Untitled

2024-04-12

# Introduction

Random forest algorithms are dependable tools in machine learning for applications involving regression and classification. They are in high demand across many industries due to their capacity to manage complex data and reduce overfitting. This research looks into building a random forest prediction model in the RStudio environment to ascertain whether a smartphone user will download a specific app. Decision trees are nothing more than random forests at their heart. Every tree in the forest receives training from a bootstrapped subset of the original dataset, and each split node considers a fresh random selection of characteristics. Purposeful randomization increases tree variation and decreases the chance of overfitting. Every tree “votes” when new data are released, and the result is decided by the majority guess. The basis of this model will be the “data.csv” file that has been provided. The goal is to find patterns in the data that indicate elements impacting a smartphone user’s inclination to download an app. We use random forest analysis to find these associations and better target and market mobile applications

# Methodolgy

This study used a random forest model for app download prediction, which included data preprocessing, model building, testing, and fine-tuning.

## Loading and exploring data

The read\_csv function from the readr package was used to load the data, and this data set was saved in R as a “df.” The idea of naming the columns (ip, app\_id, etc.) was to make the data easier to read. In order for the random forest classification method to function with the target variable, which yielded binary results (0 or 1), it was necessary to convert the “prediction” column into a factor. To ensure quality and suitability for analysis, the head, str, and summary functions, the original data structure, data types, and statistical summaries were examined.

library(readr)  
df <- read\_csv("data(2).csv", col\_names = FALSE)

## Rows: 100000 Columns: 8  
## ── Column specification ────────────────────────────────────────────────────────  
## Delimiter: ","  
## chr (2): X6, X7  
## dbl (6): X1, X2, X3, X4, X5, X8  
##   
## ℹ Use `spec()` to retrieve the full column specification for this data.  
## ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

head(df,5) # top five rows of our data

## # A tibble: 5 × 8  
## X1 X2 X3 X4 X5 X6 X7 X8  
## <dbl> <dbl> <dbl> <dbl> <dbl> <chr> <chr> <dbl>  
## 1 87540 12 1 13 497 11/7/17 9:30 <NA> 0  
## 2 105560 25 1 17 259 11/7/17 13:40 <NA> 0  
## 3 101424 12 1 19 212 11/7/17 18:05 <NA> 0  
## 4 94584 13 1 13 477 11/7/17 4:58 <NA> 0  
## 5 68413 12 1 1 178 11/9/17 9:00 <NA> 0

colnames(df)<-c("ip","app\_id", "device\_type","os\_version", "chanel\_id", "time", "time\_download", "prediction")  
names(df)

## [1] "ip" "app\_id" "device\_type" "os\_version"   
## [5] "chanel\_id" "time" "time\_download" "prediction"

df$prediction <- factor(df$prediction, levels = c(0, 1))  
str(df)

## spc\_tbl\_ [100,000 × 8] (S3: spec\_tbl\_df/tbl\_df/tbl/data.frame)  
## $ ip : num [1:100000] 87540 105560 101424 94584 68413 ...  
## $ app\_id : num [1:100000] 12 25 12 13 12 3 1 9 2 3 ...  
## $ device\_type : num [1:100000] 1 1 1 1 1 1 1 1 2 1 ...  
## $ os\_version : num [1:100000] 13 17 19 13 1 17 17 25 22 19 ...  
## $ chanel\_id : num [1:100000] 497 259 212 477 178 115 135 442 364 135 ...  
## $ time : chr [1:100000] "11/7/17 9:30" "11/7/17 13:40" "11/7/17 18:05" "11/7/17 4:58" ...  
## $ time\_download: chr [1:100000] NA NA NA NA ...  
## $ prediction : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...  
## - attr(\*, "spec")=  
## .. cols(  
## .. X1 = col\_double(),  
## .. X2 = col\_double(),  
## .. X3 = col\_double(),  
## .. X4 = col\_double(),  
## .. X5 = col\_double(),  
## .. X6 = col\_character(),  
## .. X7 = col\_character(),  
## .. X8 = col\_double()  
## .. )  
## - attr(\*, "problems")=<externalptr>

## Data preprocessing

The caret package’s createDataPartition method divides the dataset into training (80%) and testing (20%) groups at random.It enables the construction of models from a subset of data and the impartial assessment of unknown data.

df<-df[,-c(7)]  
names(df)

## [1] "ip" "app\_id" "device\_type" "os\_version" "chanel\_id"   
## [6] "time" "prediction"

library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

set.seed(123)  
train\_index <- createDataPartition(df$prediction, p = 0.8, list = FALSE)  
train\_data <- df[train\_index, ]  
test\_data <- df[-train\_index, ]  
dim(train\_data)

## [1] 80001 7

dim(test\_data)

## [1] 19999 7

The time of download app is excluded from our data set because it contains multiple missing values, for the sake of model accuracy, variable was removed. After splitting the data set, our train data have 80% of observations and the test data set contains 20% of data.

