Logistic Regression Project

Ian Mbaya

2024-04-25

## Load the Neccessary Libraries

library('tidyverse')

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.4 ✔ readr 2.1.4  
## ✔ forcats 1.0.0 ✔ stringr 1.5.0  
## ✔ ggplot2 3.4.2 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.2 ✔ tidyr 1.3.0  
## ✔ purrr 1.0.1   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library('tidymodels')

## ── Attaching packages ────────────────────────────────────── tidymodels 1.1.0 ──  
## ✔ broom 1.0.5 ✔ rsample 1.1.1  
## ✔ dials 1.2.0 ✔ tune 1.1.1  
## ✔ infer 1.0.4 ✔ workflows 1.1.3  
## ✔ modeldata 1.1.0 ✔ workflowsets 1.0.1  
## ✔ parsnip 1.1.1 ✔ yardstick 1.2.0  
## ✔ recipes 1.0.6   
## ── Conflicts ───────────────────────────────────────── tidymodels\_conflicts() ──  
## ✖ scales::discard() masks purrr::discard()  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ recipes::fixed() masks stringr::fixed()  
## ✖ dplyr::lag() masks stats::lag()  
## ✖ yardstick::spec() masks readr::spec()  
## ✖ recipes::step() masks stats::step()  
## • Search for functions across packages at https://www.tidymodels.org/find/

library('caret')

## Loading required package: lattice  
##   
## Attaching package: 'caret'  
##   
## The following objects are masked from 'package:yardstick':  
##   
## precision, recall, sensitivity, specificity  
##   
## The following object is masked from 'package:purrr':  
##   
## lift

library('pROC')

## Type 'citation("pROC")' for a citation.  
##   
## Attaching package: 'pROC'  
##   
## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

library('lubridate')  
library('writexl')  
library('knitr')

# Loading the Data

## I am using data1 because of size constraints. The original file was too big for github

#attacks <- read.csv("data/CTU1.csv", sep = "|") Deleted File  
attacks <- readxl::read\_xlsx("data/data1.xlsx")

# Preliminary Data Analysis

## Take a Glimpse of the Data

dplyr::glimpse(attacks)

## Rows: 1,008,748  
## Columns: 8  
## $ label <chr> "Malicious", "Malicious", "Malicious", "Malicious", "Mal…  
## $ ts <dttm> 2018-05-09 15:30:31, 2018-05-09 15:30:31, 2018-05-09 15…  
## $ id.resp\_h <chr> "65.127.233.163", "63.150.16.171", "111.40.23.49", "131.…  
## $ id.resp\_p <dbl> 23, 23, 23, 23, 23, 23, 49560, 21288, 23, 8080, 8080, 23…  
## $ id.orig\_p <dbl> 51524, 56305, 41101, 60905, 44301, 50244, 34243, 34840, …  
## $ proto <chr> "tcp", "tcp", "tcp", "tcp", "tcp", "tcp", "tcp", "tcp", …  
## $ history <chr> "S", "S", "S", "S", "S", "S", "S", "S", "S", "S", "S", "…  
## $ orig\_ip\_bytes <dbl> 180, 60, 60, 180, 60, 60, 180, 60, 60, 60, 60, 180, 180,…

## Data Summary

summary(attacks)

## label ts id.resp\_h   
## Length:1008748 Min. :2018-05-09 15:30:31.01 Length:1008748   
## Class :character 1st Qu.:2018-05-10 17:57:40.75 Class :character   
## Mode :character Median :2018-05-11 20:40:00.00 Mode :character   
## Mean :2018-05-11 21:43:27.22   
## 3rd Qu.:2018-05-13 01:13:40.00   
## Max. :2018-05-14 07:24:43.02   
## id.resp\_p id.orig\_p proto history   
## Min. : 0 Min. : 3 Length:1008748 Length:1008748   
## 1st Qu.: 23 1st Qu.:43730 Class :character Class :character   
## Median : 8080 Median :43763 Mode :character Mode :character   
## Mean :16098 Mean :44437   
## 3rd Qu.:28180 3rd Qu.:48814   
## Max. :65535 Max. :65394   
## orig\_ip\_bytes   
## Min. : 0.00   
## 1st Qu.: 40.00   
## Median : 60.00   
## Mean : 81.15   
## 3rd Qu.: 60.00   
## Max. :2990.00

