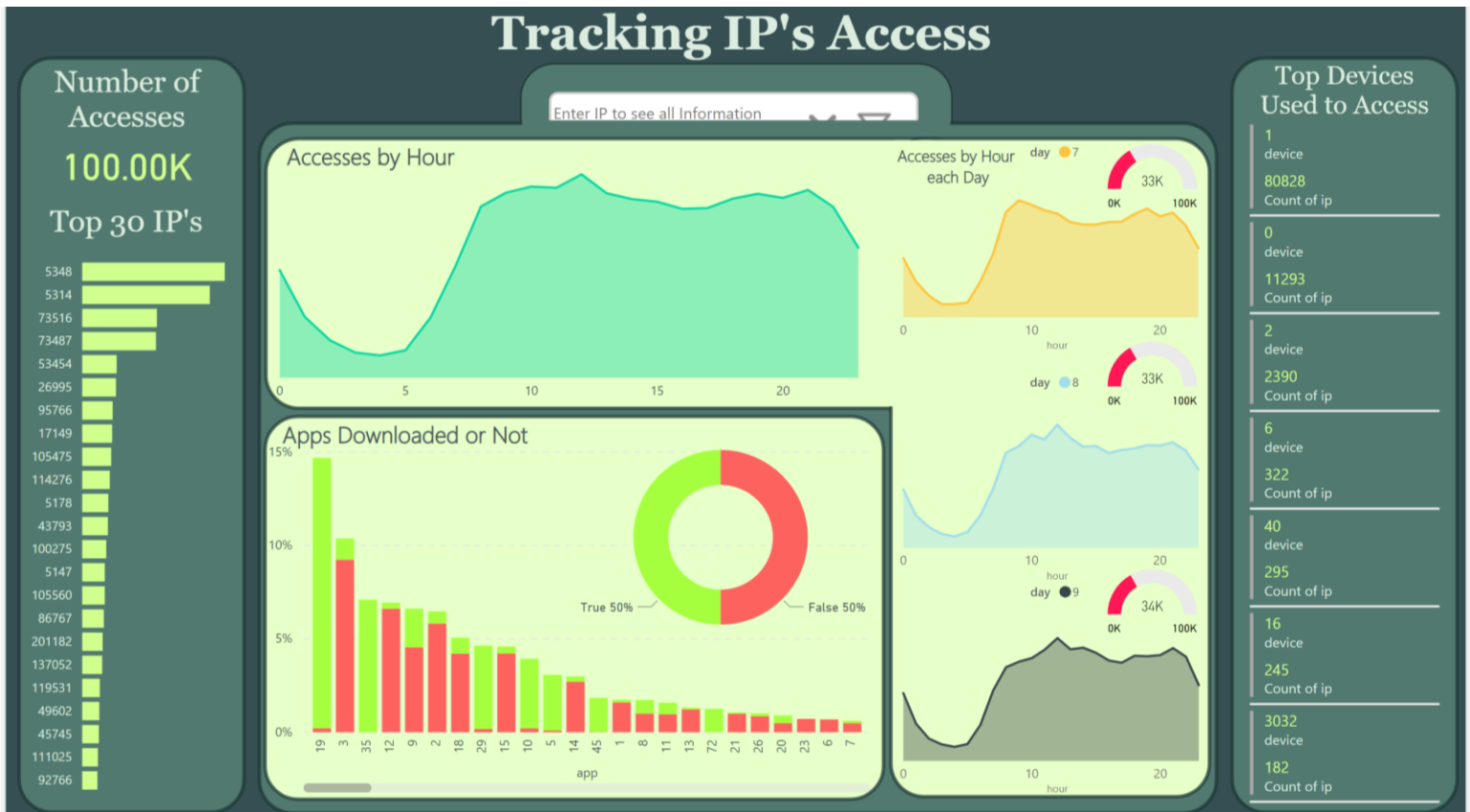


Figura 10 - Dashboard IP Accesses

As noted, we have a lot of information inside a dashboard.

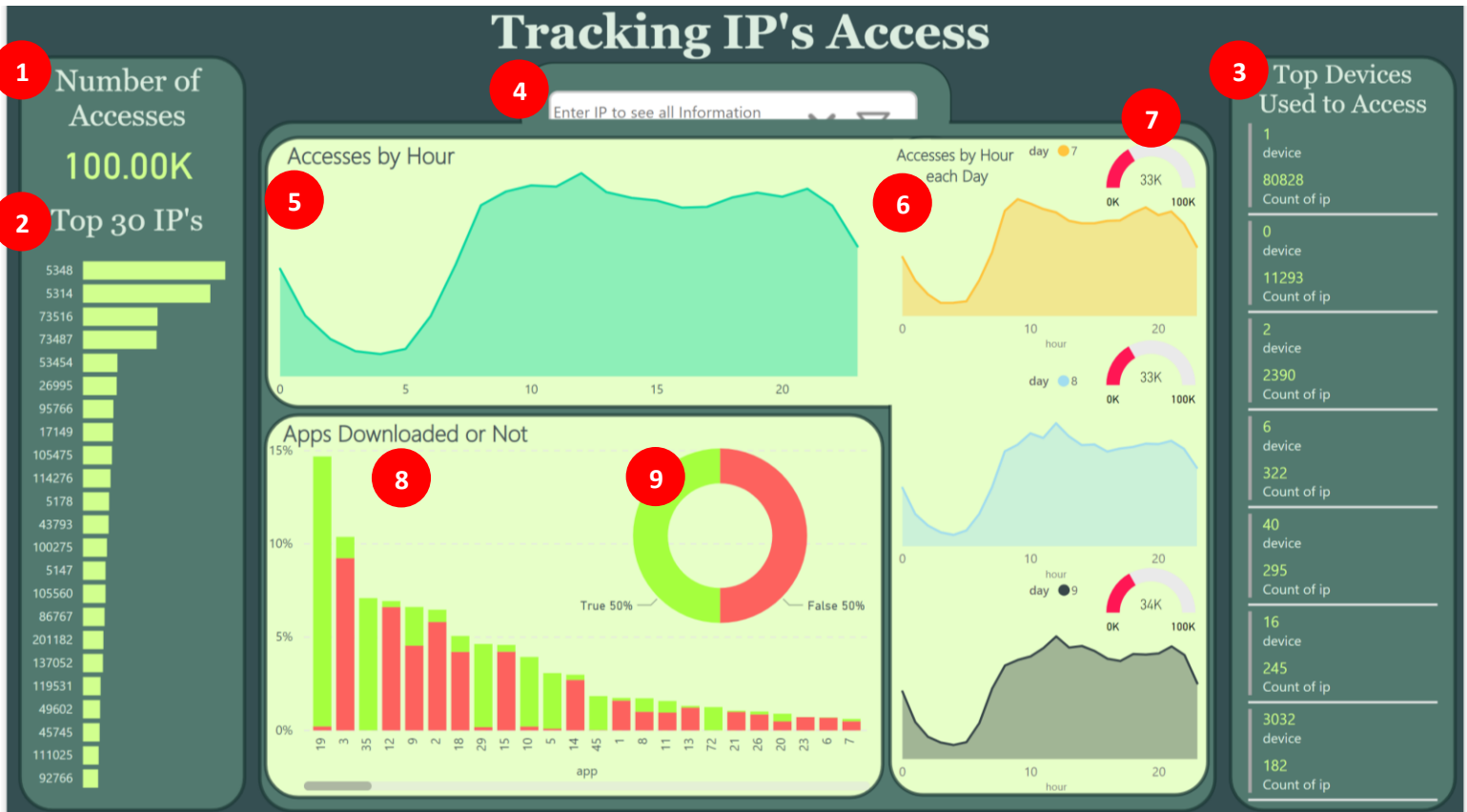


Figura 11 – Dashboard Analysis

1. Number of Accesses

The number of accesses corresponds to the total value of interactions that a given item performed within the analyzed period. For this project it was limited to 100,000 accesses between days 7, 8 and 9. The items that can be analyzed are: IP's, Accesses by Hour, Accesses by Hour each Day, Devices used for access and all applications downloaded or not in this period.

2. Top 30 IP's

This item shows the 30 most accessed IP's in this period. It also works as a filter allowing to analyze by IP all other information (Accesses by Hour, Accesses by Hour each Day, Devices used for access and all applications downloaded or not during this period). During Business Intelligence analysis in the next chapters this item will indicate important information for decision makers.

3. Top Used Devices

TalkingData-encoded item that identifies the most commonly used devices for access within the specified time period.

4. IP Filtering

Filter 1 or more IP's at the same time for access analysis and tracking.

5. Accesses by Hour

Indicative graph of the number of accesses per hour considering the whole period (days 7, 8 and 9).

6. Accesses by Hour each Day

Graph indicating the number of accesses per hour per day.

7. Accesses Indicator

Number of accesses indicator each day of the week.

8. Apps Downloaded or not

This item displays all applications that have linked advertisements and clicks through the IPs. The graph shows the number of clicks per application and also if after the click the application was downloaded (green color) or not (red color). Note that here apps are also coded by TalkingData.

9. Percentage of Downloads

This item supplements the information in the previous item by showing in% the number of applications downloaded (True) after click, or not downloaded (False). Note that the design study is considering a uniform distribution between True and False, with 50% for each side as indicated at the beginning of the feature engineering process.

With all the items explained we can move on to data analysis.

8 – Exploratory Analysis II

In this chapter we will present a series of conclusions by analyzing the dashboard presented earlier.

1. Most Accessed Time

One business opportunity is to identify the most accessed time to pass on information to the marketing team so that projects are geared toward that time to maximize application sales.