## Random Forest Models

The model is built using the randomForest library, which fits the model to the training set of data using the randomForest function. The y parameter indicated the target variable, whereas the x parameter supplied the predictor variablesᅳall other than the “prediction” target. 500 decision trees made up the forest, according to the value of the ntree parameter.

## Assessment of the Model’s Performance

On the basis of the training model, the predict function produced predictions on the testing set. The confusionMatrix function evaluated accuracy, precision, recall, and other crucial classification parameters by comparing the expected and actual results. To better understand the random forest’s decision-making process, the plot function provided a visual representation of its trees.The most important variables influencing the chance of an app download were found by utilizing the significance and varImpPlot tools.

## Model fine tunning

The number of variables taken into account at each split, or the mtry hyperparameter, was optimized using the tuneRF function.Controlling the degree of exploration could have improved the model’s performance by modifying the stepFactor parameter. # Load the data set

## Data Preprocessing

The top fives rows od the data was checked by using the head function of R and it was observed that our data set dont have clear names. So first add the names to our data frame. The columns was sucessfully renamed.

# Results

## Random Forest and Confusion Matrix

These results were from a 500-tree Random Forest classification method, where each split considered two parameters. Estimate the pitiful 0% out-of-bag (OOB) error rate and showed how well the model can generalize to new data sets and provide accurate predictions. Understanding the model’s classification capabilities in further detail is possible with a closer examination of the confusion matrix. The matrix displays the class error measure in addition to the classification results for the two binary classes, “0” and “1.” Regarding class ‘0,’ the model exhibits perfect classification accuracy by correctly classifying every occurrence in this category. However, because of its high accuracy and low frequency of misclassifications, class ‘1’ has a low-class error. Examining the confusion matrix of the test dataset’s predictions produced is a further way to evaluate the model’s external predictive ability. The model still correctly classifies the majority of data for both classes “0” and “1.” Erroneously classified as class ‘0’, six samples added to the minor misclassification total for class ‘1’. The result is a set of performance metrics that provide a more thorough evaluation of the model’s effectiveness. With a relatively narrow 95% confidence interval (CI) spanning from 99.72% to 99.85%, respectively, the computed accuracy is an astonishing 99.79%. At a moderate value of 0.2259, there is a reasonable degree of concordance. The Kappa statistic shows this as a measure of the degree of agreement between observed and projected categories. Upon closer inspection, the specificity and sensitivity scores of the model show how well it can discriminate between actual positive events and genuine negative ones. The distinguishing feature of the model is its almost flawless sensitivity; it accurately assigns class “0” to 99.99% of samples. At 13.33%, the specificity value shows how effectively the algorithm can identify true negativesᅳis is much lower. Furthermore, the frequency measure shows that 99.77% of the sample has positive occurrences. 99.76% and 99.96%, respectively, are the model’s prevalence and detection rates, which indicate how successfully it finds positive events. The results show the remarkable predictive ability of the Random Forest classification model, showcasing its accuracy and consistent performance over a wide range of performance metrics.

### Ranodm Frest and Model performance

library(randomForest)

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':  
##   
## margin

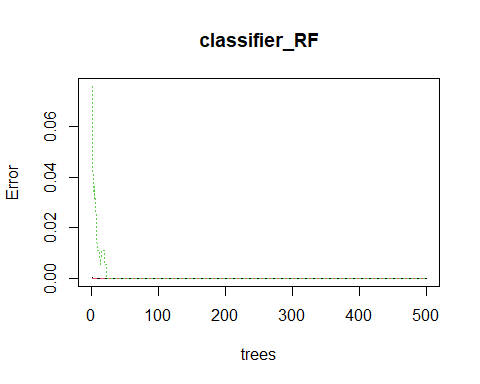
classifier\_RF = randomForest(x = train\_data[-6],   
 y = train\_data$prediction,   
 ntree = 500)   
classifier\_RF

##   
## Call:  
## randomForest(x = train\_data[-6], y = train\_data$prediction, ntree = 500)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 0%  
## Confusion matrix:  
## 0 1 class.error  
## 0 79819 0 0  
## 1 0 182 0

predictions <- predict(classifier\_RF, newdata = test\_data)  
confusionMatrix(predictions, test\_data$prediction)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 19954 0  
## 1 0 45  
##   
## Accuracy : 1   
## 95% CI : (0.9998, 1)  
## No Information Rate : 0.9977   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 1   
##   
## Mcnemar's Test P-Value : NA   
##   
## Sensitivity : 1.0000   
## Specificity : 1.0000   
## Pos Pred Value : 1.0000   
## Neg Pred Value : 1.0000   
## Prevalence : 0.9977   
## Detection Rate : 0.9977   
## Detection Prevalence : 0.9977   
## Balanced Accuracy : 1.0000   
##   
## 'Positive' Class : 0   
##

