AI-based web application firewall development

Aleksandrs Rudenoks

26/02/2021

Load Libraries

Libraries required for this task are loaded.

```
library("readr")
library("caret")
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library("tm")
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library("e1071")
library("SnowballC")
```

Set working directory

A path to the working directory is provided.

```
setwd("C:/Users/PC/Desktop/AI Firewall")
```

Data Exploration

The first step of building an intrusion detection classifier is to explore the raw data. Given data sets contain benign and malicious queries that are loaded using read_delim function. Summary of loaded data sets also provided below.

```
# Load data sets, reading each line as separate entity
benignqueries <- read_delim("benignqueries.txt", "\n", escape_double = FALSE, col_names = FALSE)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_character()
## )
```

```
maliciousqueries <- read_delim("maliciousqueries.txt", "\n", escape_double = FALSE, col_names = FALSE)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_character()
## )
```

```r
summary(benignqueries)
```

```
##        X1
##  Length:1294599
##  Class :character
##  Mode  :character
```

```r
summary(maliciousqueries)
```

```
##        X1
##  Length:47047
##  Class :character
##  Mode  :character
```

Data sets contain a list of benign and malicious requests in raw form.

```r
head(benignqueries)
```

```
## # A tibble: 6 x 1
##   X1
##   <chr>
## 1 /lcd soundsystem - lcd soundsystem/
## 2 /weather_sites/
## 3 /156211/
## 4 /tactics/
## 5 /nextarrow/
## 6 /101996/
```

```r
head(maliciousqueries)
```

```
## # A tibble: 6 x 1
##   X1
##   <chr>
## 1 "/<script>document.cookie=\"testrluj=1420;\"</script>"
## 2 "/javascript/.passwd.jpg"
## 3 "/opensiteadmin/scripts/classes/databasemanager.php?path=http://192.168.202.1~
## 4 "/examples/jsp/jsp2/el/search=<script>alert('xss')</script>"
## 5 "/javascript/signer.exe"
## 6 "/help.php?q=\"&del\\x0bq26193259&rem\\x0b"
```

It is important to double-check that loaded data sets are complete, meaning they have no missing values. Corresponding feature labels are attached to data sets.

```r
# Remove possible missing records
benignqueries <- benignqueries[complete.cases(benignqueries), ]
maliciousqueries <- maliciousqueries[complete.cases(maliciousqueries), ]

# Attach data labels
benignqueries$label <- "benign"
maliciousqueries$label <- "malicious"
benignqueries$label <- as.factor(benignqueries$label)
maliciousqueries$label <- as.factor(maliciousqueries$label)
```

While either of the data sets is relatively small-sized, text processing can be a computationally heavy process. Due to the memory limitations of my computer, it is acceptable to use a smaller sample of data instead of using the entire data set. Distribution of benign to malicious data is 3 to 1, which is an acceptable spread due to the fact that majority of the real-life queries would be benign and the primary goal of the model is to prevent malicious queries from being processed.

```
benignqueries <- benignqueries[1:30000,]
maliciousqueries <- maliciousqueries[1:10000,]
```

A single data set is created from both benign and malicious queries. This data set will be later used to create training and testing sets for the classification model.

```
mixedSet <- rbind(benignqueries, maliciousqueries)
summary(mixedSet)
```

```
##       X1                  label
##  Length:40000       benign   :30000
##  Class :character   malicious:10000
##  Mode  :character
```

Data preparation

Next step is data preparation. In order to analyse and classify given data, it needs to be standardised by converting the raw data in the format that the classification model can understand. First, all entries from the mixed data set are loaded into the corpus, which is a collection of text documents.

```
textCorpus <- VCorpus(VectorSource(mixedSet$X1))
print(textCorpus)
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 40000
```

It is possible to review the first 5 entries in the text corpus.

```
lapply(textCorpus[1:5], as.character)
```

```
## $`1`
## [1] "/lcd soundsystem - lcd soundsystem/"
##
## $`2`
## [1] "/weather_sites/"
##
## $`3`
## [1] "/156211/"
##
## $`4`
## [1] "/tactics/"
##
## $`5`
## [1] "/nextarrow/"
```

The corpus contains the raw text of 40000 queries. In order to analyse those queries, they need to be processed as separate words. Before this is done, a couple of standardising functions need to be applied to the raw data. Following functions return a lowercase version of text strings and remove all numbers from the corpus.

```
textCorpusClean <- tm_map(textCorpus, content_transformer(tolower))
textCorpusClean <- tm_map(textCorpusClean, content_transformer(removeNumbers))
```

A function was created that transforms all punctuation marks into spaces. A standard removePunctuation function does not provide the same results since it combines 2 words after punctuation removal. For instance, query *home/user* would turn into *homeuser*. In our case, the majority of the separate words in queries are separated with dashes or other punctuation marks, hence require to be populated with empty spaces in between them.

```r
# Function to replace punctuation with spaces
replace_punctuation <- function(x)
  unlist(gsub('[[:punct:]]+',' ',x))

# Apply function
textCorpusClean <- tm_map(textCorpusClean, content_transformer(replace_punctuation))
```