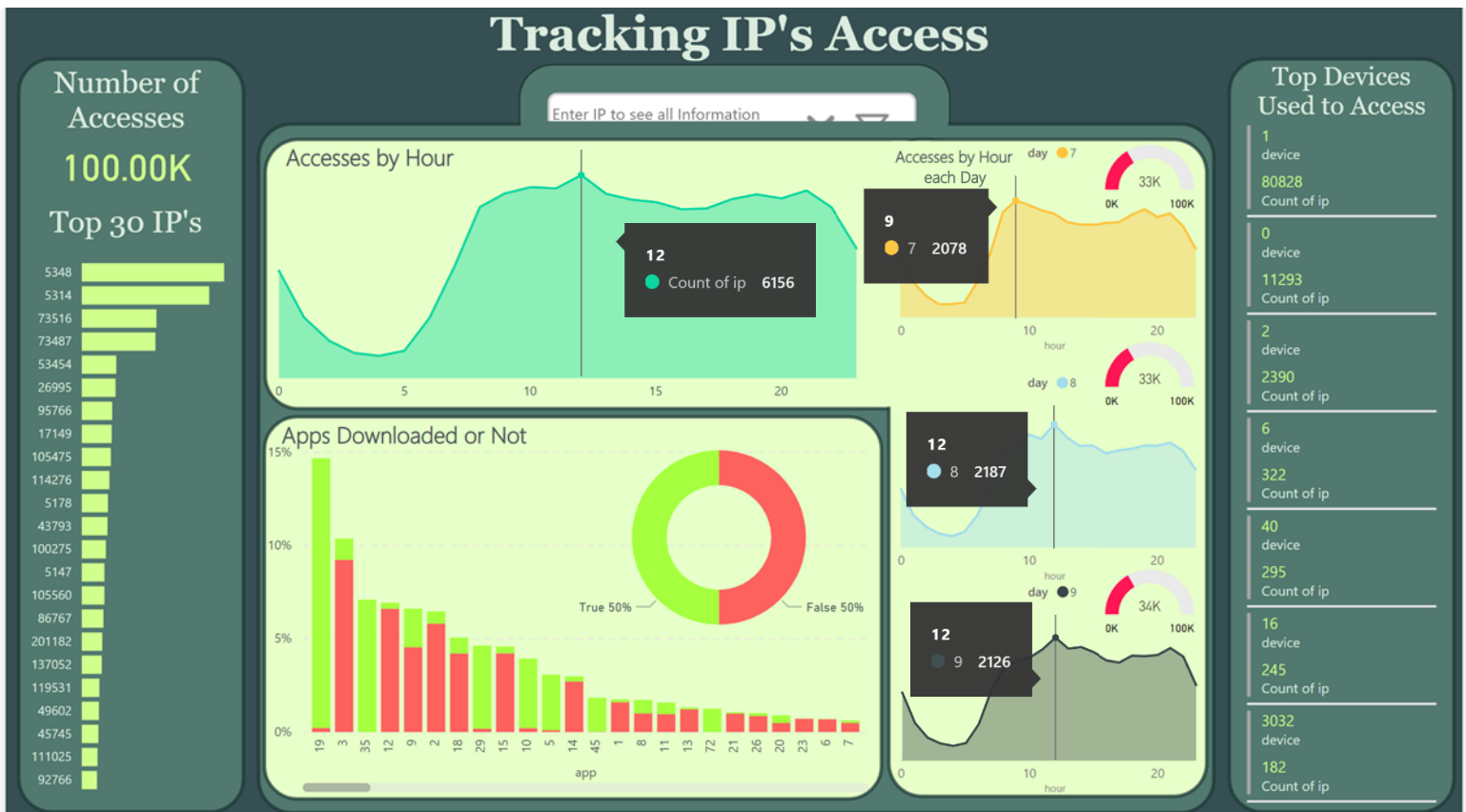


Figura 12 – Most Accessed Time

According to dashboard, the busiest time is at 12:00 PM in the China time zone. Valuable information for the marketing team, as targeting the most relevant advertisements for this period can generate more revenue for companies.

It is worth noting that this is the maximum value considering the overall sum of the 3 days (7, 8 and 9), but for a more assertive focus we can observe the peak times each day, as shown in figure 11, further specifying the correct moments to come in with advertising and marketing. Day 7, for example, shows the best time at 09:00 AM not at 12:00 PM.

2. Most Accessed Applications

Looking at the apps with the most access doesn't mean we have higher download conversion, because not necessarily a click has been converted to a download, but advertisers pay per executed click.

In this regard, care must be taken not to jump to conclusions without looking deeper into what the data is showing us.

→ App 19

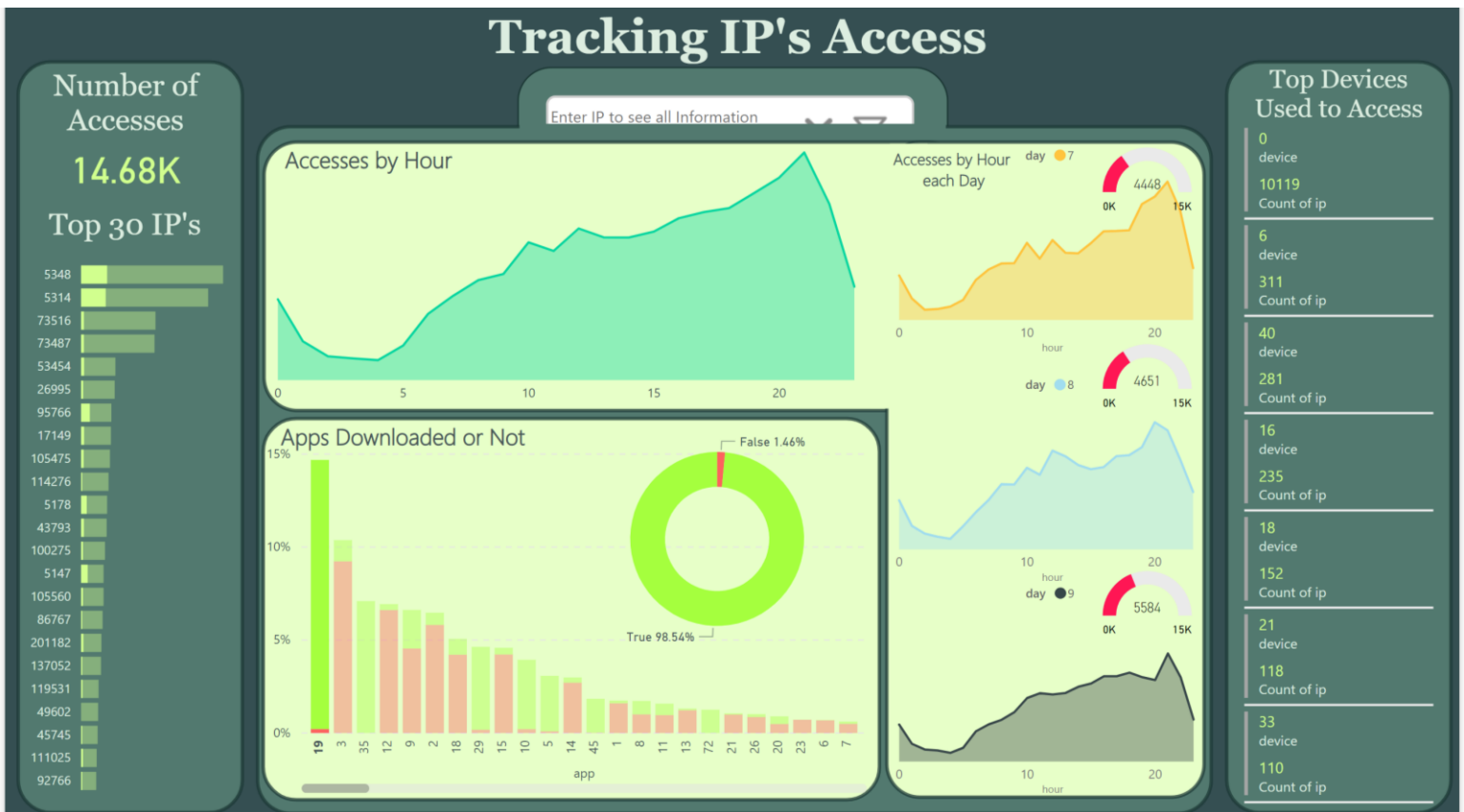


Figura 13 – Most Accessed Apps - App19

Firstly, with the most accesses we have app 19. In this case, we have an excellent download conversion:

- Accesses: 14.680
- Clicks with Downloads: 98.54%
- Clicks without Downloads: 1.46%

The time of greatest access is at 21:00 PM but the window of opportunity starts at 15:00 PM and on day 9 we had the most clicks.

→ App 3

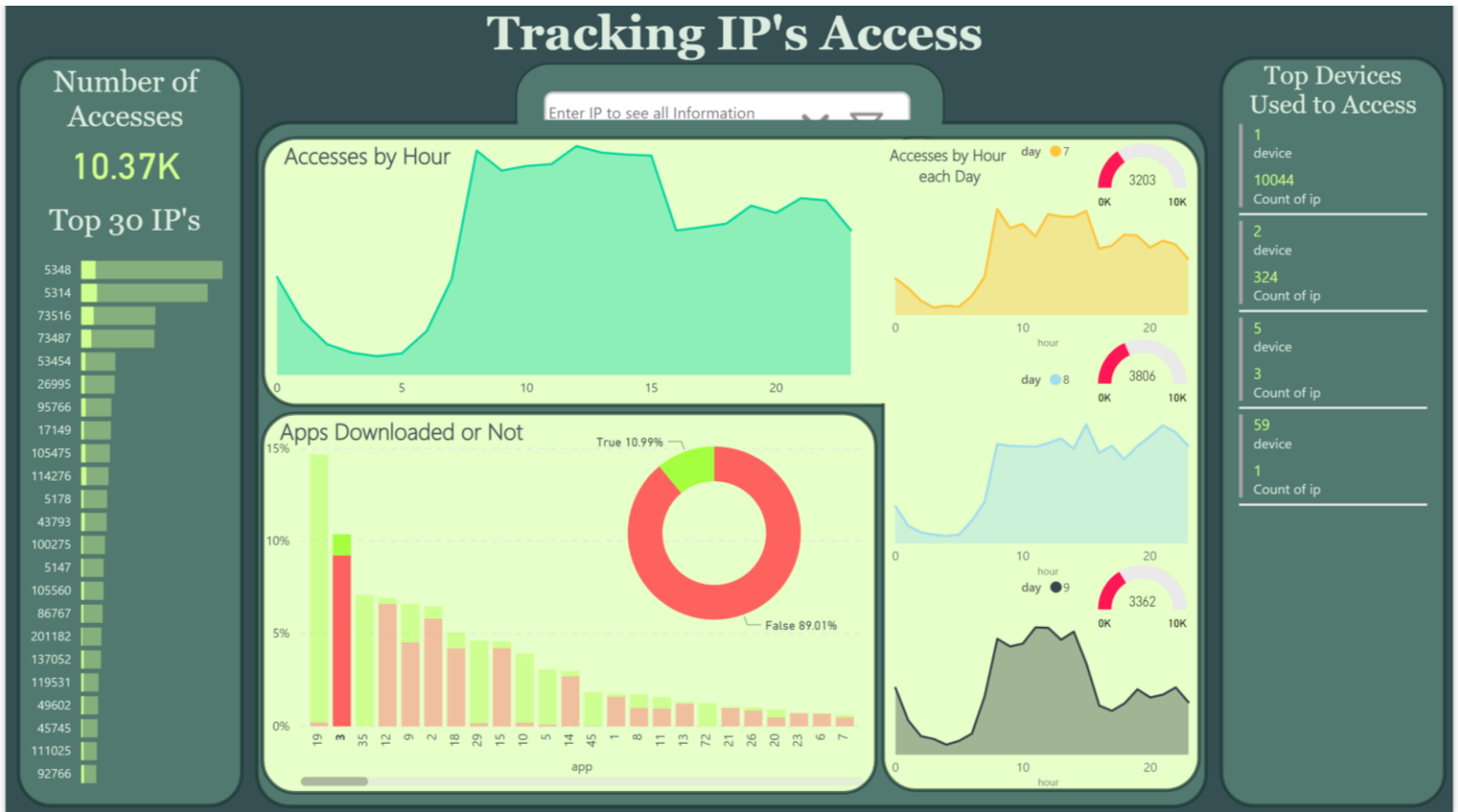


Figura 14 - Most Accessed Apps - App03

Second with the most accesses is app 3. In this case, we can see that most clicks are not converted to downloads:

- Accesses: 10.370
- Clicks with Downloads: 10.99%
- Clicks without Downloads: 89.01%

The highest access time occurs between 06:00 AM and 15:00 PM from devices with ID 1.