## Checking for Missing Values

skimr::skim(attacks)

Data summary

|  |  |
| --- | --- |
| Name | attacks |
| Number of rows | 1008748 |
| Number of columns | 8 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Column type frequency: |  |
| character | 4 |
| numeric | 3 |
| POSIXct | 1 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Group variables | None |

**Variable type: character**

| skim\_variable | n\_missing | complete\_rate | min | max | empty | n\_unique | whitespace |
| --- | --- | --- | --- | --- | --- | --- | --- |
| label | 0 | 1 | 6 | 9 | 0 | 2 | 0 |
| id.resp\_h | 0 | 1 | 7 | 15 | 0 | 597107 | 0 |
| proto | 0 | 1 | 3 | 4 | 0 | 3 | 0 |
| history | 0 | 1 | 1 | 10 | 0 | 126 | 0 |

**Variable type: numeric**

| skim\_variable | n\_missing | complete\_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id.resp\_p | 0 | 1 | 16097.71 | 19562.80 | 0 | 23 | 8080 | 28180.25 | 65535 | ▇▁▁▁▁ |
| id.orig\_p | 0 | 1 | 44436.84 | 9660.59 | 3 | 43730 | 43763 | 48814.00 | 65394 | ▁▁▂▇▂ |
| orig\_ip\_bytes | 0 | 1 | 81.15 | 94.73 | 0 | 40 | 60 | 60.00 | 2990 | ▇▁▁▁▁ |

**Variable type: POSIXct**

| skim\_variable | n\_missing | complete\_rate | min | max | median | n\_unique |
| --- | --- | --- | --- | --- | --- | --- |
| ts | 0 | 1 | 2018-05-09 15:30:31 | 2018-05-14 07:24:43 | 2018-05-11 20:40:00 | 946455 |

## Create a Data Dictionary

# Data Dictionary for Network Connection Data  
data\_dictionary <- tibble::tribble(  
 ~Field\_Name, ~Description, ~Type,  
 "ts", "The timestamp of the connection event.", "time",  
 "uid", "A unique identifier for the connection.", "string",  
 "id.orig\_h", "The source IP address.", "addr",  
 "id.orig\_p", "The source port.", "port",  
 "id.resp\_h", "The destination IP address.", "addr",  
 "id.resp\_p", "The destination port.", "port",  
 "proto", "The network protocol used (e.g., 'tcp').", "enum",  
 "service", "The service associated with the connection.", "string",  
 "duration", "The duration of the connection.", "interval",  
 "orig\_bytes", "The number of bytes sent from the source to the destination.", "count",  
 "resp\_bytes", "The number of bytes sent from the destination to the source.", "count",  
 "conn\_state", "The state of the connection.", "string",  
 "local\_orig", "Indicates whether the connection is considered local or not.", "bool",  
 "local\_resp", "Indicates whether the connection is considered local or not.", "bool",  
 "missed\_bytes", "The number of missed bytes in the connection.", "count",  
 "history", "A history of connection states.", "string",  
 "orig\_pkts", "The number of packets sent from the source to the destination.", "count",  
 "orig\_ip\_bytes", "The number of IP bytes sent from the source to the destination.", "count",  
 "resp\_pkts", "The number of packets sent from the destination to the source.", "count",  
 "resp\_ip\_bytes", "The number of IP bytes sent from the destination to the source.", "count",  
 "tunnel\_parents","Indicates if this connection is part of a tunnel.", "set[string]",  
 "label", "A label associated with the connection (e.g., 'Malicious' or 'Benign').", "string",  
 "detailed\_label","A more detailed description or label for the connection.", "string"  
)

From the summary and glimpse functions above it seems like we might have too many variables that may not be contributing to the model. While the skim functions shows that there are no missing values some of the variables have a dash has placeholder “-”. In the following steps we will be further exploring and visualizing the data to identify trends and patterns in the data that will determine the most important variables for prediction.

