Advertisement Tracking Fraud Detection

1 Introduction

Fraud risk is everywhere, but for companies that advertise online, click fraud can happen at an overwhelming volume, resulting in misleading click data and wasted money. Ad channels can drive up costs by simply clicking on the ad on a large scale. With over 1 billion smart mobile devices in active use every month, China is the largest mobile market in the world and therefore suffers from huge volumes of fraudulent traffic.

TalkingData is the largest independent big data service platform in China. Their current approach to preventing click fraud for app developers is to measure the journey of a user's click across their portfolio, and flag IP addresses who produce lots of clicks, but barely end up installing apps. With this information, they've built an IP blacklist and device blacklist. However, the desired success of the prevention is to be one step ahead of fraudsters to predict whether a user will download an app after clicking a mobile app ad. As a result, a data mining and analysis model should be built based on the dataset provided by the company, which covers approximately 200 million clicks over 4 days.

In generality, the sections included are summarized as follows: the pre-modeling data statistical analysis in section 2; the model algorithm explanation in section 3 which involving input engineering, classification, model tuning and estimation; next, the competition method with random forest on data mining, and finally a conclusion. All aforementioned tasks are done with R.

2 Statistic analysis on raw/original data

2.1 Data selection

As known, the dataset is huge up to 7 GB, which significantly slows down the running and testing of our model. In order to fast check the correctness of our model, the preparatory work is to select a sample data without losing the data structure. In the sample data, 1% of total data are randomly chosen. The size of the sample data is 81MB with 1,850,000 instances, comparing to 184 million instances of the original training dataset.

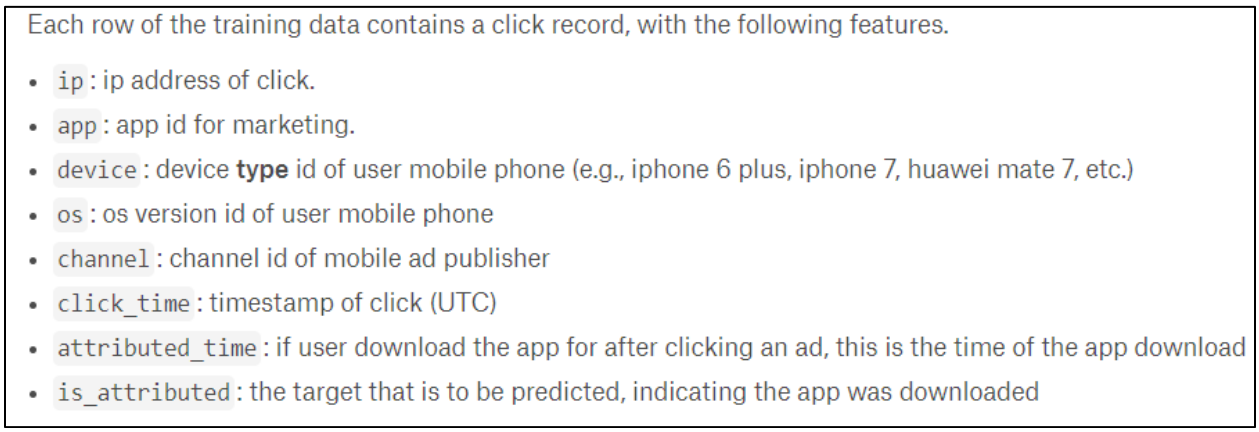
2.2 Statistic analysis

In the dataset, each row of the training data contains a click record with the attributes of ip, app, device, os, channel, click time, attributed_time, and is_attributed. The detailed attribute explanation in Figure 1 is from the Kaggle competition website:
https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data.

In this section, we will first explore the basic statistic of the sample data, then address some characteristic that will be used in determine fraud clicks. We plot the hourly number of clicks by 4 types of attributes, that is, "ip, app, device, channel", in Beijing time. We also plot the number of clicks vs. download probability by those 4 types attributes. For a dynamic view please visit this Tableau profile and click the "IE 583 Final Project":
https://public.tableau.com/profile/luning#!/.

