Prediction Model usingRandom Forest

Harry Depina

```
Load the necessary libraries required for this project.
The randomForest package is used to build and train the random forest model,
which is an ensemble learning method known for its robust predictive
capabilities.
The caret package is utilized for splitting the dataset into training and
testing sets,
providing tools for data partitioning, preprocessing, and model tuning.
Additionally, the pROC package is employed to evaluate the model's
performance


install.packages("randomForest")
install.packages("caret")
install.packages("pROC")

library(randomForest)
library(caret)
library(pROC)
```

```
## randomForest 4.7-1.2

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

```
library(caret)
```

```
## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

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

## Loading required package: lattice
```

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
# Load the dataset. This dataset contains information about user interactions
with ads for a smartphone app.
# The goal is to predict if a user will download the app after clicking the
ad.
data <- read.csv("data.csv")
```

```r
# Examine the structure of the dataset. This helps us understand the types and formats of the variables we're working with.
# The data has 8 columns: IP address, app ID, device type, OS version, channel ID, click timestamp, download timestamp, and the target variable 'downloaded'.


str(data)
```

```
## 'data.frame':    99999 obs. of  8 variables:
##  $ X87540       : int  105560 101424 94584 68413 93663 17059 121505 192967 143636 73839 ...
##  $ X12          : int  25 12 13 12 3 1 9 2 3 3 ...
##  $ X1           : int  1 1 1 1 1 1 1 2 1 1 ...
##  $ X13          : int  17 19 13 1 17 17 25 22 19 22 ...
##  $ X497         : int  259 212 477 178 115 135 442 364 135 489 ...
## $ X11.7.17.9.30: chr "11/7/17 13:40" "11/7/17 18:05" "11/7/17 4:58" "11/9/17 9:00" ...
##  $ X            : chr  "" "" "" "" ...
##  $ X0           : int  0 0 0 0 0 0 0 0 0 0 ...
```

```r
head(data)
```

```
##   X87540 X12 X1 X13 X497 X11.7.17.9.30 X X0
## 1 105560  25  1  17  259 11/7/17 13:40    0
## 2 101424  12  1  19  212 11/7/17 18:05    0
## 3  94584  13  1  13  477  11/7/17 4:58    0
## 4  68413  12  1   1  178  11/9/17 9:00    0
## 5  93663   3  1  17  115  11/9/17 1:22    0
## 6  17059   1  1  17  135  11/9/17 1:17    0
```

```r
summary(data)
```

```
##      X87540            X12               X1                X13        
##  Min.   :     9   Min.   :  1.00   Min.   :   0.00   Min.   :  0.00  
##  1st Qu.: 40552   1st Qu.:  3.00   1st Qu.:   1.00   1st Qu.: 13.00  
##  Median : 79827   Median : 12.00   Median :   1.00   Median : 18.00  
##  Mean   : 91256   Mean   : 12.05   Mean   :  21.77   Mean   : 22.82  
##  3rd Qu.:118252   3rd Qu.: 15.00   3rd Qu.:   1.00   3rd Qu.: 19.00  
##  Max.   :364757   Max.   :551.00   Max.   :3867.00   Max.   :866.00  
##       X497       X11.7.17.9.30           X                   X0         
##  Min.   :  3.0   Length:99999       Length:99999       Min.   :0.00000  
##  1st Qu.:145.0   Class :character   Class :character   1st Qu.:0.00000  
##  Median :258.0   Mode  :character   Mode  :character   Median :0.00000  
##  Mean   :268.8                                         Mean   :0.00227  
##  3rd Qu.:379.0                                         3rd Qu.:0.00000  
##  Max.   :498.0                                         Max.   :1.00000  
```

```r
# Rename columns for easier reference throughout the analysis.
# This makes the code more readable and easier to interpret.
```

```r
colnames(data) <- c("ip_address", "app_id", "device_type", "os_version",