# Data Processing, Exploration Feature Engineering.

## Correlation Analysis

Running A Correlation Analysis to Identify the variables that have the highest corellation with network traffic label.

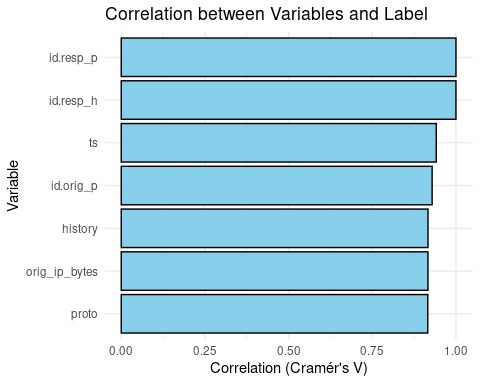
# Check the levels of the label variable  
levels(attacks$label)

## NULL

# Convert label to a factor if it's not already  
attacks$label <- as.factor(attacks$label)  
  
# Calculate correlation using Cramér's V for categorical variables  
cramer\_v <- function(x, y) {  
 confusion\_matrix <- table(x, y)  
 n <- sum(confusion\_matrix)  
 chi\_sq <- chisq.test(confusion\_matrix)$statistic  
 return(sqrt(chi\_sq / (n \* (min(nrow(confusion\_matrix), ncol(confusion\_matrix)) - 1))))  
}  
  
# Calculate correlation between label and each categorical variable  
correlation\_results <- lapply(attacks[, -which(names(attacks) == "label")], cramer\_v, y = attacks$label)

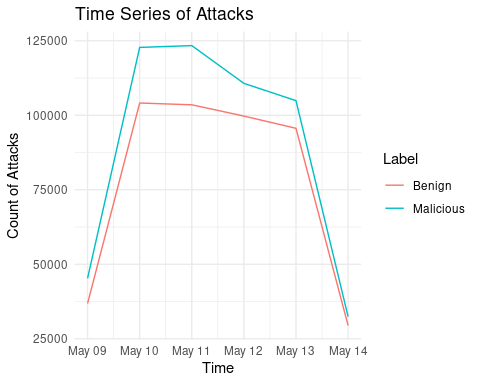
## Warning in chisq.test(confusion\_matrix): Chi-squared approximation may be  
## incorrect  
  
## Warning in chisq.test(confusion\_matrix): Chi-squared approximation may be  
## incorrect  
  
## Warning in chisq.test(confusion\_matrix): Chi-squared approximation may be  
## incorrect  
  
## Warning in chisq.test(confusion\_matrix): Chi-squared approximation may be  
## incorrect  
  
## Warning in chisq.test(confusion\_matrix): Chi-squared approximation may be  
## incorrect  
  
## Warning in chisq.test(confusion\_matrix): Chi-squared approximation may be  
## incorrect

# Convert the results to a data frame  
correlation\_df <- data.frame(  
 variable = names(correlation\_results),  
 correlation = unlist(correlation\_results)  
)  
  
# Visualize correlation  
ggplot(correlation\_df, aes(x = reorder(variable, correlation), y = correlation)) +  
 geom\_bar(stat = "identity", fill = "skyblue", color = "black") +  
 coord\_flip() +  
 labs(x = "Variable", y = "Correlation (Cramér's V)",  
 title = "Correlation between Variables and Label") +  
 theme\_minimal()

 In the first iteration there were 23 variables, I opted to remove all the variables that had a correlation of less than 0.75 and columns that had all values set to “-” the placeholder for missing. This alllowed me to save on time and space when running the models and also allowed me to upload the Project to GitHub without getting the error file is too large.