Now let's look just at the False portion (89.01%) of App 3, possible fraudulent clicks:

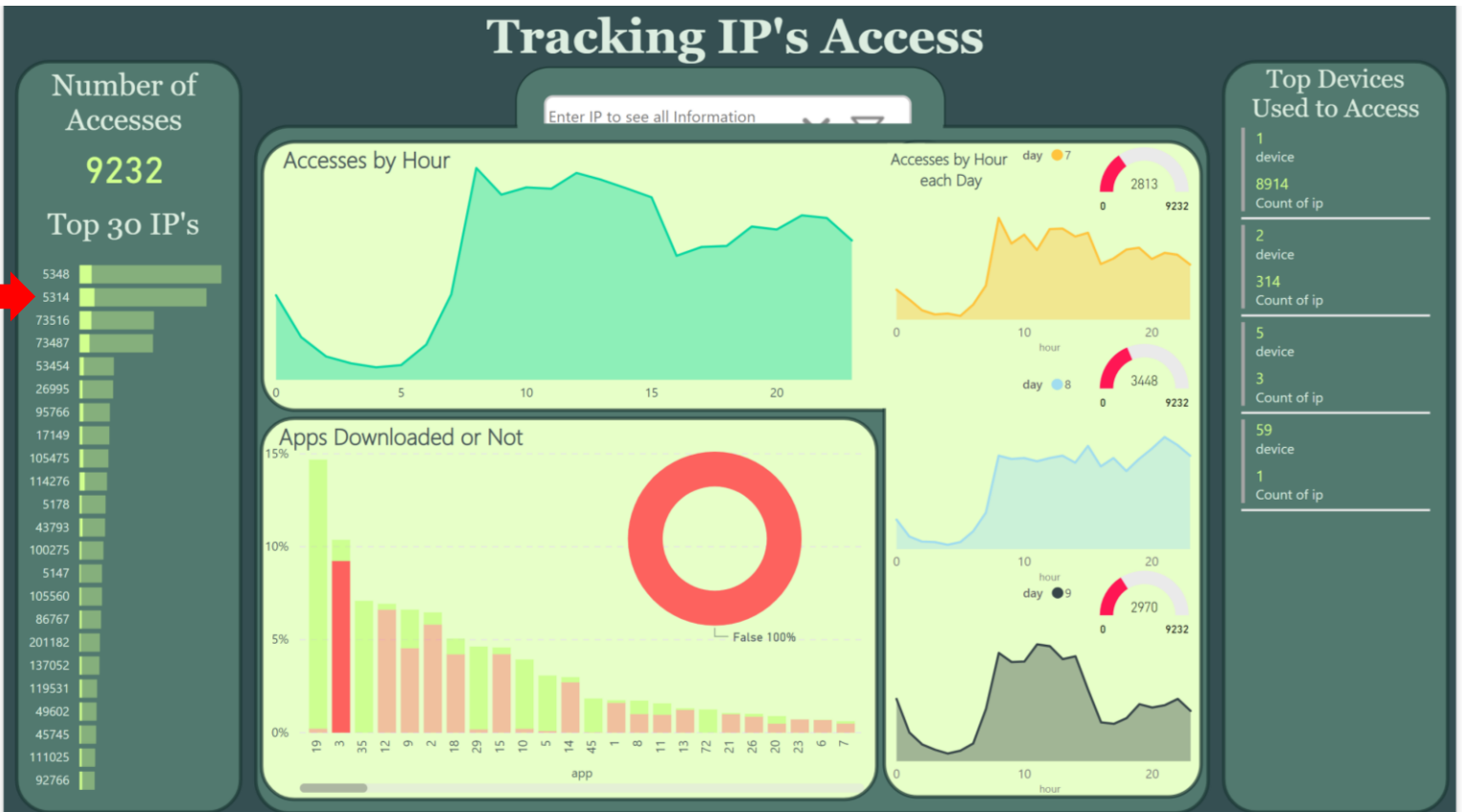


Figura 15 - False Clicks App03

As shown in Figure 14, IP “5314” is signaling a lot of clicks but few downloads for this app, but again it is not enough to punctually analyze just one piece of information to determine if this IP is a fraudulent user, it is best to analyze the click journey around all the portfolio. We will then perform this analysis on the next item, identifying how some IP's (including 5314) perform clicks with or without download.

3. Click Journey by IP

In this item we will explore how the following users behave within the analyzed period:

- 5.348;
- 5.314;
- 53.454;
- 114.276.

→ IP 5.314

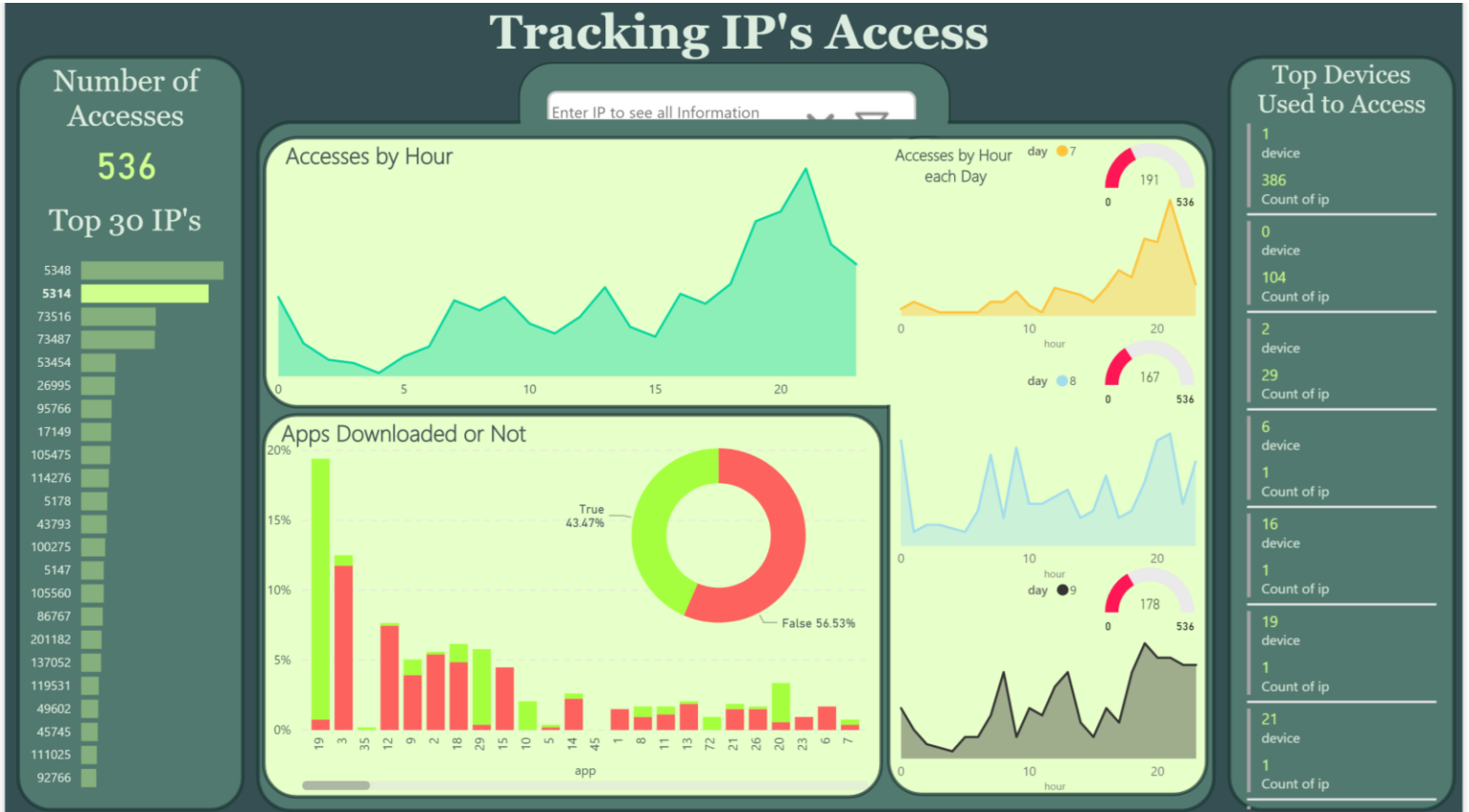


Figura 17 - Click Journey - IP 5.314

- Accesses: 536
- Clicks with Downloads: 43.47%
- Clicks without Downloads: 56.53%
- Highest Accesses Range: 17:00PM – 23:00PM
- Accessed Apps with more Downloads: 19, 29, 10
- Accessed Apps with fewer Downloads: 3, 12, 2
- Day with most accesses: 7
- Most used devices: 1 e 0

Without much variation, when compared to the first IP analyzed, it can be observed that less downloaded applications also follow the same pattern for this IP. This may indicate that these apps really don't have much of a download conversion rate, unlike what we thought would be fraudulent clicks. The analysis of the next IP's will confirm or refute this theory.