"channel_id", "click_timestamp",              "download_timestamp",
"downloaded")

# Convert the target variable 'downloaded' into a factor. This is necessary
because we are performing classification.
data$downloaded <- as.factor(data$downloaded)

# Verify the changes made to the data structure to ensure everything is set
up correctly.
str(data)

## 'data.frame':    99999 obs. of  8 variables:
##  $ ip_address        : int  105560 101424 94584 68413 93663 17059 121505
192967 143636 73839 ...
##  $ app_id            : int  25 12 13 12 3 1 9 2 3 3 ...
##  $ device_type       : int  1 1 1 1 1 1 1 2 1 1 ...
##  $ os_version        : int  17 19 13 1 17 17 25 22 19 22 ...
##  $ channel_id        : int  259 212 477 178 115 135 442 364 135 489 ...
##  $ click_timestamp   : chr "11/7/17 13:40" "11/7/17 18:05" "11/7/17 4:58"
"11/9/17 9:00" ...
##  $ download_timestamp: chr  "" "" "" "" ...
##  $ downloaded        : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...

# Set a seed for reproducibility of results. This ensures that the random
split we perform next will be the same each time we run the code.
set.seed(999)

# Split the dataset into training (70%) and testing (30%) sets using caret's
createDataPartition function.
# This prepares the data for building and evaluating the model.
trainIndex <- createDataPartition(data$downloaded, p = 0.7, list = FALSE)
train_data <- data[trainIndex, ]
test_data <- data[-trainIndex, ]

# Train the Random Forest model. We are using ip_address, app_id,
device_type, os_version, and channel_id as predictors to forecast if the app
will be downloaded.
# The model is set to measure variable importance.
rf_model <- randomForest(downloaded ~ ip_address + app_id + device_type +
os_version + channel_id,
                         data = train_data, importance = TRUE)

# Print the model summary. This includes details like the number of trees,
the variables tried at each                         split, and the Out-Of-Bag
(OOB) error estimate.
# The OOB error rate provides an estimate of the model's error on unseen
data.
print(rf_model)
```

```
##
## Call:
## randomForest(formula = downloaded ~ ip_address + app_id + device_type +
os_version + channel_id, data = train_data, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of error rate: 0.2%
## Confusion matrix:
##       0  1  class.error
## 0 69823 18 0.0002577283
## 1   122 37 0.7672955975
```

```
# Predict on the test data using the trained Random Forest model. This step
gives us the predicted download statuses for the test data.
predictions <- predict(rf_model, test_data)

# Display the first few predictions to get a sense of the model's output.
head(predictions)
```

```
##  2  9 15 22 23 30
##  0  0  0  0  0  0
## Levels: 0 1
```

```
# Evaluate the model's performance using a confusion matrix. This matrix
compares the predicted values to the actual values in the test data.
# Metrics like accuracy, sensitivity, and specificity are calculated to
assess how well the model performed.
confusion_matrix <- confusionMatrix(predictions, test_data$downloaded)
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 29927    54
##          1     4    14
##
##                Accuracy : 0.9981
##                  95% CI : (0.9975, 0.9985)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 0.1228
##
##                   Kappa : 0.3249
##
##  Mcnemar's Test P-Value : 1.243e-10
##
##             Sensitivity : 0.9999
##             Specificity : 0.2059
##          Pos Pred Value : 0.9982
```

```
##          Neg Pred Value : 0.7778
##              Prevalence : 0.9977
##          Detection Rate : 0.9976
##    Detection Prevalence : 0.9994
##       Balanced Accuracy : 0.6029
##
##        'Positive' Class : 0
##
```

