Own_project

José Antonio Cardoso

2024-11-20

Introduction

To complete the Professional Data Science training, the student was asked to develop a project of his own (authored). Some rules were established, for example, that the purpose of the project object (algorithm) was freely chosen, as well as the data set to be used in this development. Therefore, this report presents the aforementioned algorithm, as well as describing its objective and functionality.

Overview

The project object (algorithm) uses classification techniques and for this, it uses Random Forest, a machine learning algorithm that uses decision trees to make predictions. It also uses a GUIDE data set made available by Microsoft, on the Kaggle platform, which is a global community of data scientists. (/kaggle/input/microsoft-security-incident-prediction/).

Executive Summary

The project objective (algorithm) is to classify cybersecurity incidents by analyzing the responses to calls made by the SOC (Security Operations Center) team, as described in the aforementioned data set.

Methods

Several R language packages are required to enable the development of the algorithm. The dataset is originally available on the Kaggle platform, but a copy of it was downloaded and made available together with the files for this project.

```
#################################################
# START PREPARING THE TRAINING AND TEST DATA SETS
#################################################

# Read the training data set
GUIDE_train <- read.csv("GUIDE_Train.csv", header = TRUE)
# Read the test data set
GUIDE_test <- read.csv("GUIDE_Test.csv", header = TRUE)
```

```r
# Convert the Id field to character from the training dataset
GUIDE_train$Id <-as.character(GUIDE_train$Id)
# Convert the Id field to character from the test data set
GUIDE_test$Id <-as.character(GUIDE_test$Id)

# As you can see in this code, the data set is already divided.
# However, to demonstrate the knowledge about the logic and necessity of this
division
# I present here the code related to this process, where for educational
purposes I consider the
# complete GUDE file, which I fictitiously called complet_GUIDE.csv
# Just for information...
# Note: The division between Training and Testing is 70% and 30% respectively
# Dividing the data into training and validation
# Defining the seed for generating random numbers
#set.seed(0)
#train_index <- createDataPartition(complet_GUIDE$IncidentGrade, p = 0.7,
list = FALSE)
#GUIDE_train <- complet_GUIDE[train_index,]
#GUIDE_test <- complet_GUIDE[-train_index,]
```