→ IP 53.454

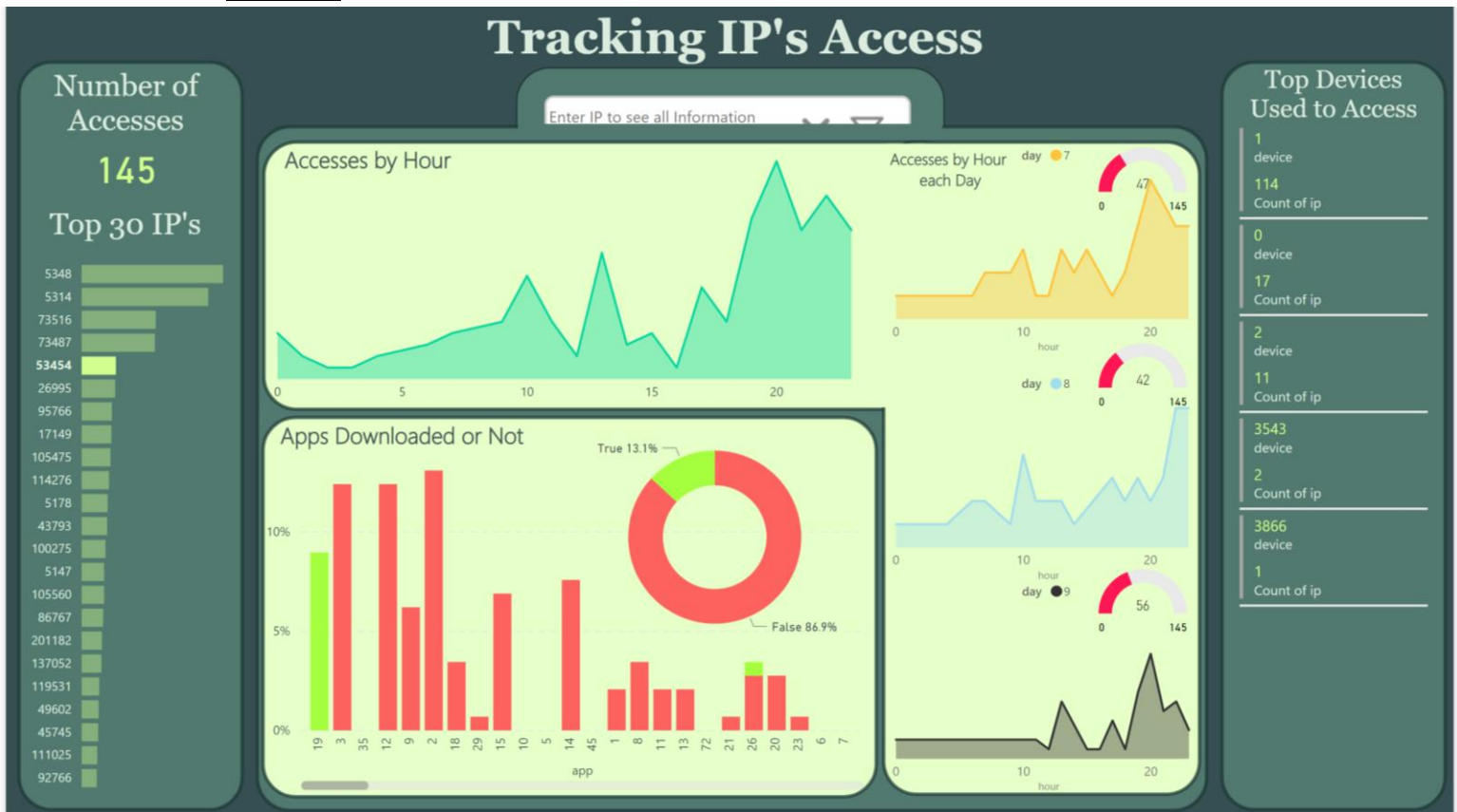


Figura 18 - Click Journey - IP 53.454

- Accesses: 145
- Clicks with Downloads: 13.10%
- Clicks without Downloads: 86.90%
- Highest Accesses Range: 17:00PM – 23:00PM
- Accessed Apps with more Downloads: 19
- Accessed Apps with fewer Downloads: 3, 12, 2
- Day with most accesses: 9
- Most used devices: 1 e 0

Here we have an IP that is more likely to be fraudulent as the history of clicks without downloads is very low. A recommendation to be assertive would be:

1. Initially, track more intensely this IP by extracting more clicks;
2. Observe the apps with the least downloads and the percentage of clicks on each;
3. Thirdly, look if access times have default entries because in this observation the peak hours are in the same pattern as the IPs previously analyzed, leaving some uncertainty about fraud, so acquiring more information can confirm this uncertainty.

→ IP 114.276

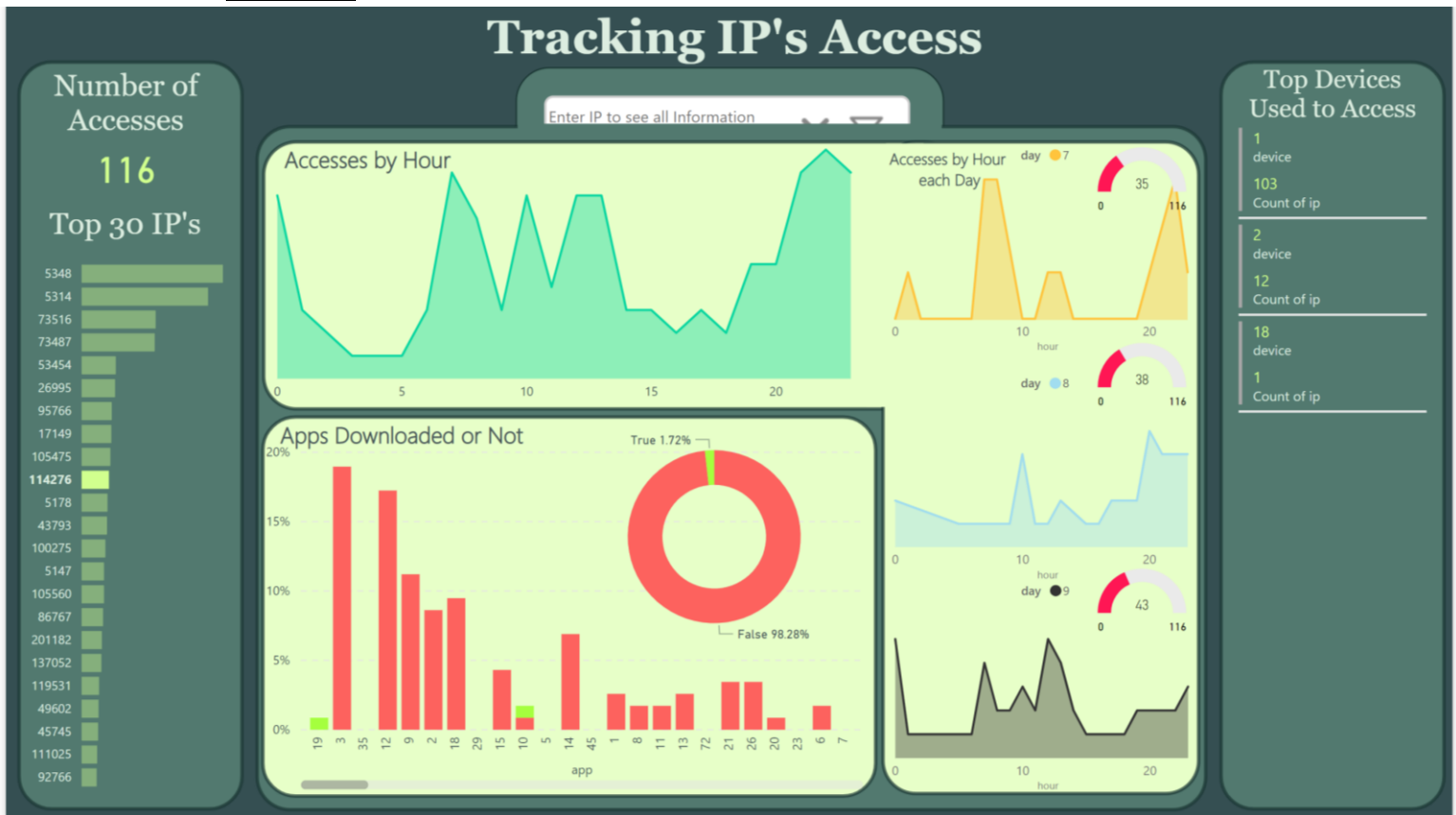


Figura 19 - Click Journey - IP 114.276

- Accesses: 116
- Clicks with Downloads: 1.72%
- Clicks without Downloads: 98.28%
- Highest Accesses Range: Aleatório
- Accessed Apps with more Downloads: 19, 10
- Accessed Apps with fewer Downloads: 3, 12, 9
- Day with most accesses: 9
- Most used devices: 1 e 2

In this last example, we have 99% chance of being an IP responsible for fraudulent clicks. First of all, 98.28% of clicks made are not converted to downloads running away from the overall average; second there is no access pattern, note in the "Accesses by Hour" chart that access peaks occur at 00:00, 07:00, 10:00, 12:00, 21:00; third, note in the "Accesses By Hour each Day" that there is no constancy; Finally, considering the whole, this IP shows that app 19 (with the most downloads in all previous reviews) has low number of clicks with downloads.

This is only part of the data analysis, an in-depth study can provide many other opportunities for improvement and identification of business opportunities.

9 – Machine Learning Model Building

In this chapter, we will build three different machine learning models in order to analyze which one we get the most accuracy from and then present new and unknown data and then evaluate if we have a reliable model.

Machine Learning is an area of artificial intelligence that studies learning techniques by applying programming and computing to build mathematical models with the ability to get knowledge automatically based on the data provided.

Through this data, the trained algorithm is able to make decisions when new data is presented to it, thus providing complex problem solving solutions that in many cases would be difficult for a person to perform.

Within this concept we have 2 types of learning:

- **Supervised Learning**
In this type of learning, we have a set of input data and possible output data that should be used to train the model. In other words, by testing this model we can compare the predicted output data with that provided and evaluate the accuracy of the trained model. If unsatisfactory, it is possible to go back to the beginning of model creation and improve the predictor variables to perform new tests.
- **Unsupervised Learning**
In this other type of learning, we have an input data set but the outputs are unknown, so I can't compare the predicted data with the expected outputs. For this learning modality other techniques are applied, for example, the data will be grouped and the results will change according to the variables.

Machine Learning can be used to solve a range of problems from internet search suggestions, spam message tracking, to digital marketing usage.

For our business problem, searching for fraudulent clicks, we will use supervised learning, but before we go into creating the predictive model, let's understand better how I used a technique called cross-validation to get better performance from the machine learning model.

1. Cross – Validation

During the Machine Learning process several iterations (repetitions) happen so that models can deliver better reliability. In this process many choices must be made, and each will generate a reflection, positive or negative, at the final result. To choose between more relevant arguments; machine learning model suited to the business problem; which variables to deliver to the predictive model; among others, can be an endless challenge.

Fortunately, we have several techniques for evaluating and validating the predictive model based on the chosen variables, one of them being cross - validation.

Imagine a scenario where we have 150,000 rows of data to analyze. Usually a sample of approximately 20%, corresponding to 30,000 random lines, would be set aside to perform validation at the end of the machine learning process and the remainder (80%) would be used to train the model. Because it is 20% of randomly collected data there is no guarantee that this data will generate the best validation performance.

Maybe you should imagine that taking a larger amount of sample, somewhere between 50% and 70%, could lessen the uncertainty and possible correct validation errors? Yes, but doing so will decrease the amount of data available to train the model and can lead to failures in the prediction process.