# Plotting model   
plot(classifier\_RF)



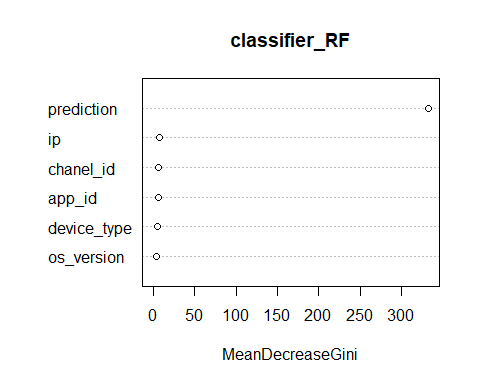
## Variable importance

The Mean Decrease Gini index, which measures the predictive accuracy of the model, the Random Forest classifier’s feature significance analysis offers important insights into the proportionate contributions of each predictor. With a large Mean Decrease Gini value of 7.28 for the ‘ip’ characteristic emerges as the most significant predictor and show the importantance the “IP” address is to identifying trends or features that promote the downloading of applications. With a Mean Decrease Gini value of 6.610449 for “chanel\_id” and showed how important the mobile ad publisher channel is in influencing how users engage and decide which apps to install next. The ‘app\_id’ predictor is crucial, and its 6.256855 Meameancdecreasedni value indicates that it has a significant effect. It demonstrates how important it is for the particular product being advertised to have an impact on consumers’ download decisions and behavior. Additionally, the Mean Decrease Gini values of 4.195648 and 5.275011, respectively, for the predictors “os\_version” and “device\_type,” demonstrate their substantial importance. It draws attention to how important operating system versions and device specs are in determining app adoption patterns and user experiences. It’s interesting to note that the relative weights assigned to these predictors highlight the complex dynamics of user engagement, where a variety of device characteristics interact to affect the download speed of applications. When compared to all other predictors, the “prediction” characteristic has a startling Mean Decrease Gini value of 332.976090, making it much more meaningful. It highlights how important the target variable—a proxy for app download data—is to the model’s predictive power. The “prediction” part of the model mostly finds patterns or traits that represent app download behavior.

# Importance plot   
importance(classifier\_RF)

## MeanDecreaseGini  
## ip 7.284254  
## app\_id 6.256855  
## device\_type 5.275011  
## os\_version 4.195649  
## chanel\_id 6.610449  
## prediction 332.976090

# Variable importance plot   
varImpPlot(classifier\_RF)

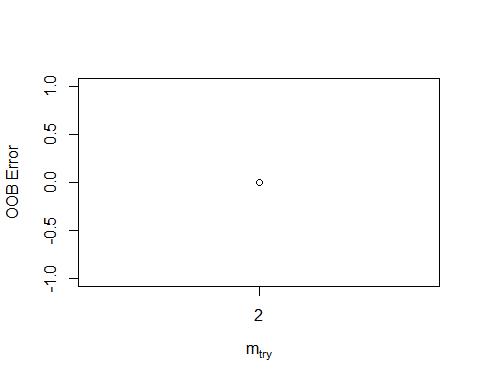


# Fine Tunning the model

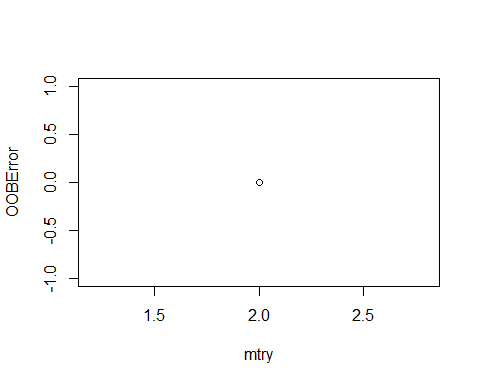
We used the ‘tuneRF’ to fine-tune our random forest model after determining which factors are most crucial. The’mtry’ parameter, which controls the number of variables the model takes into account at each decision point, was the main focus of adjustment. The outcomes are very outstanding! The model’s out-of-bag (OOB) error rate decreased to zero, indicating that it can now accurately predict results even on newly-generated data. The “finetune” object displays the optimal parameters for our improved model: a’mtry’ value of two and an OOB error rate of 0. This illustrates how meticulous model tweaking increases the model’s accuracy and increases its dependability for future forecasts.

# Tune the random forest model with adjusted improve parameter  
finetune <- tuneRF(  
 x = train\_data[-6],  
 y = train\_data$prediction,  
 stepFactor = 1,trace = TRUE)

## mtry = 2 OOB error = 0%   
## Searching left ...  
## Searching right ...



plot(finetune)



finetune

## mtry OOBError  
## 2.OOB 2 0