## Visualizing Pattern in Network Traffic based on Timestamps

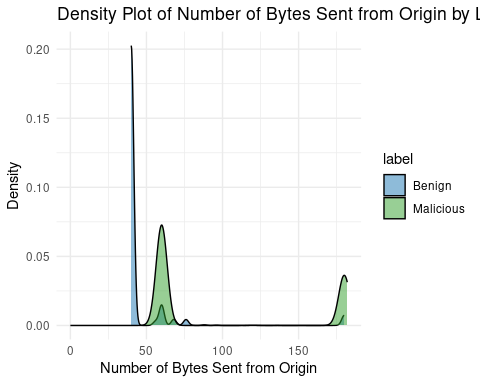
# Convert 'ts' to a DateTime object   
attacks$ts <- as.POSIXct(attacks$ts, format = "%Y-%m-%d %H:%M:%S")   
  
# Create a new dataframe with counts of attacks per time period (e.g., per day)  
attacks\_per\_period <- attacks %>%  
 group\_by(ts = floor\_date(ts, "day"), label) %>% # group by day and label;   
 summarise(count = n(), .groups = 'drop') # count the number of rows (attacks) per group  
  
# Create the time series plot  
ggplot(attacks\_per\_period, aes(x = ts, y = count, group = label, color = label)) +  
 geom\_line() +   
 labs(title = "Time Series of Attacks", x = "Time", y = "Count of Attacks", color = "Label") +  
 theme\_minimal() +  
 scale\_x\_datetime(date\_breaks = "1 day", date\_labels = "%b %d")

 The line charts shows that both Malicious and Benign attacks follow the same pattern. Although the data shows that there are more malicious requests in this particular data set.

## Visualizing the Number of Bytes Sent from Origin

# Calculate the 99th percentile to identify the 1% threshold  
percentile\_99 <- quantile(attacks$orig\_ip\_bytes, probs = 0.99)  
  
# Create a density plot with a more focused x-axis limit  
ggplot(attacks, aes(x = orig\_ip\_bytes, fill = label)) +  
 geom\_density(alpha = 0.5, trim = TRUE) + # Trim the density to limit the plot  
 labs(x = "Number of Bytes Sent from Origin",  
 y = "Density",  
 title = "Density Plot of Number of Bytes Sent from Origin by Label") +  
 scale\_fill\_manual(values = c("#1f78b4", "#33a02c")) + # Custom colors for the fill  
 theme\_minimal() +  
 xlim(c(0, max(180, percentile\_99))) # Set x-axis limits to highlight the 1% tail

## Warning: Removed 9585 rows containing non-finite values (`stat\_density()`).

 The density charts shows that request that send over 60 bytes from the origin Ip address tend to be mostly malicious. Most benign requests send less than 50 bytesfrom the origin IP.

## Exploring the Categorical Variables History, ID.Resp\_p, ID.Resp\_p, ID.Resp\_h, ID.Resp\_p, Proto (All have high correaltions with label)

# Determine the number of unique values for each variable  
num\_unique\_history <- attacks %>% select(history) %>% n\_distinct()  
num\_unique\_id\_resp\_p <- attacks %>% select(id.resp\_p) %>% n\_distinct()  
num\_unique\_id\_resp\_h <- attacks %>% select(id.resp\_h) %>% n\_distinct()  
num\_unique\_id\_orig\_h <- attacks %>% select(id.orig\_p) %>% n\_distinct()  
num\_unique\_proto <- attacks %>% select(proto) %>% n\_distinct()  
  
# Create a data frame to store the results  
unique\_values\_df <- data.frame(  
 variable = c("id.resp\_p", "id.resp\_h", "id.orig\_p", "proto", "history"),  
 unique\_values = c(num\_unique\_id\_resp\_p, num\_unique\_id\_resp\_h, num\_unique\_id\_orig\_h, num\_unique\_proto, num\_unique\_history)  
)  
  
# Use kable for presentation  
kable(unique\_values\_df, caption = "Number of Unique Values in Each Variable")

Number of Unique Values in Each Variable

| variable | unique\_values |
| --- | --- |
| id.resp\_p | 65426 |
| id.resp\_h | 597107 |
| id.orig\_p | 28243 |
| proto | 3 |
| history | 126 |

Of the FIVE categorical models picked from the first iteration (cor > 0.75 and non missing) only one (protocol) seem to be appropriate for linear regression model. The destination and origin (IP addresses and Ports), and network history have very large dimensionality so they can only be fit into a regression model after rigorous stemming and lemitization that I did not have the resources to implement. However I opted to use wordcloud to visualize how the different values of history appear in the dataset.