In Cross - Validation the process of separating validation and training data happens dynamically, following image for better understanding:

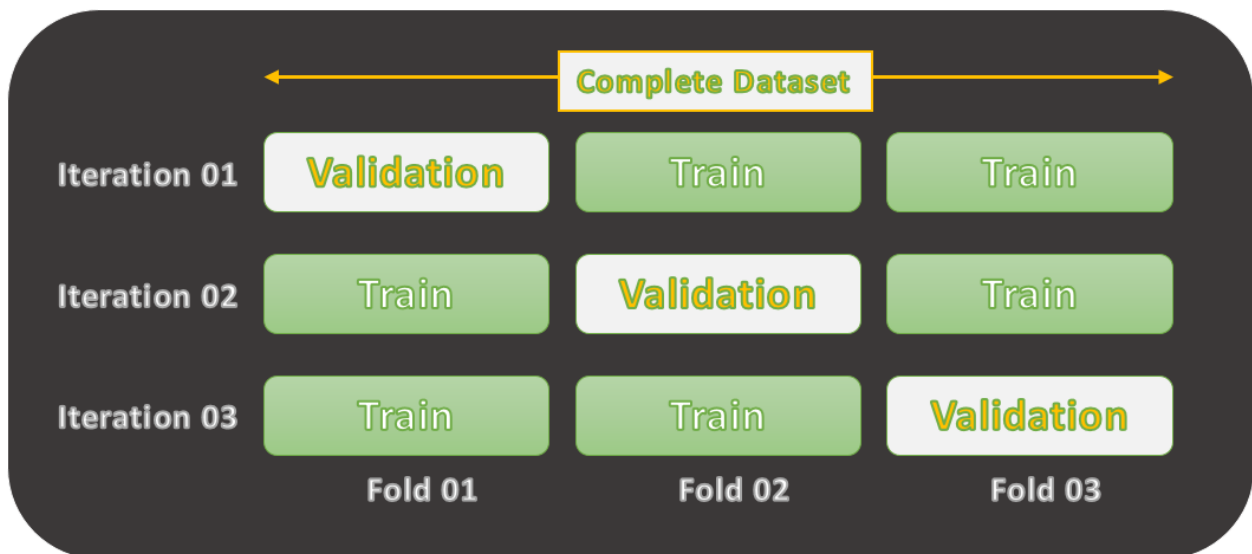


Figura 20 - Cross - Validation

As noted in Figure 20, the dataset is divided into training data and validation data called folds.

How this process works:

Initially each iteration will have a percentage of training data predefined by the folds, in which case we would have 2/3 of the data intended for model training and the remaining 1/3 for validation, since it was set fold = 3.

This procedure is repeated for the amount of folds determined by the data scientist until 100% of the data has been validated in consecutive iterations, providing us the model accuracy considering all dataset rows and not just those 20% selected only once.

This way I do not need to separate just one (uncertain) sample of data as validation will occur with multiple samples passing through the entire dataset.

Finding the optimal value for folds can be a challenge when designing Machine Learning algorithms as high numbers can yield better results but also slow and lose performance due to the requested processing cost.

The ideal scenario will depend on each business problem, for this project I used folds ranging from 1 to 5 and the most appropriate number found between performance and processing was fold = 3.

```
# Using Cross Validation I'll run all algorithms with 3 fold
trctrl <- trainControl(method="cv", number=3)
# Metric to get measures of erros
evaluation <- "Accuracy"
```

Figura 21 – Parameterization of Cross - Validation and Accuracy

2. Evaluation Metric

To validate the model I used “Accuracy” as a metric. Accuracy can be represented by the following formula:

$$Accuracy = \frac{\text{number of correct predictions}}{\text{total of instances}} \times 100$$

This formula divides the correct predictions by the total number of instances in the dataset generating a number as a percentage of prediction accuracy.

There are other evaluation metrics, such as Root Mean Square Error (RMSE) where the mean square error is evaluated, but, as this is a classification problem, where target values assume a fixed number (0 or 1), the accuracy will determine accurately the correct amount between errors and success prediction.

3. Machine Learning Algorithms

As a solution proposal I will perform the evaluation using three different algorithms, CART, KNN and Random Forest, explained below.

3.1 CART¹¹

Cart, Classification and Regression Trees, is a methodology that allows you to perform both classification and regression trees, that is, to classify a categorical variable indicating which class would fit most accurately or to predict the value of a continuous variable within the regression.

Each problem will ask for a specific type of application, for this project the classification will be used.

The CART algorithm works with questions and answers to make decisions. With each question answered a different path is followed within the tree until the final node is reached and the final answer is completed.

Um diferencial do CART está no fato de que o algoritmo é capaz de utilizar as mesmas variáveis mais de uma vez em diferentes partes da árvore eliminando possíveis dependências complexas existentes entre elas. Com isso consigo realizar também o cross – validation com mais qualidade.

```
# CART
set.seed(9)
model_cart <- train(target~., data=train_sample.df,
                    method="rpart", metric=evaluation,
                    trControl=trctrl)
plot(model_cart)
```

Figura 22 - CART train

Let's see how the model behaved in this training run.

¹¹ <https://www.rdocumentation.org/packages/MachineLearning/versions/0.1.3/topics/CART>

CART

100000 samples
15 predictor
2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (3 fold)
Summary of sample sizes: 66666, 66667, 66667
Resampling results across tuning parameters:

cp	Accuracy	Kappa
0.05826	0.8444102	0.6888204
0.06184	0.8142599	0.6285207
0.58894	0.6953690	0.3907488

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was $cp = 0.05826$.

Figura 23 – CART Modeling Status

Figure 23 summarizes how the machine learning model developed during training:

- 100,000 samples were used;
- We have in the model 15 predictor variables and 2 classes to predict;
- We use the Cross - Validation method with fold = 3 with each iteration having approximately 66,666 lines for training;
- And 3 accuracy values were analyzed according to the cp complexity coefficient, and the best value was $cp = 0.05826$ generating 84.44% of accuracy.

Accuracy chart follows:

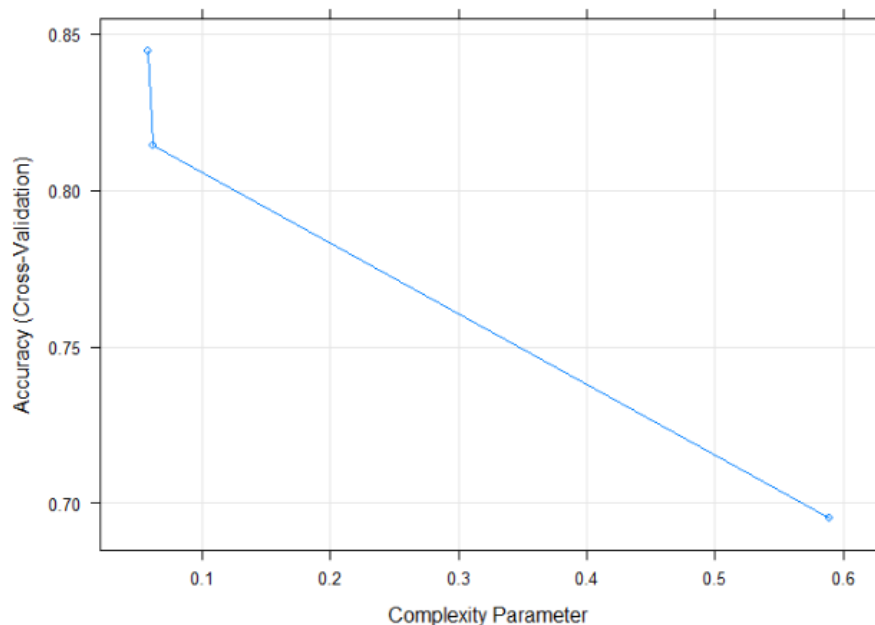


Figura 24 - CART Model Accuracy Graph

3.2 KNN¹²

K-nearest neighbors is an algorithm that can also be used for classification or regression and works by finding the nearest neighbors of the predicted value where K is satisfied:

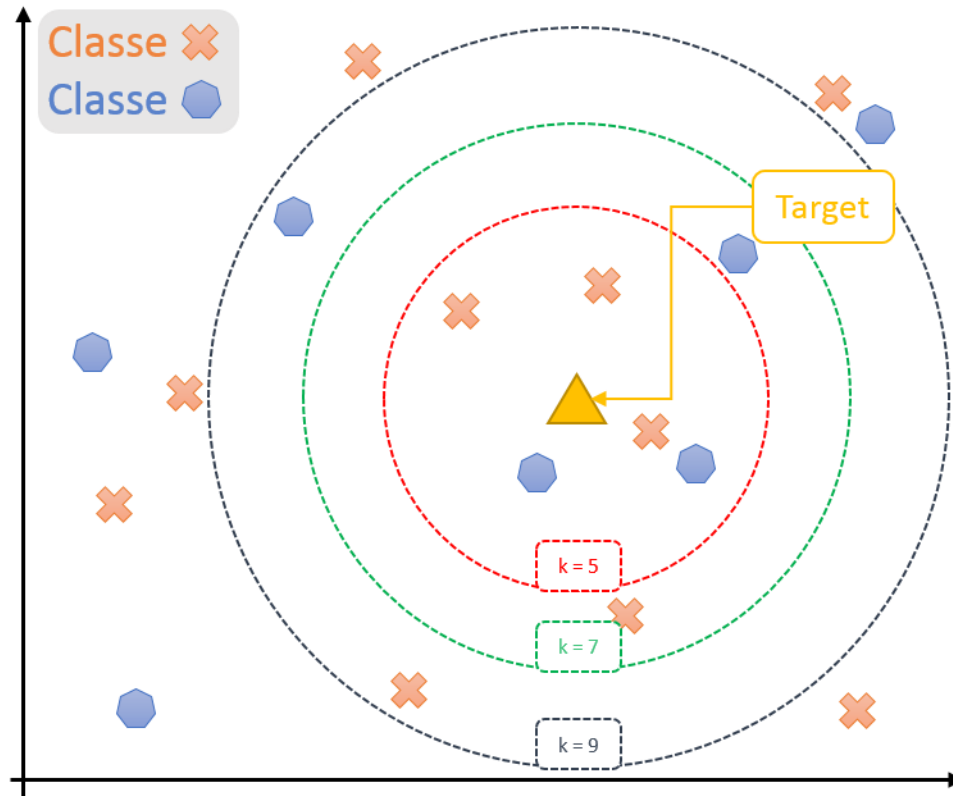


Figura 25 - kNN and relation k

Note that for each circle an amount of data is wrapped depending on the specified k value. When $k = 5$ the algorithm finds the 5 closest data to target and votes for the highest amount, which in this case would be the class 'X'.

The key to success for this algorithm lies in the correct choice of parameter k keeping in mind that a high value brings more error to the training because the increase of the area encompasses more classes, while the reflection of the error in the validation data tends to decrease to a certain limit and then increases considerably.