Analysis

Originally the GUIDE dataset (Training and Testing) contains over 13 million pieces of evidence across 33 entity types, covering 1.6 million alerts and 1 million annotated incidents, distributed across 45 columns with information such as: DeviceId (Unique identifier for the device), IpAddress (Involved IP address), Url (Involved URL) AccountUpn (Email account identifier) etc. However, in order to meet the objective of classifying cybersecurity incidents by analyzing the responses to the calls made by the SOC (Security Operations Center) team, as described in the "Executive Summary", we decided to use the identification columns ("Id" and "IncidentId") as a matter of good practice, and we used the columns ActionGrouped (SOC alert remediation action (high level)), ActionGranular (SOC alert remediation action (fine grain)) and LastVerdict (Final verdict of the threat analysis) as predictor variables and IncidentGrade (SOC grade assigned to the incident) is the response variable.

```r
####################################
# PREPARING DATASETS (TRAINING ONLY)
####################################

# Make a copy of the training dataset
df_data_train <- GUIDE_train

# Select only the columns that are important for the evaluation
cols_df <-
```

```r
c("Id","IncidentId","ActionGrouped","ActionGranular","LastVerdict","IncidentG
rade")

# Apply the selection on the Training dataset
df_data_train <- df_data_train[, cols_df]

# Filters only the rows that contain SOC responses, thus eliminating any
column that contains "Not Available" values
df_data_train <- df_data_train %>%
   filter(ActionGrouped != "" & ActionGranular != "" & LastVerdict != "" &
IncidentGrade != "")

# Presents some quantitative information about the dataset.
cat("Number of records in the Training dataset","\n")

## Number of records in the Training dataset

cat("Show the first few rows of the dataset","\n")

## Show the first few rows of the dataset

head(df_data_train,5)

##             Id IncidentId ActionGrouped  ActionGranular LastVerdict
## 1 936302872206      56426 IsolateDevice isolateresponse  Suspicious
## 2 627065227429     138497 IsolateDevice isolateresponse  Suspicious
## 3 807453856769       8065 IsolateDevice isolateresponse  Suspicious
## 4 970662611054     327814 IsolateDevice  quarantinefile  Suspicious
## 5 652835031931       5371 IsolateDevice isolateresponse  Suspicious
##    IncidentGrade
## 1   TruePositive
## 2   TruePositive
## 3   TruePositive
## 4   TruePositive
## 5 BenignPositive

cat("\n","Show unique values from columns")

##
##  Show unique values from columns

# Apply the function to each column and store the results in a list
newcols_df <-
c("ActionGrouped","ActionGranular","LastVerdict","IncidentGrade")
list_result <- map(newcols_df, ~df_data_train %>%
                  pull(.x) %>%
                  unique())
names(list_result) <- newcols_df
list_result

## $ActionGrouped
## [1] "IsolateDevice"  "ContainAccount"
```

```
##
## $ActionGranular
##  [1] "isolateresponse"
##  [2] "quarantinefile"
##  [3] "change user password."
##  [4] "disable account."
##  [5] "update stsrefreshtokenvalidfrom timestamp."
##  [6] "forcepasswordresetremediation"
##  [7] "account password changed"
##  [8] "disableuser"
##  [9] "account disabled"
## [10] "reset user password."
## [11] "set force change user password."
## [12] "msecidentitiessuspenduser"
## [13] "msecidentitiesconfirmusercompromised"
##
## $LastVerdict
## [1] "Suspicious"     "NoThreatsFound" "Malicious"
##
## $IncidentGrade
## [1] "TruePositive"   "BenignPositive" "FalsePositive"
```

```
# In this line above, the code prints the list containing the unique values
of
# each column, as explained above,

cat("\n","Shows the structure of the dataset","\n")
```

```
##
##  Shows the structure of the dataset
```

```
str(df_data_train)
```

```
## 'data.frame':    1110 obs. of  6 variables:
##  $ Id             : chr  "936302872206" "627065227429" "807453856769"
"970662611054" ...
##  $ IncidentId     : int  56426 138497 8065 327814 5371 40983 119255 19426
766 5371 ...
##  $ ActionGrouped : chr  "IsolateDevice" "IsolateDevice" "IsolateDevice"
"IsolateDevice" ...
##  $ ActionGranular: chr  "isolateresponse" "isolateresponse"
"isolateresponse" "quarantinefile" ...
##  $ LastVerdict   : chr  "Suspicious" "Suspicious" "Suspicious"
"Suspicious" ...
##  $ IncidentGrade : chr  "TruePositive" "TruePositive" "TruePositive"
"TruePositive" ...
```

```
cat("\n","Summarizes the dataset")
```

```
##
##  Summarizes the dataset
```

```r
summary(df_data_train)

##       Id              IncidentId     ActionGrouped      ActionGranular
##  Length:1110        Min.   :    14   Length:1110        Length:1110
##  Class :character   1st Qu.:  5371   Class :character   Class :character
##  Mode  :character   Median : 31790   Mode  :character   Mode  :character
##                     Mean   : 72686
##                     3rd Qu.: 96466
##                     Max.   :566689
##  LastVerdict        IncidentGrade
##  Length:1110        Length:1110
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##

# Transform the string column into a factor
df_data_train$IncidentGrade <- as.factor(df_data_train$IncidentGrade)
df_data_train$ActionGrouped <- as.factor(df_data_train$ActionGrouped)
df_data_train$ActionGranular <- as.factor(df_data_train$ActionGranular)
df_data_train$LastVerdict <- as.factor(df_data_train$LastVerdict)
```

Execution

The model is executed using the machine learning algorithm that uses decision trees to make predictions, in this case Random Forest. It is extracted from the train function, which trains the model using Random Forest in an unencapsulated way. The trained_model object generated by the train function has the various methods that we use to demonstrate the execution of the model, including graphs of class distribution and importance of variables. The test is performed using the trained_model object, generating the appropriate predictions.

```r
##############################################
# PERFORM TRAINING AND EVALUATION OF THE MODEL
##############################################

# Set the seed for random number generation
set.seed(0)

# Set the control parameters
ctrl <- trainControl(method = "cv", number = 10)

# Define the hyperparameter grid
hyper_grid <- expand.grid(mtry = c(3, 4, 5))

# Train the model
trained_model <- train(IncidentGrade ~ ActionGrouped + ActionGranular +
```

```r
LastVerdict,
                       data = df_data_train,
                       method = "rf",
                       tuneGrid = hyper_grid,
                       trControl = ctrl)

# Print the trained model
print(trained_model)
```

```
## Random Forest
##
## 1110 samples
##    3 predictor
##    3 classes: 'BenignPositive', 'FalsePositive', 'TruePositive'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 999, 998, 998, 1000, 999, 999, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   3     0.6874503  0.3256632
##   4     0.6856485  0.3217928
##   5     0.6883756  0.3287985
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 5.
```

```r
# Presents an overview of the different hyperparameter combinations tested
and their respective performance metrics.
trained_model$results
```

```
##   mtry  Accuracy     Kappa AccuracySD    KappaSD
## 1    3 0.6874503 0.3256632 0.02626770 0.05512794
## 2    4 0.6856485 0.3217928 0.02640943 0.05542052
## 3    5 0.6883756 0.3287985 0.02770996 0.05931271
```

```r
# Show cross-validation results.
trained_model$resample
```

```
##     Accuracy     Kappa Resample
## 1  0.6936937 0.3368477   Fold01
## 2  0.6339286 0.2283650   Fold02
## 3  0.6936937 0.3177874   Fold05
## 4  0.6909091 0.3352293   Fold04
## 5  0.6964286 0.3601075   Fold03
## 6  0.7027027 0.3444882   Fold06
## 7  0.7027027 0.3697522   Fold09
## 8  0.6756757 0.2806481   Fold08
## 9  0.6576577 0.2722567   Fold07
## 10 0.7363636 0.4425026   Fold10
```

```r
# Shows the final selected model after hyperparameter sweeping and cross-validation.
trained_model$finalModel
```

```
## 
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry) 
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 5
## 
##         OOB estimate of  error rate: 31.62%
## Confusion matrix:
##                BenignPositive FalsePositive TruePositive class.error
## BenignPositive            165             0          183   0.5258621
## FalsePositive              32             0           54   1.0000000
## TruePositive               82             0          594   0.1213018
```