Among the results in the profile, two attributes perform obviously abnormally, which are the "app" and "device". In Figure 2-1 to 2-4, orange line represents the download probability of each type instance. The black horizontal solid line is the average value of the download probability in this

type. The blue bar is the number of click of the of each type instance. The left vertical axis is the number of click scale of the attribute type, the right vertical axis is the download probability scale of the attribute type.


Each row of the training data contains a click record, with the following features.

- `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

Figure 1 Attribute Description

Figure 2-1 and 2-2 show the number of clicks vs. download probability of different devices. The total clicks of device 1 and 2 occupy 98% of all clicks. The gap of clicks by other devices are huge, which *obviously differs from the actual market share of smart phone in China*. If we exclude device 1 and 2, as shown in Figure 2-2, we can see there are still some devices (3032, 3543, 3866, 59 and 5) having a relatively high number of clicks 0 download rates, as a comparison, the average download probability of all devices is 15%.

The *is_attributed* field should NOT be used as the class, because normal users may click advertisement but do not download the app (not interested, have advertisements, charge inside the App, etc.).

Therefore, we build a new class attribute called “fraud” to describe the suspect fraud activity. The clicks from device 3032, 3543, 3866, 59 and 5, and all *non-download* clicks from device 1 and 2 are determined as frauds. All other data are marked as non-fraud. Note that device 0 has a download probability of 9%, which is close to the average download probability. In this analysis, device 0 is not determined as fraud activities.

We can also see that several apps also show distinct activities in Figure 2-3 and 2-4. The format of the graph is the same as above, except that these graphs are down sorted by download probability. It can be seen in Figure 2-3 fraud activities are concentrated within several apps. Figure 2-4 shows the zoomed in the figure of Figure 2-3 of those abnormal apps. These apps have a high number of click by extremely low download rate, which is almost 0%, comparing to the average download rate of all app, which is about 9%. To be clearer, these apps are the victims of the fraud ad clicks.

Although the apps certainly have some abnormal feature, we did not use it to determine the fraud class. The reasons are: (1) about half of the potential abnormal apps in Figure 2-4, such as app 32, 27, 17 and so on, are difficult to directly be defined as frauds or non-frauds without further information. The number of clicks of these apps is not distinguishable from other apps that have higher or lower download rate (2) There is an expense of having more than one device for people who make fraud clicks. But click more than one app from a fraud ad clicking device have almost no cost. As a result, the engineering input of abnormal apps is relatively weak to determine the fraud clicks.

In summary, in the 1.85 million sample dataset, there are millions of fraud behaviors from very few devices type on several victim apps.