Segue figura 25 exemplificando este processo:

¹² <https://www.rdocumentation.org/packages/DMwR/versions/0.4.1/topics/kNN>

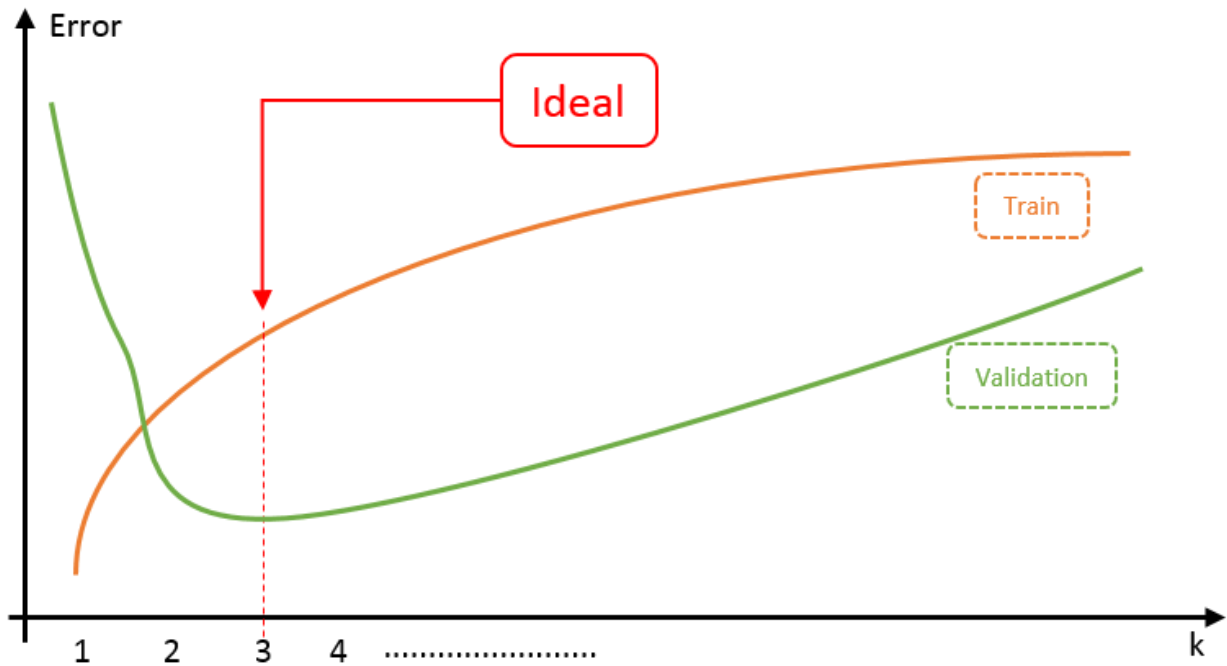


Figura 26 – Choosing Factor k

KNN can be trained as follows:

```
# kNN
set.seed(9)
model_knn <- train(target~., data=train_sample.df,
                    method="knn", metric=evaluation,
                    trControl=trctrl)

plot(model_knn)
```

Figura 27 - kNN train

Let's see how the model behaved in this training run.

k-Nearest Neighbors

```
100000 samples
  15 predictor
  2 classes: '0', '1'
```

```
No pre-processing
Resampling: Cross-Validated (3 fold)
Summary of sample sizes: 66666, 66667, 66667
Resampling results across tuning parameters:
```

k	Accuracy	Kappa
5	0.73539	0.47078
7	0.73414	0.46828
9	0.73199	0.46398

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was $k = 5$.

Figura 28 - kNN Modeling Status

Figure 28 summarizes how the machine learning model developed during training:

- 100,000 samples were used;
- We have in the model 15 predictor variables and 2 classes to predict;
- We use the Cross - Validation method with fold = 3 with each iteration having approximately 66,666 lines for training;
- And 3 accuracy values were analyzed according to the k coefficient of 5, 7 and 9 with the best value being $k = 5$ giving 73.54% of accuracy.

Accuracy chart follows:

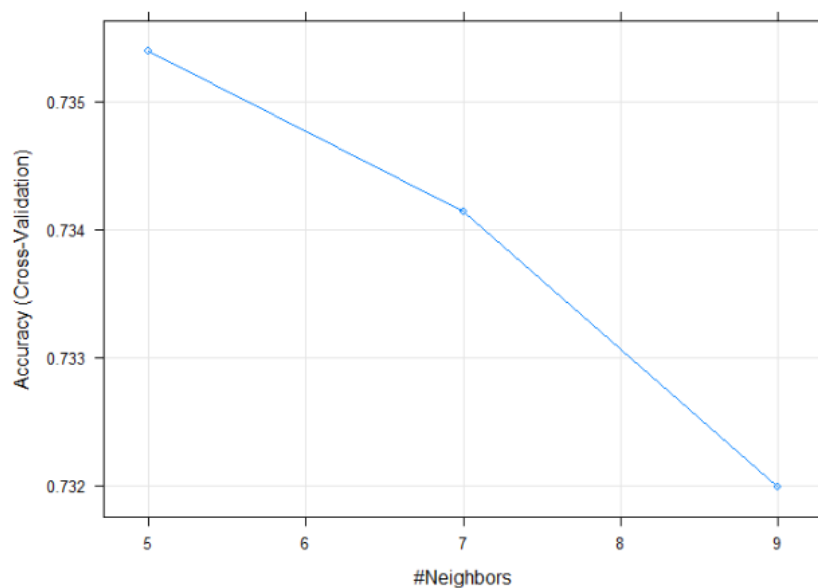


Figura 29 - Accuracy Graph of the kNN Model

3.3 Random Forest¹³

As the name suggests, randomForest means Random Forest, which in Data Science we can make analogy to Decision Trees where each tree has a depth and decides between its 'leaves' which is the best path to travel.

Imagine an inverted tree:



Figura 30 – Inverted tree

End nodes (or leaves) are at the bottom of the decision tree. This means that the decision trees are drawn upside down. Thus, the leaves are the bottom and the roots are the tops (figure above).

A Decision Tree works with both categorical and continuous variables and works by dividing the population (or sample) into subpopulations (two or more sets) based on the most significant divisors of the input variables. For this and many other reasons, decision trees are used in classification and regression problems where the supervised learning algorithm has a predefined target variable.

In randomForest, or Random Forest, we grow multiple trees instead of a single tree. But how does the classification process work? Initially for classifying a new attribute-based object, a tree generates a classification for that object (which is as if the tree gives votes for this class). This process goes on for each tree in the forest and finally, the forest chooses the classification with the most votes (from all trees in the forest).

As shown in figure 31, we can have n trees and each tree can have as many leaves as it wants. This is where we have a problem, because a shallow tree that has been trained to classify an object may not be accurate because it has learned little (higher error), or in other words

¹³ <https://www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/randomForest>

underfitting. At the other extreme we have overfitting, that is, if no limit is set the model will give 100% accuracy in the training set because it ends up making a leaf for each observation but will fail in validation.

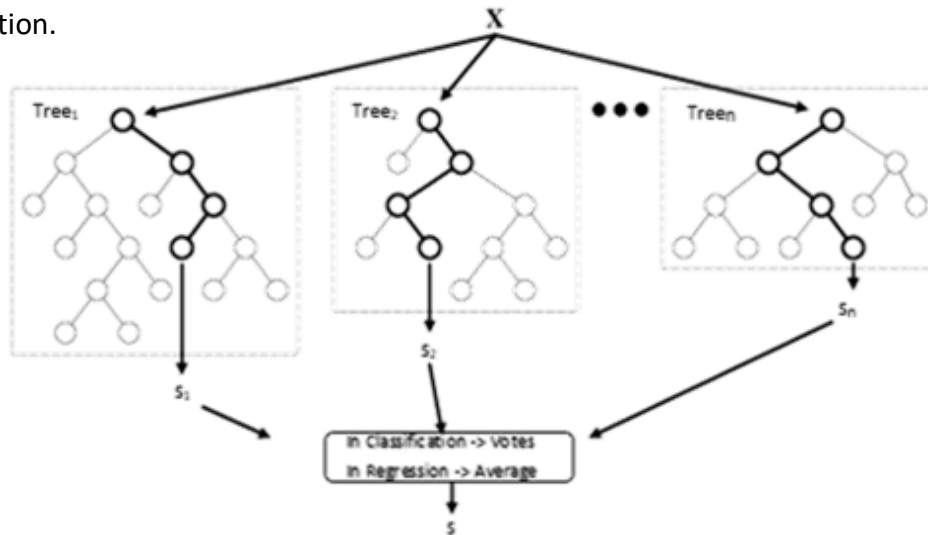


Figura 31 – randomForest

The ideal point is where the error in the test data (validation) is as little as possible (target in figure below), giving the model better accuracy.

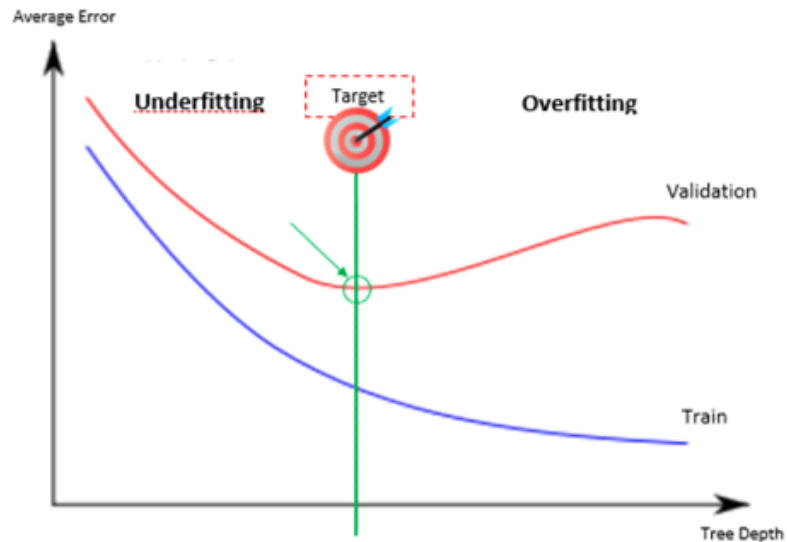


Figura 32 - Underfitting x Overfitting

In this project we will not study underfitting or overfitting because we are studying cross-validation, so we go straight to the construction of the machine learning model:

```
# Random Forest
set.seed(9)
model_rf <- train(target~., data=train_sample.df,
  method="rf", metric=evaluation,
  trControl=trctrl)
plot(model_rf)
```

Figura 33 - randomForest train

Random Forest

```
100000 samples
  15 predictor
  2 classes: '0', '1'
```

No pre-processing

Resampling: Cross-Validated (3 fold)

Summary of sample sizes: 66666, 66667, 66667

Resampling results across tuning parameters:

mtry	Accuracy	Kappa
2	0.89970	0.79940
8	0.91423	0.82846
15	0.91220	0.82440

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 8.