##Visualizing The Categorical Variables History with a WordCloud

# Load necessary libraries  
library(wordcloud)

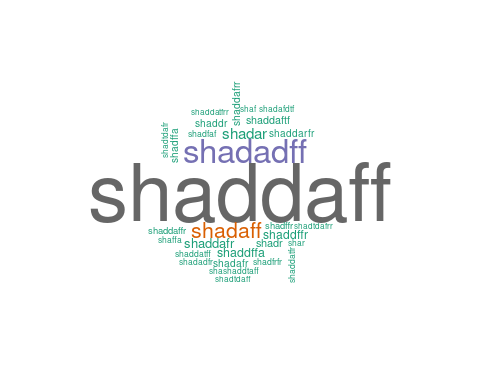
## Loading required package: RColorBrewer

# Filter out "unknown" values  
attacks\_filtered <- subset(attacks, history != "unknown")  
  
# Create separate datasets for malicious and benign  
malicious\_data <- subset(attacks\_filtered, label == "Malicious")  
benign\_data <- subset(attacks\_filtered, label == "Benign")  
  
# Create word cloud for malicious data  
wordcloud(malicious\_data$history, scale=c(5,0.5), min.freq = 10, random.order=FALSE, colors=brewer.pal(8, "Dark2"), main="Malicious")

## Loading required namespace: tm

## Warning in tm\_map.SimpleCorpus(corpus, tm::removePunctuation): transformation  
## drops documents

## Warning in tm\_map.SimpleCorpus(corpus, function(x) tm::removeWords(x,  
## tm::stopwords())): transformation drops documents



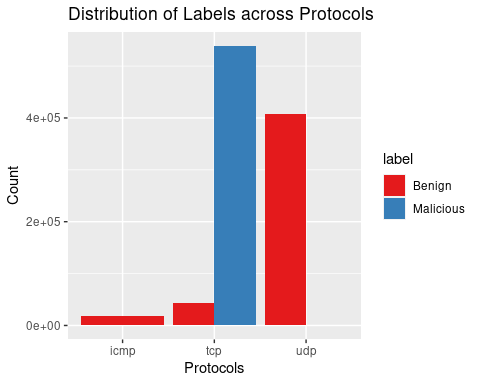
# Create word cloud for benign data  
wordcloud(benign\_data$history, scale=c(5,0.5), min.freq = 10, random.order=FALSE, colors=brewer.pal(8, "Dark2"), main="Benign")

## Warning in tm\_map.SimpleCorpus(corpus, tm::removePunctuation): transformation  
## drops documents  
  
## Warning in tm\_map.SimpleCorpus(corpus, tm::removePunctuation): transformation  
## drops documents

 The two worldclouds confirm my intial sentiments with the history “haddaaff” dominating benign request and all the other states in the malicious set.

##Visualizing The Protocols Values on a Barchart

# Ensure all varaibales are factors  
attacks$label <- as.factor(attacks$label)  
attacks$proto <- as.factor(attacks$proto)  
  
# Plot1 Protocols  
ggplot(data = attacks, aes(x = proto, fill = label)) +  
 geom\_bar(position = "dodge") +  
 labs(title = "Distribution of Labels across Protocols",  
 x = "Protocols",  
 y = "Count") +  
 scale\_fill\_brewer(palette = "Set1")

 The Data appaers to be very skewed with most of the malicious attacks concentrated in the tcp protocol and benign attacks in the udp protocol.

## Dimensionality Reduction

This was the most challenging part of the project I attempted differerent approaches like using a stepwise linear discrimination, lemitization of IP addresses and ports to reduce dimensionality with limited success. I opted to create a subset of the data since I was also having problems pushing the huge original data file to Github. I decide to create a subset of the data with only the varaibles that I was using in my project.