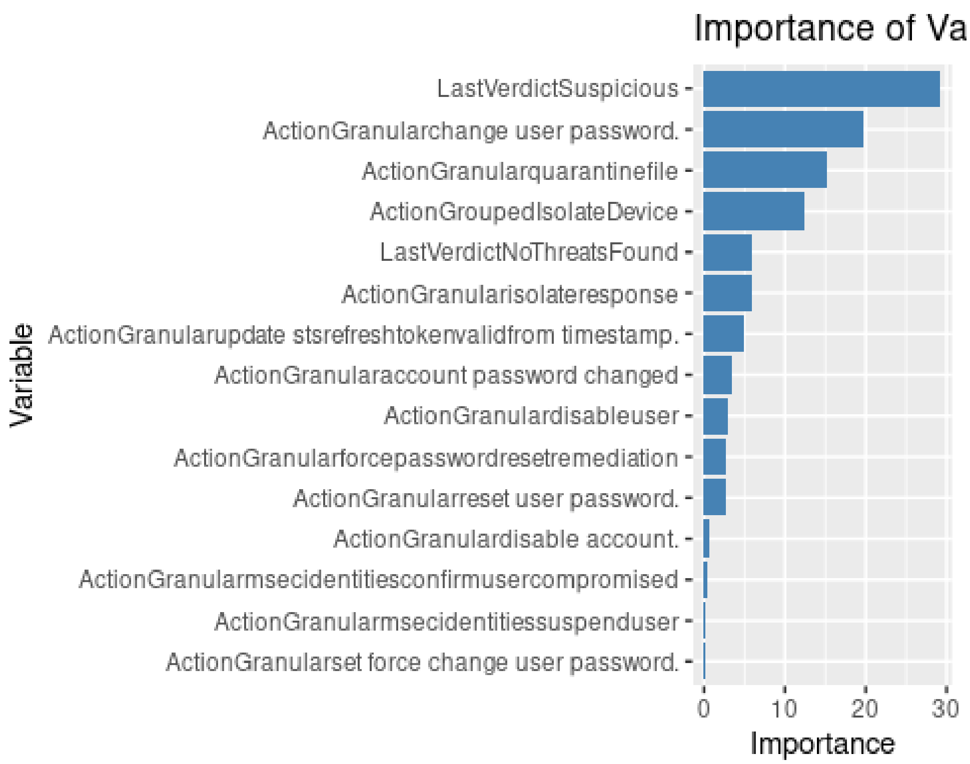
```r
# Shows the relative importance of each predictor variable in the final model
trained_model$finalModel$importance
```

```
##                                                           MeanDecreaseGini
## ActionGroupedIsolateDevice                                      12.5354011
## ActionGranularaccount password changed                           3.5399873
## ActionGranularchange user password.                             19.6301176
## ActionGranulardisable account.                                   0.7235116
## ActionGranulardisableuser                                        2.9308146
## ActionGranularforcepasswordresetremediation                      2.7904143
## ActionGranularisolateresponse                                    5.8971866
## ActionGranularmsecidentitiesconfirmusercompromised               0.4269763
## ActionGranularmsecidentitiessuspenduser                          0.1364855
## ActionGranularquarantinefile                                    15.2801257
## ActionGranularreset user password.                               2.5494853
## ActionGranularset force change user password.                    0.1045615
## ActionGranularupdate stsrefreshtokenvalidfrom timestamp.         4.8546657
## LastVerdictNoThreatsFound                                        5.9898430
## LastVerdictSuspicious                                           29.1997856
```

```r
# This graph shows the distribution of classes in the data set, helping to understand if there is an imbalance.
# Plots the distribution of classes
ggplot(df_data_train, aes(x = IncidentGrade)) +
  geom_bar(fill = "steelblue") +
  labs(title = "Distribution of Classes in the Training Set", x = "Class", y = "Count")
```


```
# This graph shows the relative importance of each predictor variable.
# Plots the importance of variables
importance <- varImp(trained_model, scale = FALSE)
ggplot(importance, aes(x = reorder(Overall, Overall), y = Overall)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(title = "Importance of Variables", x = "Variable", y = "Importance")

## Coordinate system already present. Adding new coordinate system, which
will
## replace the existing one.
```




```
# Make predictions on the TRAINING dataset to evaluate the model's
performance
train_predictions <- predict(trained_model, newdata = df_data_train)

# Converts categorical data into factors for classification
actual_class <- df_data_train$IncidentGrade
predicted_class <- train_predictions

# Calculates accuracy, as an assessment of model performance
accuracy_train <- accuracy(actual_class, predicted_class)
cat("Accuracy:", accuracy_train, "\n")

## Accuracy: 0.6963964

# Calculate the F1-score, as an evaluation of the model's performance
f1_train <- f1(actual_class, predicted_class)
cat("F1-score:", f1_train, "\n")

## F1-score: 0.8

#####################################
# TEST AND EVALUATE THE TRAINED MODEL
#####################################

# Make a copy of the test dataset
df_data_test <- GUIDE_test
```

```r
# Apply the selection to the Test dataset
df_data_test <- df_data_test[, cols_df]

# Filters only the rows that contain SOC responses, thus eliminating any
column that contains "Not Available" values
df_data_test <- df_data_test %>%
   filter(ActionGrouped != "" & ActionGranular != "" & LastVerdict != "" &
IncidentGrade != "")

# Transform the string column into a factor
df_data_test$IncidentGrade <- as.factor(df_data_test$IncidentGrade)
df_data_test$ActionGrouped <- as.factor(df_data_test$ActionGrouped)
df_data_test$ActionGranular <- as.factor(df_data_test$ActionGranular)
df_data_test$LastVerdict <- as.factor(df_data_test$LastVerdict)

# Make predictions on the TEST dataset to evaluate the model's performance
test_predictions <- predict(trained_model, newdata = df_data_test)

# Converts categorical data into factors for classification
actual_class <- df_data_test$IncidentGrade
predicted_class <- test_predictions

# Calculates accuracy, as an assessment of model performance
accuracy_test <- accuracy(actual_class, predicted_class)
cat("Accuracy:", accuracy_test, "\n")

## Accuracy: 0.7058824

# Calculate the F1-score, as an evaluation of the model's performance
f1_test <- f1(actual_class, predicted_class)
cat("F1-score:", f1_test, "\n")

## F1-score: 0.8
```