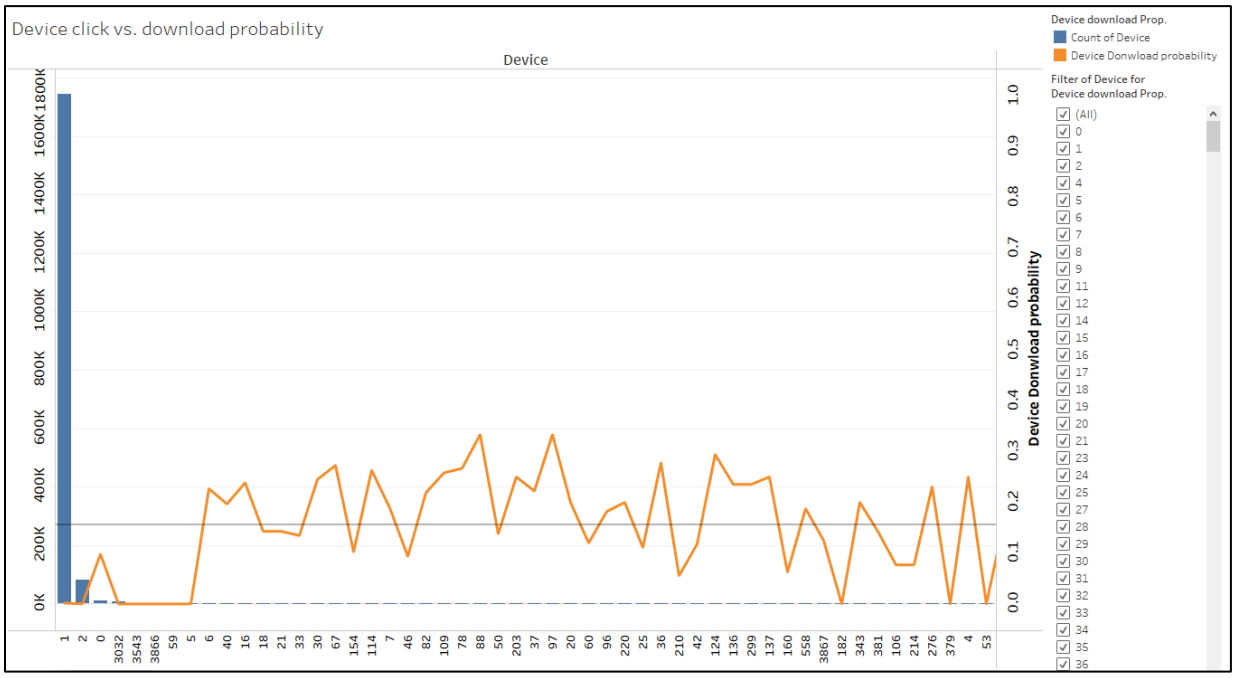


Figure 2-1 Number of click vs. download probability by device

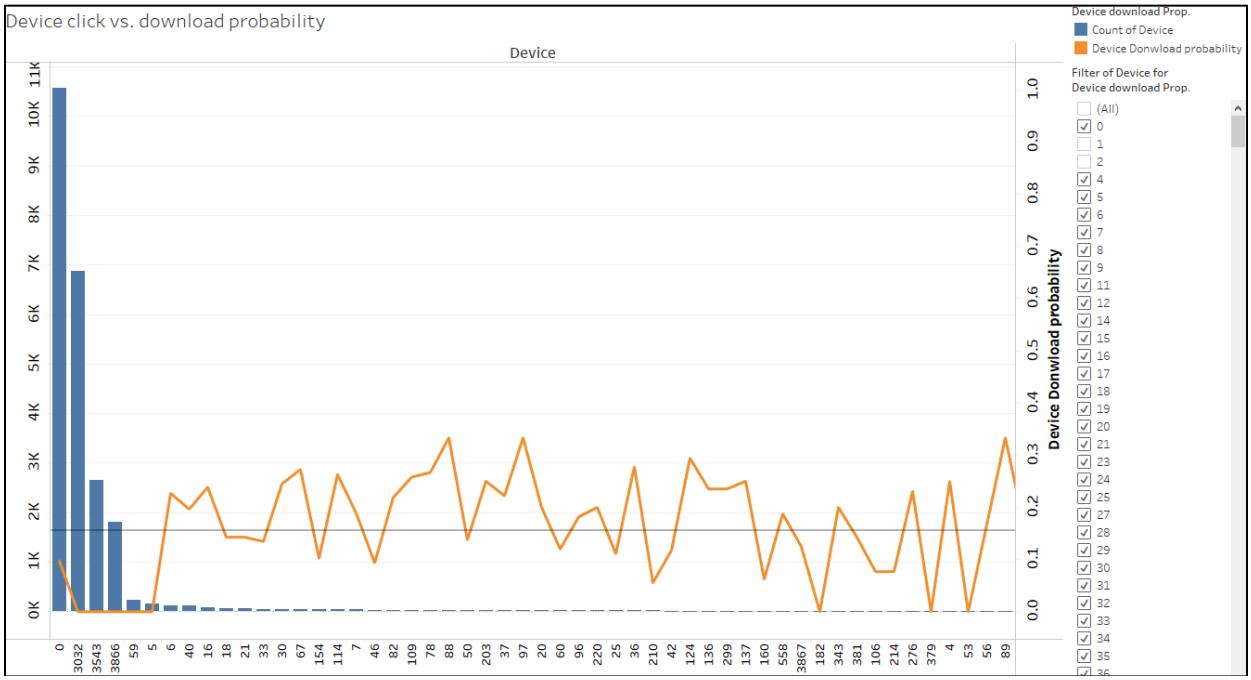


Figure 2-2 Number of click vs. download probability by device, exclude Device 1 and 2