Figura 34 - randomForest Model Status

Figure 34 summarizes how the machine learning model developed during training:

- 100,000 samples were used;
- We have in the model 15 predictor variables and 2 classes to predict;
- We use the Cross - Validation method with fold = 3 with each iteration having approximately 66,666 lines for training;
- And 3 accuracy values were analyzed according to the tree depth being mtry = 2, 8 and 15 where the best mtry value = 8 and 91.42% of accuracy.

Segue gráfico da acurácia:

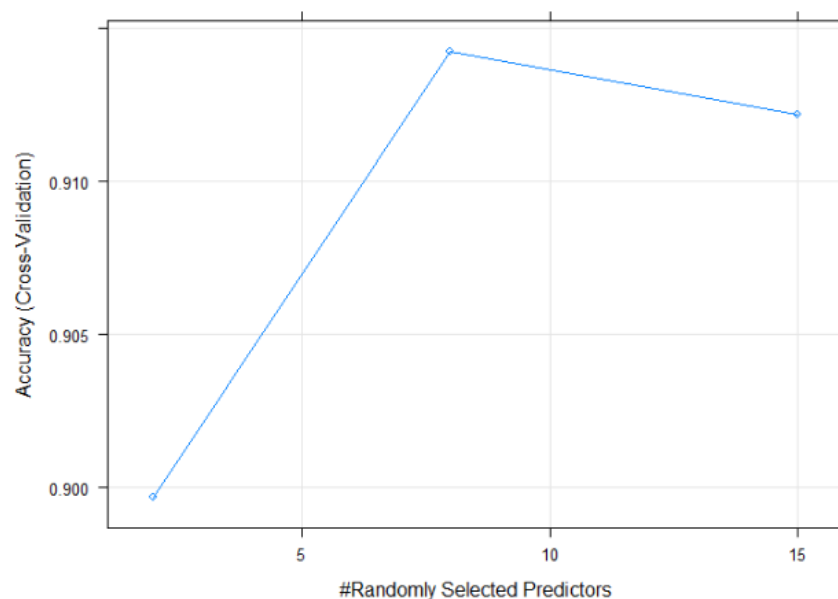


Figura 35 - Accuracy Graph of randomForest Model

3.4 Choosing the Best Machine Learning Model

Choosing the best machine learning model is questionable because it depends on a number of factors such as training time, accuracy, complexity, available data, and more where each business problem will determine your need.

Before we make a decision, let us graphically analyze the results:

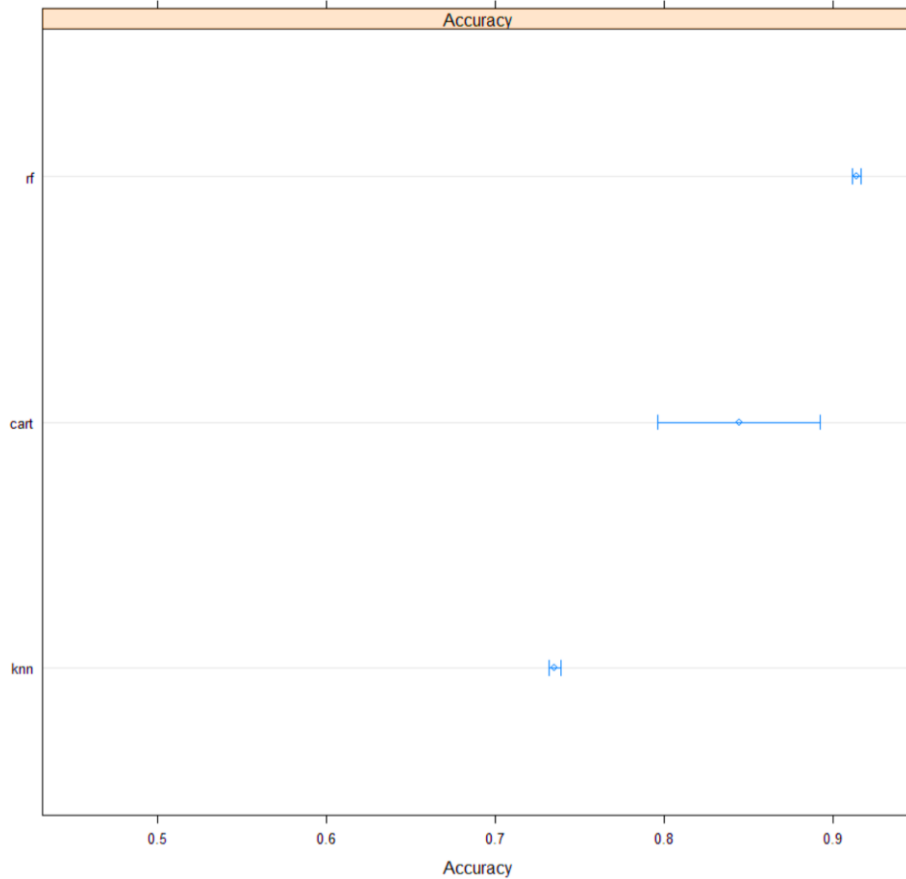


Figura 36 - Model Accuracy

As shown in Figure 36, the highest accuracy is in the randomForest (rf) model, followed by CART and kNN.

Each model had a variation in accuracy during training characterized by line sizes in figure 36, with CART being the largest variation according to the table:

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
knn	0.733657	0.734921	0.736185	0.735391	0.736256	0.736327
cart	0.822074	0.838811	0.855549	0.844411	0.855579	0.855609
rf	0.913479	0.913644	0.913809	0.914231	0.914605	0.915402

Figura 37 - Accuracy Variation

For this project we will consider Mean Accuracy ("Mean" value above) as metric.

In addition to accuracy we can identify how long each model took to be trained and then decide which model to use for validation:

Model	Time to Train	Accuracy
kNN	4 min	73,54%
CART	9 seg	84,45%
randomForest	10 min	91,42%

Figura 38 – Time to train models

In Figure 38 we can see how long it took to train randomForest when compared to CART, but as I am not looking for agility in training time but accuracy in the delivery of results I will choose to use randomForest to continue the project.

Although CART does not provide the best accuracy, it offers a huge advantage when it comes to performance. Perhaps for a corporate environment CART would be the best choice and with it a study of how to improve the parameterization of the model to gain greater accuracy and consequently surpass randomForest.

4. Making Predictions and Evaluating the Predictive Model

To evaluate the predictive model initially new and unknown data were presented to the model and then we used what we call the Confusion Matrix.

4.1 Confusion Matrix

Confusion Matrix is one of many performance gauges for evaluating data predicted by a machine learning model. We already know that the accuracy is close to 92% but what about the data that was not classified correctly? A Confusion Matrix can tell us what happened to all classified data, following image for a better understanding of its operation:

	1 (Original Data)	0 (Original Data)
1 (Predicted Data)	TruePositive	FalsePositive
0 (Predicted Data)	FalseNegative	TrueNegative

Figura 39 - Confusion Matrix

Let's interpret what the Confusion Matrix is telling us:

- TruePositive(TP): It means that my model predicted from the data provided that the class would be 1 and it really is correct, it was supposed to be 1.
- FalseNegative(FN): It means that my model predicted from the data provided that the class would be 0 but was supposed to be 1.
- FalsePositive(FP): It means that my model predicted from the data provided that the class would be 1 but also missed because it was supposed to be 0.
- TrueNegative(TN): It means that my model predicted from the data provided that the class would be 0 and this time it was right because it was supposed to be 0.

Our goal is to predict and succeed so we seek to maximize the values present in TP and TN, and on the other hand mitigate values of FN and FP.

Our machine learning model returned the following results:

	1 – Non-Fraud Click (Original Data)	0 – Possible Fraud Clicks (Original Data)
1 – Non-Fraud Click (Predicted Data)	TP = 43.865	FP = 2.264
0 – Possible Fraud Clicks (Predicted Data)	FN = 6.135	TN = 47.736

Figura 40 - Confusion Matrix my model

As the Confusion Matrix of Figure 40 illustrates, we have TP and TN with the highest values indicating that our model has excellent performance in predicting a fraudulent click or not.

Regarding the FP and FN, they have 2.26% and 6.13%, respectively, of wrong predicted data. An improvement that could be made would be to decrease the percentage of FN as these were non-fraud clicks but were considered to be fraudulent by the predictive model blocking users improperly.

10 – Final Considerations

The initial statistical analysis (percentage of distributed data) made all the difference in the project development since there was no induction for a given result but equity. Training the model with variable unbalance leads to failure in the predictive process; improving balance makes the project develop with better quality.

For this project we also studied a number of important concepts in the data science process starting with feature engineering performing data cleansing and transformation generating better opportunities for machine learning models.

By adding new variables it was possible to track the click journey, as we entered the exploratory analysis. For this was used PowerBi generating an interactive dashboard where new business opportunities were identified.

One of the opportunities found was identifying the time of most accesses (total and individualized by IP), valuable information for marketing teams who could draw a pipeline identifying the sales funnel, in other words, app downloads conversion and strategically be part of the customer journey from the first contact with the app until the moment they download.

During the exploratory phase the analysis of fraudulent clicks was developed through the complete click journey of some users and a blacklist could be created, since it was possible to identify the access pattern of each IP, analyzing the applications with more clicks and fewer downloads, devices used, access times, and more. Delivering the dashboard with reporting to decision makers in the organization will surely create a competitive advantage for the company.

Entering the creation of the machine learning model we studied an optimization method called Cross - Validation that assisted in the separation of variables during the training of CART, kNN and randomForest models that provided accuracy of 84.45%, 73.54% and 91.42% and time to train around 4 minutes, 9 seconds and 10 minutes respectively.

The choice of the model was based on accuracy rather than training time, so randomForest was used to predict unknown data returning an excellent accuracy of approximately 92% evidenced in the Confusion Matrix.

To further enhance this result it would be possible to focus more intensely on improving model parameters, or adding new feature engineering variables, or even collect more data by focusing on the resources analyzed to train the model.