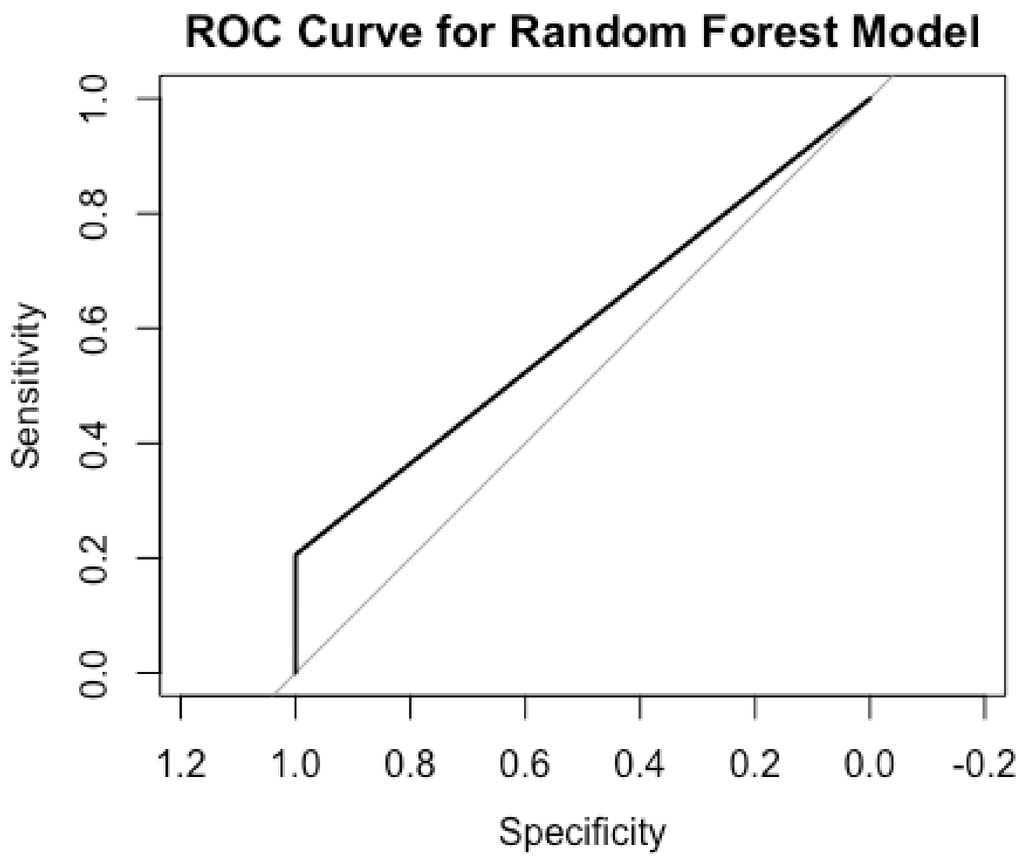
```r
# The model achieves an accuracy of around 99.81%. However, the sensitivity and specificity metrics reveal an imbalance,
# as the model is better at identifying non-downloads (class "0") than downloads (class "1").
# This imbalance is common in classification problems with skewed class distributions.

# Plot the ROC curve to visualize the trade-off between the true positive rate and false positive rate.
# A higher Area Under the Curve (AUC) value indicates better model performance.
roc_curve <- roc(test_data$downloaded, as.numeric(predictions))
```

```
## Setting levels: control = 0, case = 1
```