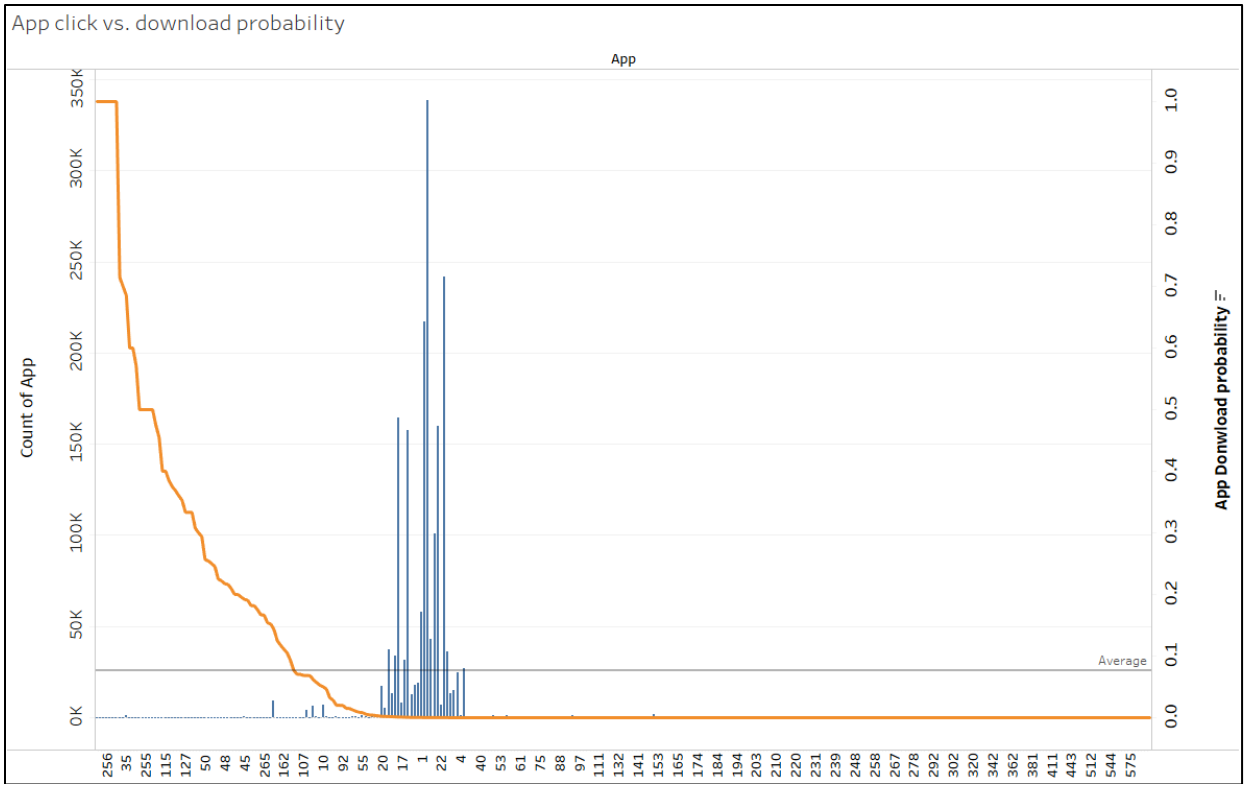


Figure 2-3 Number of click vs. download probability by app

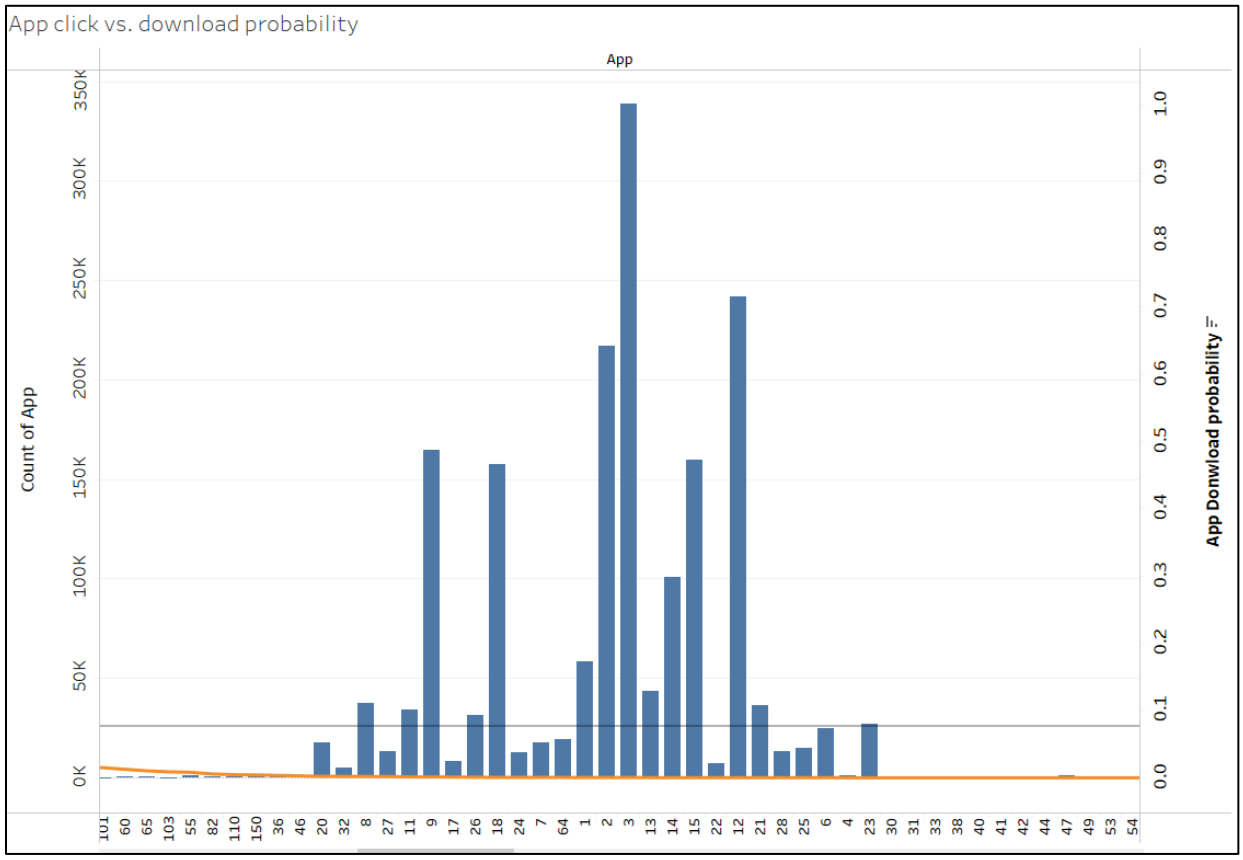


Figure 2-4 Number of click vs. download probability by app, zoomed onto abnormal apps

3 Model algorithms

In this section, we mainly introduce the analysis process and the outputs from R coding. Four parts are included, preparation of data analysis, input engineering application, independent test approach with decision tree model, and cross-validation approach with random forest approach.

3.1 Preparation of analysis

First of all, load some libraries.

```
library(dplyr)
library(pacman)
library(data.table)
library(e1071)
library(ggplot2)
library(DMwR)
library(caret)
library(randomForest)
library(RWeka)
```

The next is for data reading. The train dataset given by Kaggle has about 185 million observations and 7 GB. This results in difficulty in manipulating data and building models in R. Therefore, we randomly select 1% of data with 1.85 million rows. We have saved the sampled dataset as SampleData.csv. The dataset is very clean without any missing value.