# Removing some of the less important variables from the data   
data1 <- select(attacks,   
 label, ts,   
 id.resp\_h, id.resp\_p, id.orig\_p,  
 proto, history, orig\_ip\_bytes)  
  
# View the first few rows of the selected data  
summary(data1)

## label ts id.resp\_h   
## Benign :469275 Min. :2018-05-09 15:30:31.01 Length:1008748   
## Malicious:539473 1st Qu.:2018-05-10 17:57:40.75 Class :character   
## Median :2018-05-11 20:40:00.00 Mode :character   
## Mean :2018-05-11 21:43:27.22   
## 3rd Qu.:2018-05-13 01:13:40.00   
## Max. :2018-05-14 07:24:43.02   
## id.resp\_p id.orig\_p proto history   
## Min. : 0 Min. : 3 icmp: 17421 Length:1008748   
## 1st Qu.: 23 1st Qu.:43730 tcp :583134 Class :character   
## Median : 8080 Median :43763 udp :408193 Mode :character   
## Mean :16098 Mean :44437   
## 3rd Qu.:28180 3rd Qu.:48814   
## Max. :65535 Max. :65394   
## orig\_ip\_bytes   
## Min. : 0.00   
## 1st Qu.: 40.00   
## Median : 60.00   
## Mean : 81.15   
## 3rd Qu.: 60.00   
## Max. :2990.00

### Export the Subset For Use in Second Iteration of the Project

#write\_xlsx(data1, path = "data/data1.xlsx")   
#Used in the first Iteration to create current Data Set

## Separating Data into Train and Test

The train data will be used to calibrate the data and the test will be used to validate model predictions and generate a ROC curve.

# Create a data partition  
set.seed(5774)   
trainIndex <- createDataPartition(data1$label, p = 0.7, list = FALSE, times = 1)  
  
# Create training and test datasets  
train <- data1[trainIndex, ]  
test <- data1[-trainIndex, ]  
  
# Viewing the dimensions of the train and test sets  
dim(train)

## [1] 706125 8

dim(test)

## [1] 302623 8

# Count the number of occurrences for each level of the 'label' variable  
label\_counts <- table(attacks$label)  
  
# Print the counts  
print(label\_counts)

##   
## Benign Malicious   
## 469275 539473

## Logistic Regression Model

Our Logistic Regression Model will thus use the number of bytes sent from the origin and network protocol to determine if a network request is malicious or benign. The base equation for this model is

# Running a logistic regression model with received\_callback as the response variable  
# and years\_experience, race, and gender as explanatory variables  
  
mult\_log\_mod <- glm(label ~ orig\_ip\_bytes + proto,   
 data = train,   
 family = binomial)  
  
# Displaying the summary of the model  
tidy(mult\_log\_mod)

## # A tibble: 4 × 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) -17.7 35.7 -0.494 6.21e- 1  
## 2 orig\_ip\_bytes 0.00113 0.0000775 14.5 9.60e-48  
## 3 prototcp 20.1 35.7 0.561 5.74e- 1  
## 4 protoudp 8.19 35.7 0.229 8.19e- 1

## Results

### Equation for icmp

Intercept (β0): −17.66528932 represents the log odds of the outcome when all predictors are at their reference level O bytes sent sent and icmp protocol. Coefficient for orig\_ip\_bytes (β1): The coefficient 0.00112571, indicates the amount of change in the log odds of the outcome for a one-unit increase in ‘orig\_ip\_bytes’.

### Equation for TCP

Coefficient for prototcp is 8.19231031 this tells us that when the protocol is TCP (as opposed to ICMP), the likelihood of a malicious is 8.19231031 units higher than when the protocol is ICMP, while holding bytes sent from origin constant.

### Equation for UDP

Coefficient for prototcp is 20.06098491 this tells us that when the protocol is TCP (as opposed to ICMP), the likelihood of a malicious is 20.06098491 units higher than when the protocol is ICMP, while holding bytes sent from origin constant.