Now the text corpus contains queries that are separated into words and stripped from all numbers and punctuation marks. Additionally, the words in queries can be reduced to their root forms in a process called stemming. For instance, word *learning* would turn into *learn*. Such practice allows learning algorithms to treat related terms as a single concept. The last step of data standardisation is to remove multiple white spaces between the words where punctuation marks were located previously.

```r
textCorpusClean <- tm_map(textCorpusClean, content_transformer(stemDocument))
textCorpusClean <- tm_map(textCorpusClean, content_transformer(stripWhitespace))
```

Modified data can be reviewed using the following command.

```r
lapply(textCorpusClean[1:5], as.character)
```

```
## $`1`
## [1] "lcd soundsystem lcd soundsystem"
##
## $`2`
## [1] "weather site"
##
## $`3`
## [1] ""
##
## $`4`
## [1] "tactic"
##
## $`5`
## [1] "nextarrow"
```

Clean and standardised data is ready to be split into separate words. The DocumentTermMatrix function will be used to convert the text corpus into a data structure in which rows indicate queries and columns indicate separate words. It is unlikely that all of the words will be useful for classification algorithm, therefore, a DTM can be reduced by the number of features. For this task, only words that are appearing more than 9 times in the DTM will be used.

```r
# Create Document Term Matrix(DTM)
textDTM <- DocumentTermMatrix(textCorpusClean)

freqWords <- findFreqTerms(textDTM,9)
textDTM <- textDTM[,freqWords]
```

At this point, text data is ready for analysis. Next step is the separation into training and testing data sets. For this task, 80% of data will be used for training and 20% for testing.

```r
# Create training and testing data sets
validation_index <- createDataPartition(mixedSet$label, p=0.80, list=FALSE)
```

```
trainSet <- mixedSet[validation_index,] # use 80% for training
```

```
## Warning: The `i` argument of ``[`()` can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
testSet <- mixedSet[-validation_index,] # use 20% for validation
```

We can verify consistent label distribution by displaying train and test set summaries. In both sets distribution of benign queries to malicious queries is 3 to 1.

```
summary(trainSet)
```

```
##       X1                   label
##  Length:32000       benign   :24000
##  Class :character   malicious: 8000
##  Mode  :character
```

```
summary(testSet)
```

```
##       X1                   label
##  Length:8000        benign   :6000
##  Class :character   malicious:2000
##  Mode  :character
```

Corresponding DTM entries should be obtained for training and testing data sets.

```
# Corresponding DTM entries
textDTMTrain <- textDTM[validation_index,]
textDTMTest <- textDTM[-validation_index,]
```

Data Convertation

The Naive Bayes classifier is typically trained on data with categorical features. Current DTMs contain numerical data that represents the number of times a word appears in a query. Using the following function, data will be converted into categorical representation, where "T" indicates word presence in the query, and "F" indicates its absence.

```
convert_counts <- function(x) {  x <- ifelse(x > 0, "T", "F")}
#Apply function
textDTMTrainNew <- apply(textDTMTrain, MARGIN = 2,convert_counts)
textDTMTestNew <- apply(textDTMTest, MARGIN = 2, convert_counts)
```

Model Training

Categorical data is now ready to be used for classification. The following command will create a model using the Naive Bayes classifier.

```
NBmodel <- naiveBayes(textDTMTrainNew, trainSet$label)
```

Model Performance Evaluation

To evaluate our model, it needs to be applied to the test data. The test data does not contain corresponding feature labels, meaning the model will have to predict those labels.

```
testPrediction <- predict(NBmodel, textDTMTestNew)
```

The accuracy of the model will be evaluated using confusion matrix.

```r
confusionMatrix(testPrediction, testSet$label, positive = "benign")
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  benign malicious
##   benign      5870       254
##   malicious    130      1746
##
##                Accuracy : 0.952
##                  95% CI : (0.9471, 0.9566)
##     No Information Rate : 0.75
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8693
##
##  Mcnemar's Test P-Value : 3.456e-10
##
##             Sensitivity : 0.9783
##             Specificity : 0.8730
##          Pos Pred Value : 0.9585
##          Neg Pred Value : 0.9307
##              Prevalence : 0.7500
##          Detection Rate : 0.7338
##    Detection Prevalence : 0.7655
##       Balanced Accuracy : 0.9257
##
##        'Positive' Class : benign
##
```

According to the confusion matrix, 95% of test data was classified as True Positive or True Negative. It should also be noted that the number of False Positive cases is larger than False Negatives, meaning that our model leans towards assigning test queries to benign class. While it is a preferable situation, the number of False Negatives is about 2% out of all benign queries. This means that a few benign queries may be classified as malicious, which can lead to disruption of service for the server. Higher accuracy may be achieved using other machine learning algorithms.