The R code below is for the 1% data selection mentioned in section 2.1.

```
set.seed(583)
AllData <- fread('train.csv', header = T, sep = ',')
SampleData <- sample_n(AllData, 1850000, replace = FALSE)
write.csv(SampleData, file = "SampleData.csv",row.names=FALSE)
rm(AllData)
rm(SampleData)
df <- read.csv("SampleData.csv")
#check missing values
colSums(is.na(df))
##            ip           app        device            os
##             0             0             0             0
##       channel    click_time attributed_time   is_attributed
##             0             0             0             0
```

3.2 Input engineering application

Divide existing attribute

Comparing to other attributes, the click_time is an exact time and hard to be used on any method. Thus, we convert click_time into 2 fields: wday (day of week) and hour (1-24). Then remove click_time and attributed_time since they will not be used anymore.

```
df$click_time <-as.POSIXct(df$click_time,format="%Y-%m-%d %H:%M:%S",tz='UTC')
#covert to China timezone.
attributes(df$click_time)$tzone <- "Asia/Shanghai"
#extract day of week and hour
df$wday <- factor(weekdays(df$click_time))
```

```
df$hour <- factor(hour(df$click_time))
#remove unuseful timestamps
df$attributed_time <- NULL
df$click_time <- NULL
```

Check how many unique values in each field.

```
apply(df, 2, function(x) length(unique(x)))
##           ip          app       device           os      channel
##        96631          322          579          237          173
## is_attributed         wday         hour
##            2            5           24
```

Create class attribute

Create class to indicate whether clicks are fraud or not. We have found that the clicks by device (1, 2, 3032, 3543, 3866, 59, 5) are very suspicious. These devices have way more clicks than the other devices, but very few or even none of which lead to download. There devices are very likely used widely by fraud click makers. However, there are some clicks with download for device (1, 2). We believe these clicks come from normal users, as described in Section 2. There is a class imbalance problem (only 0.88% of classes are non-fraud clicks).

```
df$class <- 'n' #non-fraud
df$class[df$device %in% c(3032,3543,3866,59,5)] <- 'f' #fraud
df$class[(df$device %in% c(1,2)) & (df$is_attributed==0)] <- 'f'
df$class <- factor(df$class)
table(df$class)
##
##       f       n
## 1833796   16204
```

Add new variables

We count the number of clicks associated with each attribute, and the rate of downloading apps. We also count the number of clicks by 2 fields and time, since fraud clicks likely occur continuously and intensively during a short time.

```
#add click count variables
df %<>%
  add_count(ip) %>% setnames("n", "n_ip") %>%
  add_count(app) %>% setnames("n", "n_app") %>%
  add_count(device) %>% setnames("n", "n_device") %>%
  add_count(os) %>% setnames("n", "n_os") %>%
  add_count(channel) %>% setnames("n", "n_channel")

#add app download rate variables
df %<>%
  group_by(ip) %>%
  mutate(r_ip=sum(is_attributed)/length(is_attributed)) %>%
  ungroup() %>%
  group_by(app) %>%
  mutate(r_app=sum(is_attributed)/length(is_attributed)) %>%
  ungroup() %>%
```

```
  group_by(device) %>%
  mutate(r_device=sum(is_attributed)/length(is_attributed)) %>%
  ungroup() %>%
  group_by(os) %>%
  mutate(r_os=sum(is_attributed)/length(is_attributed)) %>%
  ungroup() %>%
  group_by(channel) %>%
  mutate(r_channel=sum(is_attributed)/length(is_attributed)) %>%
  ungroup()

#add click counts by two attributes and by hour
df %<>%
  add_count(device, ip, hour) %>%
  setnames("n", "n_device_ip_h") %>%
  add_count(device, app, hour) %>%
  setnames("n", "n_device_app_h") %>%
  add_count(device, os, hour) %>%
  setnames("n", "n_device_os_h") %>%
  add_count(device, channel, hour) %>%
  setnames("n", "n_device_channel_h")

#add app download rate variables
df %<>%
  group_by(device, ip, hour) %>%
  mutate(r_device_ip_h=sum(is_attributed)/length(is_attributed)) %>%
  ungroup() %>%
  group_by(device, app, hour) %>%
  mutate(r  device_app_h=sum(is_attributed)/length(is_attributed)) %>%
  ungroup() %>%
  group_by(device, os, hour) %>%
  mutate(r_device_os_h=sum(is_attributed)/length(is_attributed)) %>%
  ungroup() %>%
  group_by(device, channel, hour) %>%
  mutate(r_device_channel_h=sum(is_attributed)/length(is_attributed)) %>%
  ungroup()
```