### Cross Validation

# Create 10-fold cross-validation sets  
set.seed(123) # for reproducibility  
folds <- createFolds(train$label, k = 10)  
  
# Function to perform cross-validation and compute AUC for each fold  
cv\_roc\_auc <- function(train\_data, folds) {  
 auc\_list <- c()  
   
 for(i in 1:length(folds)) {  
 # Splitting data into training and test sets  
 test\_indices <- folds[[i]]  
 train\_indices <- setdiff(seq\_len(nrow(train\_data)), test\_indices)  
   
 train\_fold <- train\_data[train\_indices, ]  
 test\_fold <- train\_data[test\_indices, ]  
   
 # Fit the model on the training set  
 glm\_model <- glm(label ~ orig\_ip\_bytes + proto, data = train\_fold, family = binomial)  
   
 # Predict probabilities on the test set  
 prob\_predictions <- predict(glm\_model, newdata = test\_fold, type = "response")  
   
 # Compute ROC AUC  
 roc\_curve <- roc(test\_fold$label, prob\_predictions)  
 auc\_value <- auc(roc\_curve)  
 auc\_list <- c(auc\_list, auc\_value)  
 }  
   
 return(auc\_list)  
}  
  
# Perform cross-validation and compute AUC  
auc\_results <- cv\_roc\_auc(train, folds)

## Setting levels: control = Benign, case = Malicious

## Setting direction: controls < cases

## Setting levels: control = Benign, case = Malicious

## Setting direction: controls < cases

## Setting levels: control = Benign, case = Malicious

## Setting direction: controls < cases

## Setting levels: control = Benign, case = Malicious

## Setting direction: controls < cases

## Setting levels: control = Benign, case = Malicious

## Setting direction: controls < cases

## Setting levels: control = Benign, case = Malicious

## Setting direction: controls < cases

## Setting levels: control = Benign, case = Malicious

## Setting direction: controls < cases

## Setting levels: control = Benign, case = Malicious

## Setting direction: controls < cases

## Setting levels: control = Benign, case = Malicious

## Setting direction: controls < cases

## Setting levels: control = Benign, case = Malicious

## Setting direction: controls < cases

# Calculate the mean AUC from all folds  
mean\_auc <- mean(auc\_results)  
  
# Output the results  
print(paste("Mean AUC from cross-validation:", mean\_auc))

## [1] "Mean AUC from cross-validation: 0.954349342908229"

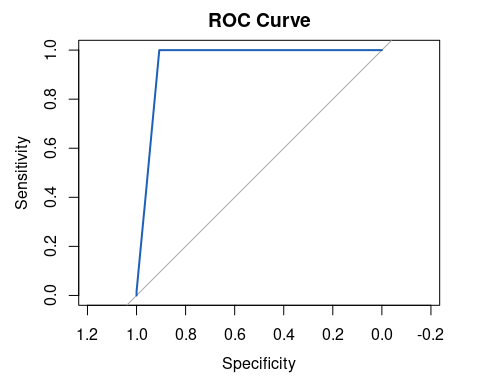
### ROC Curve

# First, predict the probabilities on the validation set (Split Test Data)  
prob\_predictions <- predict(mult\_log\_mod, newdata = test, type = "response")  
  
# Generate the ROC object  
roc\_obj <- roc(test$label, prob\_predictions)

## Setting levels: control = Benign, case = Malicious

## Setting direction: controls < cases

# Plot the ROC curve  
plot(roc\_obj, main = "ROC Curve", col = "#1c61b6", lwd = 2)



# Add AUC to the plot  
auc(roc\_obj)

## Area under the curve: 0.9541

## Conlusions and Reccommenadtions

AUC is approximately 0.954, which is very close to 1, indicating that the model has excellent discriminative ability to distinguish between the malicious and benign requests.

However this result should be taken with a grain of salt since the data is higly skewed with majority of Malicious attacks concentrated in the TCP protocol while a majority of benign attacks in the udp protocol.

A better approach to determine the efficiency of this model would be to use a more balanced dataset that distributes requests across the protocols. Further effort to lemitize and stem the IP addresses, Ports, History(Connections) would also help in providing more insightful patterns on network traffic and improving the models efficiency.

Also different approaches to discriminant analysis (instead of the cramer’s correlation) and clustering (instead of logistic regression) would have probably yielded different results.