Results

Although the data set is considerably large, which is very good for achieving greater reliability in the model's execution, I tried to eliminate as many rows and columns of incomplete data as possible, thus generating a smaller amount of data, but with all rows and columns containing reliable information. I tried to use accuracy to evaluate the model's performance, as well as the F1 Score to help balance the importance of false positives and false negatives.

```r
####################################
# MODEL OUTPUT AND ITS PERFORMANCE #
####################################

# IMPRIME A Calculates accuracy, as an assessment of model performance
cat("Accuracy:", accuracy_train, "\n")
```

```
## Accuracy: 0.6963964

# Calculate the F1-score, as an evaluation of the model's performance
cat("F1-score:", f1_train, "\n")

## F1-score: 0.8

# Calculates accuracy, as an assessment of model performance
cat("Accuracy:", accuracy_test, "\n")

## Accuracy: 0.7058824

# Calculate the F1-score, as an evaluation of the model's performance
cat("F1-score:", f1_test, "\n")

## F1-score: 0.8
```

Conclusion

Reaching an accuracy of 0.7058824 and an F1-Score of 0.8 can be considered a good result in some aspects, but I understand that there is a lot of room for improvement. The exclusion of incomplete data rows and columns, as mentioned above, contributed to obtaining these performance numbers, although we were left with a more limited amount of data. However, in order to continue the work, I intend to reevaluate the data set, looking for more subsidies for the model to make its classifications.