3.3 Independent test with decision tree

3.3.1 Independent data

Split date into training and test datasets. We will use the independent test set to test models. The training data is 70% of the entire sample dataset, the left 30% is the test data.

```
set.seed(583)
index <- createDataPartition(df$class, times=1, p=0.7, list=FALSE)
train <- df[index, ]
table(train$class)
##       f       n
## 1283658   11343
test <- df[-index, ]
table(test$class)
```

```
##      f      n
## 550138   4861
```

3.3.4 Data Imbalance

In our sample data, the instances having 1 in the class attribute is obviously less than those with 0. To deal with the class imbalanced problem, we use down-sampling. Down-sampling will randomly sample a data set so that all classes have the same frequency as the minority class.

```
#down-sampling
train_down <- downSample(x=train[, -9],y=train$class)
table(train_down$Class)
##     f     n
## 11343 11343
```

3.3.5 Modeling

Build decision tree model based on the down-sampled train data, and test the model using the independent test set. From the confusion matrix, the model accuracy is 0.9996. All non-fraud clicks can be correctly predicted. And 99.96% of fraud clicks can be detected by the model. Figure 3 plots the structure of the decision tree, where dark color represents non-fraud clicks and light color represents fraud clicks.

```
set.seed(583)
model_DT <- train(Class~.,data=train_down,method='J48',trControl=trainControl(method="cv"
))
model_DT
## C4.5-like Trees
## 22686 samples
##    26 predictor
##     2 classes: 'f', 'n'
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 20418, 20417, 20418, 20418, 20416, 20417, ...
## Resampling results across tuning parameters:
##   C      M  Accuracy   Kappa
##   0.010  1  0.9996915  0.9993830
##   0.010  2  0.9996915  0.9993830
##   0.010  3  0.9996915  0.9993830
##   0.255  1  0.9997796  0.9995593
##   0.255  2  0.9997796  0.9995593
##   0.255  3  0.9997796  0.9995593
##   0.500  1  0.9997796  0.9995593
##   0.500  2  0.9997796  0.9995593
##   0.500  3  0.9997796  0.9995593
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were C = 0.255 and M = 1.
prediction_DT = predict(model_DT, test)
confusionMatrix(prediction_DT, test$class)
## Confusion Matrix and Statistics
##          Reference
## Prediction      f      n
```

```
##          f 549907      0
##          n    231   4861
##               Accuracy : 0.9996
##                 95% CI : (0.9995, 0.9996)
##    No Information Rate : 0.9912
##    P-Value [Acc > NIR] : < 2.2e-16
##
##                  Kappa : 0.9766
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9996
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9546
##              Prevalence : 0.9912
##          Detection Rate : 0.9908
##    Detection Prevalence : 0.9908
##       Balanced Accuracy : 0.9998                          ##        'Positive' Class : f
```