Source Code

```
## Talking Data Project - Building a Predictive Model for Fraudulent Click Analysis -----

# Obs: If you have problems with accenting, see this link.:
# https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding

# Configuring the working directory
# Quote the working directory you are using on your computer.

## Directory -----
# SET WORKING DIRECTORY
setwd("C:/FCD/BigDataRAzure/Cap20/TalkingData")
# GETTING CURRENT DIRECTORY
getwd()

## Kaggle -----
# https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data

## Data Description -----
# - train.csv - the training set
# - train_sample.csv - 100,000 randomly-selected rows of training data, to inspect data before downloading full set
# - test.csv - the test set
#X- sampleSubmission.csv - a sample submission file in the correct format
#X- test_supplement.csv - This is a larger test set that was unintentionally released at the start of the competition. It is not necessary
  to use this data, but it is permitted to do so. The official test data is a subset of this data.

## Data Dictionary -----
# ip -> ip address of click
# app -> app id for marketing
# device -> device type id of user mobile phone (e.g., iphone 6 plus, iphone 7, huawei mate 7, etc.)
# os -> os version id of user mobile phone
# channel -> channel id of mobile ad publisher
# click_time -> timestamp of click (UTC)
# attributed_time -> if user download the app for after clicking an ad, this is the time of the app download
# is_attributed -> the target that is to be predicted, indicating the app was downloaded

# Note that ip, app, device, os, and channel are encoded.

## Library -----
# IMPORTING NECESSARY LIBRARIES
library(data.table)
library(dplyr)
# Using readr package
#install.packages("readr")
library(readr)      # Using to write_csv
#install.packages("RColorBrewer")
library("RColorBrewer") # Color Library to plot Graphics
library(ggplot2)
library(gridExtra)
```

```

library(lattice)
library(caret)
library(randomForest)
library(lubridate)    # So I could work with date/time in a more intuitive syntax
library(scales)

## Datasets -----
if (FALSE) {
# Loading the dataset "train_sample.csv" with 100,000 rows
# Eliminating 'attributed_time' because it is the same information as 'is_attributed', avoiding Data Leakage to the predictive model
train <- "train.csv"
train.df <- fread(train, drop = c('attributed_time'))
train.df$n <- 1:nrow(train.df) # will be used to sample data and then eliminated

# Checking if we have balanced data between 1 and 0 of target variable so it'll be a fair prediction model
nrow(filter(train.df, is_attributed == 1))/nrow(train.df)*100
nrow(filter(train.df, is_attributed == 0))/nrow(train.df)*100

# Unfortunately not, in fact we have:
# ~ 00.25% in 1 and
# ~ 99.75% in 0
# That is a real problem because I have to create an unbiased model. I'll have to deal with it.

# First let's separate all 1 data and 0 data
onedata <- filter(train.df, is_attributed == 1)
zerdata <- filter(train.df, is_attributed == 0)

# Now, as we have too many data and too less memory, let's get a sample of 50.000 of both files
set.seed(123)
samp <- createDataPartition(y = onedata$n, p=0.1096, list = FALSE)
set.seed(123)
onedata_train <- onedata[samp,]
onedata_train <- onedata_train[1:50000,]
set.seed(123)
onedata_test <- sample_n(onedata[-samp,], 50000)

set.seed(321)
samp <- createDataPartition(y = zerdata$n, p=0.0002711, list = FALSE)
set.seed(321)
zerdata_train <- zerdata[samp,]
zerdata_train <- zerdata_train[1:50000,]
set.seed(321)
zerdata_test <- sample_n(zerdata[-samp,], 50000)

# Binding balanced data
train_sample.df <- rbind(onedata_train, zerdata_train)
test_sample.df <- rbind(onedata_test, zerdata_test)
train_sample.df$n <- NULL
test_sample.df$n <- NULL

# Checking again if we have balanced data between 1 and 0 of target variable so it'll be a fair prediction model

```

```

nrow(filter(train_sample.df, is_attributed == 1))/nrow(train_sample.df)*100
nrow(filter(train_sample.df, is_attributed == 0))/nrow(train_sample.df)*100

nrow(filter(test_sample.df, is_attributed == 1))/nrow(test_sample.df)*100
nrow(filter(test_sample.df, is_attributed == 0))/nrow(test_sample.df)*100

# Now we have 100.000 data:
# - 50.00% in 1 and
# - 50.00% in 0

#View(train_sample.df)
#View(test_sample.df)

# Cleaning memory
rm(train)
rm(train.df)
rm(samp)
rm(onedata)
rm(zerdata)
rm(onedata_test)
rm(onedata_train)
rm(zerdata_test)
rm(zerdata_train)
}
## Just to use after first dataset loaded ----
if (TRUE) {
#write_csv(train_sample.df, "20A-tr.csv")
#write_csv(test_sample.df, "20A-ts.csv")
train_sample.df <- fread("20A-tr.csv")
test_sample.df <- fread("20A-ts.csv")
}

## Feature Engineering -----

# Transforming to China TimeZone, just to use at PowerBi
#train_sample.df$click_time <- ymd_hms(train_sample.df$click_time) + hours(8)
#test_sample.df$click_time <- ymd_hms(test_sample.df$click_time) + hours(8)

# First of all I will unite both datas to one data and deal with all Feature Engineering
# Control Variable
train_sample.df$control <- 1
test_sample.df$control <- 0

# Binding
datatemp <- rbind(train_sample.df, test_sample.df)

# Just for convenience I will rename the predictor variable 'is_attributed' to 'target'
datatemp$target <- datatemp$is_attributed
datatemp$is_attributed <- NULL

# CHECKING FOR MISSING VALUES: NO!

```

```

any(is.na(datatemp))

# CHECKING VARIABLE TYPES: ALL ARE int TYPE EXCEPT click_time THAT IS char
glimpse(datatemp)

# As we can see, all variables are 'int' type, but 'click_time' that is char
# click_time should be in Date-Time format, let's make some changes:

# Transforming to dttm
datatemp$click_time <- ymd_hms(datatemp$click_time)

# Creating New Variables just by separating the date and time of 'click_time'
# Year
datatemp$year <- year(datatemp$click_time)

# Month
datatemp$month <- month(datatemp$click_time)
#datatemp$Month <- month(datatemp$click_time, label = TRUE) # if month labeled wanted

# Day
datatemp$day <- day(datatemp$click_time)

# Hour
datatemp$hour <- hour(datatemp$click_time)

# Minutes
datatemp$minutes <- minute(datatemp$click_time)

# Second
datatemp$second <- second(datatemp$click_time)

# yday
datatemp$yday <- yday(datatemp$click_time)

# mday
#datatemp$mday <- mday(datatemp$click_time)

# wday
datatemp$wday <- wday(datatemp$click_time)
#datatemp$wday <- wday(datatemp$click_time, label = TRUE) # if wday labeled wanted

# week
datatemp$week <- week(datatemp$click_time)

# Internal integer representation of click_time
datatemp$click_time <- unclass(datatemp$click_time)

# Separating train and test again
train_sample.df <- filter(datatemp, control == 1)
train_sample.df$control <- NULL
test_sample.df <- filter(datatemp, control == 0)

```

```

test_sample.df$control <- NULL

# Rescaling click_time from 0 to 1 to get better results
# Note: I'm not changing the values, just the scale
ggplot(data = train_sample.df, aes(click_time)) +
  geom_density(kernel = 'gaussian')
#hist(train_sample.df$click_time)
train_sample.df$click_time <- rescale(train_sample.df$click_time)
test_sample.df$click_time <- rescale(test_sample.df$click_time)

# Plotting Rescaled click_time histogram
ggplot(data = train_sample.df, aes(click_time)) +
  geom_density(kernel = 'gaussian')
#ggplot(data = test_sample.df, aes(click_time)) +
# geom_density(kernel = 'gaussian')

# Cleaning Memory
rm(datatemp)

# CSV to Power BI
#write_csv(train_sample.df, "20A-PowerBiData.csv")

## Exploratory Analysis -----
# All exploratory analysis were made and transported to PowerBi, so I won't explore it here
if (FALSE) {
# IP's with most access
ipmostac <- train_sample.df %>%
  select(ip) %>%
  count(ip) %>%
  arrange(desc(n))

View(ipmostac[1:20,1:2])

# App's with most access
apmostac <- train_sample.df %>%
  select(app) %>%
  count(app) %>%
  arrange(desc(n))

View(apmostac[1:20,1:2])

# Device most used
dvmostac <- train_sample.df %>%
  select(device) %>%
  count(device) %>%
  arrange(desc(n))

View(dvmostac[1:10,1:2])

# Now let's study the click journey of IP on top
topip <- as.numeric(ipmostac[1,1])

```

```
# App's most used by IP on top
topipap <- train_sample.df %>%
  select(ip, app) %>%
  filter(ip == topip) %>%
  count(ip, app) %>%
  arrange(desc(n))
```

View(topipap)

```
# How Many access per day does the top ip
topipdy <- train_sample.df %>%
  select(ip, day) %>%
  filter(ip == topip) %>%
  count(ip, day) %>%
  arrange((day))
```

View(topipdy)

```
# When does top ip access per day
topiphr <- train_sample.df %>%
  select(ip, day, hour) %>%
  filter(ip == topip) %>%
  count(ip, day, hour) %>%
  arrange(day)
```

View(topiphr)

```
# App's downloaded or not by top ip
topipdl <- train_sample.df %>%
  select(ip, app, target) %>%
  filter(ip == topip) %>%
  count(ip, app, target)
```

View(topipdl)

```
# Cleaning Memory
rm(ipmostac)
rm(apmostac)
rm(dvmostac)
rm(topip)
rm(topipap)
rm(topipdy)
rm(topiphr)
rm(topipdl)
}
```

```
## Machine Learning I -----
# Using Cross Validation I'll run all algorithms with 3 fold
trctrl <- trainControl(method="cv", number=3)
# Metric to get mesures of erros
```

```

evaluation <- "Accuracy"

# Building Models
# Target as factor to make predictions
train_sample.df$target <- as.factor(train_sample.df$target)

# kNN
Inicio <- print(Sys.time())
set.seed(9)
model_knn <- train(target~., data=train_sample.df,
                  method="knn", metric=evaluation,
                  trControl=trctrl)
Fim <- print(Sys.time())
cat('Processo kNN Finalizado em: ', Fim-Inicio, ' segundos!')
plot(model_knn)

# CART
Inicio <- print(Sys.time())
set.seed(9)
model_cart <- train(target~., data=train_sample.df,
                  method="rpart", metric=evaluation,
                  trControl=trctrl)
Fim <- print(Sys.time())
cat('Processo CART Finalizado em: ', Fim-Inicio, ' segundos!')
plot(model_cart)

# Random Forest
Inicio <- print(Sys.time())
set.seed(9)
model_rf <- train(target~., data=train_sample.df,
                  method="rf", metric=evaluation,
                  trControl=trctrl)
Fim <- print(Sys.time())
cat('Processo RF Finalizado em: ', Fim-Inicio, ' segundos!')
plot(model_rf)
plot(model_rf$finalModel)

# Cleaning Memory
rm(trctrl)
rm(evaluation)

# Gathering accuracy of models
acc_result <- resamples(list(knn=model_knn, cart=model_cart, rf=model_rf))
summary(acc_result)

# Plotting Models Accuracy and Kappa
dotplot(acc_result)

# See model
print(model_rf)

# Making some predictions and evaluating results using a Confusion Matrix
test_sample.df$target <- as.factor(test_sample.df$target)
test <- test_sample.df
test$target <- NULL

pred_rf <- predict(model_rf, test)
confusionMatrix(pred_rf, test_sample.df$target)

```