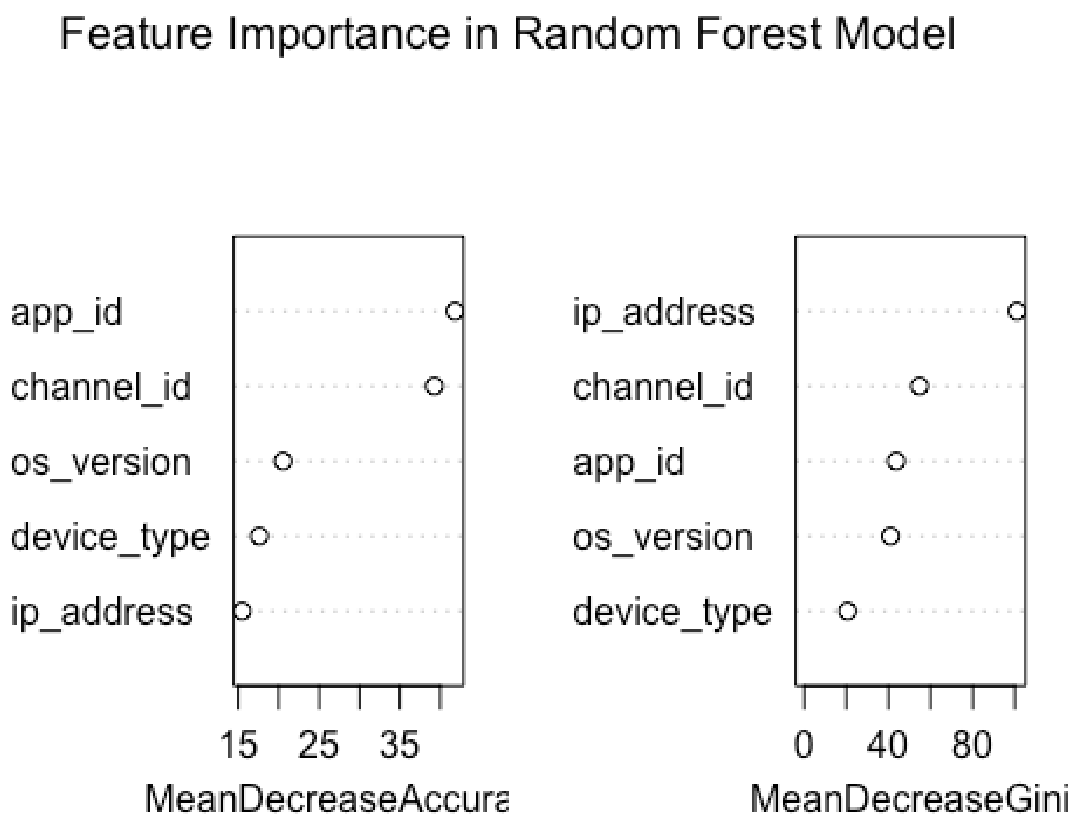
```
## Setting direction: controls < cases
```

```r
plot(roc_curve, main = "ROC Curve for Random Forest Model")
```




```r
# The ROC curve shows that the model has a relatively good ability to differentiate between downloadsand non-downloads,
# as it lies above the 45-degree line, indicating predictive power.

# Plot feature importance to identify which variables had the most impact on the predictions.
# This helps us understand the driving factors behind user app downloads.
varImpPlot(rf_model, main = "Feature Importance in Random Forest Model")
```




```r
# Extract and display the feature importance values from the model.
# The MeanDecreaseAccuracy metric shows the importance of each variable in
maintaining the model's accuracy.
# The MeanDecreaseGini metric indicates each variable's role in maintaining
node purity across the trees in the forest.
importance_values <- importance(rf_model)
print(importance_values)

##                       0        1 MeanDecreaseAccuracy MeanDecreaseGini
## ip_address  -6.585658 44.14471             15.46904        101.12717
## app_id      36.632892 63.17607             41.82380         43.70156
## device_type 15.713511 12.80190             17.59922         20.66655
## os_version  16.456376 24.58729             20.54581         40.98300
## channel_id  35.964003 32.57665             39.25806         54.81954

# Key findings from feature importance analysis:
# The 'app_id' and 'channel_id' variables are among the top predictors,
suggesting that certain apps and ad channels are more effective in driving
downloads.
# This insight can help marketers optimize ad targeting based on the
performance of specific channels and app IDs.

# Conclusion
```

```
# In this project, we successfully developed a Random Forest model to predict
whether a user will download an                      app after clicking on an
ad.
# The model achieved high accuracy but displayed a bias towards non-download
predictions, as seen from the specificity metric.
# The feature importance analysis indicates that app_id and channel_id are
key factors in predicting downloads,
# offering actionable insights for improving ad targeting strategies.

# This concludes the analysis. The model's predictions, accuracy metrics, ROC
curve, and feature importance collectively provide a comprehensive view
# of the factors influencing app downloads.
```