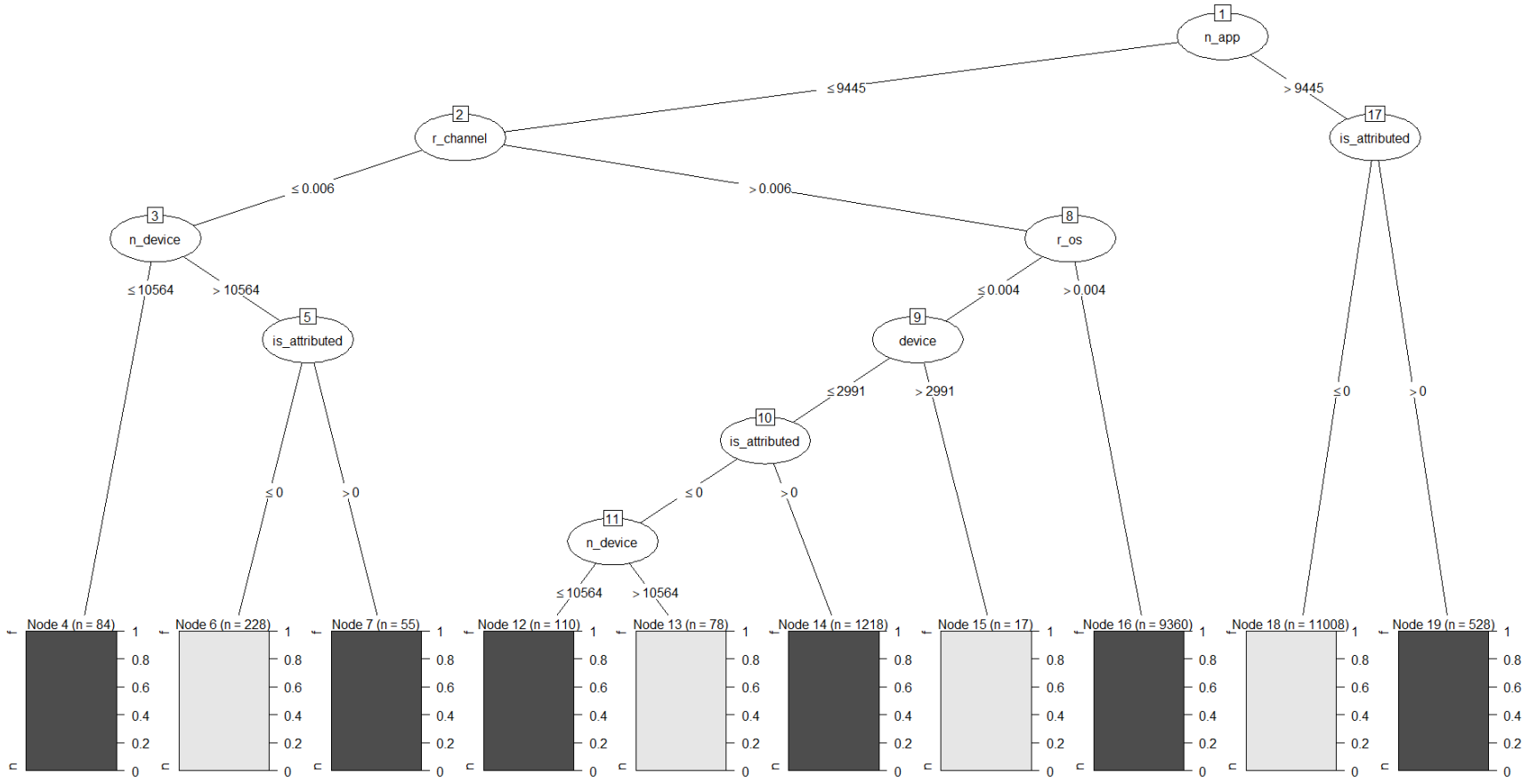


Figure 3 The decision tree using for prediction and estimation

3.4 Cross-validation with random forest approach

Use the whole sample data as the original input data, split the data into 10-fold, and use a for loop to treat each 1-fold as test data and the other 9 folds as training data. The training data is down-sampled to meet the number of minority class, and an inside 10-fold CV loop is used to train the random forest model. It spent 13.5 hours in R to fit and test the model. The model accuracy is even higher with almost 1. In the model, 5 or 9 attributed were selected.

```
#random forest
ctrl = trainControl(method="cv",number=10,savePred=T,classProb=T)
Grid_RF <- expand.grid(.mtry=c(1:15))
folds = split(sample(nrow(df), nrow(df),replace=FALSE), as.factor(1:10))
best.k = NULL
train.accuracy.estimate = NULL
fold.accuracy.estimate = NULL
f=1
for(f in 1:10){
  trainingData = df[-folds[[f]],]
  testData = df[folds[[f]],]
  train_down <- downSample(x=trainingData[, -9],y=trainingData$class)
  model_RF = train(Class~.,data=train_down, method="rf",trControl=ctrl,tuneGrid=Grid_RF)
  best.k[f] = as.numeric(model_RF$bestTune)
  train.accuracy.estimate[f] = as.numeric(model_RF$results[best.k[f],2])
  fold.accuracy.estimate[f] = (table(predict(model_RF,testData),testData$class)[1,1]+table(pre
dict(model_RF,testData),testData$class)[2,2])/length(testData$class)
}
mean(train.accuracy.estimate)
## [1] 0.9999931
mean(fold.accuracy.estimate)
## [1] 0.9999427
best.k
##  [1] 11  9  5  5  5  5  5  6 10  9
```