

Content Page

Content Page	2
Data Management	3
Data Visualization	4
Data Analytics	12
Data Modelling	23
Regression using R	23
J48 Decision Tree	28
Naive Bayes	32
Multi Layer Perceptron	36
Recurrent Neural Network	39
References	42
Appendix	43
A. Decision Tree Function	43
B. Classification Metrics of Decision Tree	46
C. Naive Bayes Function	47
D. Classification Metrics of Naive Bayes	48
E. Multi Layer Perceptron Function	49
F. Classification Metrics of Multi Layer Perceptron	50
G. Recurrent Neural Network Notebook	51
H. Recurrent Neural Network Code	61

Data Management

The dataset is obtained from [1]. The dataset is called the Network Security Lab - Knowledge Discovery and Data Mining (NSL-KDD) dataset. The aim of this dataset is to act as an effective benchmark data set to help researchers compare various intrusion detection methods [2].

The dataset can either be downloaded from the [Github repository](#) or hosted directly at the Canadian Institute for Cybersecurity, University of Brunswick's [website](#).

From the repository, the NSL-KDD dataset has 2 parts, the training and test sets. They are available in either .csv or .arff file formats.

The training set has 125,973 rows whereas the test set has 22,543 rows. There are 41 feature columns along with an associated class label (*xAttack*).

To keep our analysis unbiased, we will only be analysing the training set file. The test set file will only be used to obtain the test set results for each classification model.

The features of this dataset comes from network data captured and categorized into 5 labels: normal, probes, remote-to-local attack, user-to-root attack, denial-of-service attack. The features represent network data such as protocol type, number of failed logins and source bytes.

All of the features have already been converted to integer or floats.

Data Visualization

Distribution of xAttack Labels

For the *xAttack* labels, the counts and pie chart (detailing the distribution for each label) are as follows:

xAttack Category	Counts
1 (<i>dos</i>): Denial of service attacks. The attacker aims to make the host too busy to process other requests from legitimate users.	45,927
2 (<i>u2r</i>): User to Root attacks. The attacker would exploit vulnerabilities to gain root privileges.	52
3 (<i>r2l</i>): Remote to Local attacks. The attacker would exploit vulnerabilities in the system to gain local access.	995
4 (<i>probe</i>). The attacker would attempt to gather important reconnaissance information about a network.	11,656
5 (<i>normal</i>): Benign, legitimate communications.	67,343

Table 1. Counts for each *xAttack* label

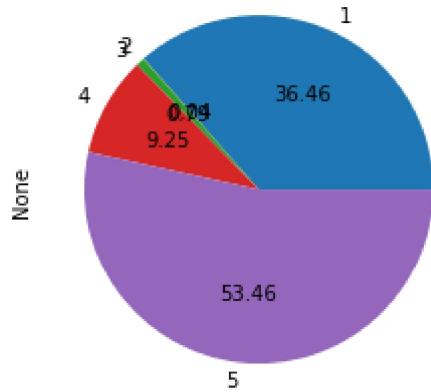


Figure 1. Pie chart showing distribution of *xAttack* labels

We can observe that there is a significant class imbalance of *xAttack* classes 2 (*u2r*) and 3 (*r2l*).

Other Features

The other 41 features are as follows:

No.	Feature name	No.	Feature name
1	duration	21	is_hot_login
2	protocol_type	22	is_guest_login
3	service	23	count
4	src_bytes	24	serror_rate
5	dst_bytes	25	rerror_rate
6	flag	26	same_srv_rate
7	land	27	diff_srv_rate
8	wrong_fragment	28	srv_count
9	urgent	29	srv_serror_rate
10	hot	30	srv_rerror_rate
11	num_failed_logins	31	srv_diff_host_rate
12	logged_in	32	dst_host_count
13	num_compromised	33	dst_host_srv_count
14	root_shell	34	dst_host_same_srv_rate
15	su_attempted	35	dst_host_diff_srv_rate
16	num_root	36	dst_host_same_src_port_rate
17	num_file_creations	37	dst_host_srv_diff_host_rate
18	num_shells	38	dst_host_serror_rate
19	num_access_files	39	dst_host_srv_serror_rate
20	num_outbond_cmd	40	dst_host_rerror_rate

Table 2. Features in NSL-KDD dataset

Most of the features are quantitative values. Filtering for categorical values (ie features with small number of values yields the following):

```
for col in train_set:
    unique_vals = (train_set[col].unique())
    if len(unique_vals) < 50:
        print(col, 'has ', len(unique_vals), 'unique vals. They are:', np.sort(unique_vals))

protocol_type has 3 unique vals. They are: [1 2 3]
flag has 10 unique vals. They are: [ 1  2  3  4  5  6  7  8  9 11]
land has 2 unique vals. They are: [0 1]
wrong_fragment has 3 unique vals. They are: [0 1 3]
urgent has 4 unique vals. They are: [0 1 2 3]
hot has 28 unique vals. They are: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 14 15 17 18 19 20 21 22 24 25
28 30 33 44 77]
num_failed_logins has 6 unique vals. They are: [0 1 2 3 4 5]
logged_in has 2 unique vals. They are: [0 1]
root_shell has 2 unique vals. They are: [0 1]
su_attempted has 3 unique vals. They are: [0 1 2]
num_file_creations has 35 unique vals. They are: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 25 26 27 28 29 33 34 36 38 40 43]
num_shells has 3 unique vals. They are: [0 1 2]
num_access_files has 10 unique vals. They are: [0 1 2 3 4 5 6 7 8 9]
num_outbound_cmds has 1 unique vals. They are: [0]
is_host_login has 2 unique vals. They are: [0 1]
is_guest_login has 2 unique vals. They are: [0 1]
xAttack has 5 unique vals. They are: [1 2 3 4 5]
```

Python output of features with few categorical values

We can observe that the categorical features have already been encoded into integer variables.

Note that the feature *num_outbound_cmds* has only one possible value, 0. This could be a possible redundant column.

Correlation Analysis

As there are a whopping 41 features, we will be focusing our visualizations only on the more relevant features. We have decided to limit our scope to features that are strongly correlated with the *xAttack* labels. In this case, only features with correlation magnitudes larger than 0.7 will be analyzed via visualizations.

```
# Get features which are correlated with xAttack labels ie correlation magnitude of above 0.7
correlations = train_set.corr()
for index, row in (correlations[['xAttack']]).iterrows():
    correlation_with_xattack = row['xAttack']
    if abs(correlation_with_xattack) > 0.7:
        print(index, correlation_with_xattack)

serror_rate -0.7779543353136086
srv_serror_rate -0.7767725719685258
same_srv_rate 0.8203949901293894
dst_host_same_srv_rate 0.700547657783034
dst_host_serror_rate -0.7799095886878171
dst_host_srv_serror_rate -0.7827871811434195
```

Python output of features strongly correlated with *xAttack*

The features which have correlations magnitudes larger than 0.7 are *serror_rate*, *srv_serror_rate*, *same_srv_rate*, *dst_host_same_srv_rate*, *dst_host_serror_rate* and *dst_host_srv_serror_rate*.

Scatterplots vs xAttack Labels

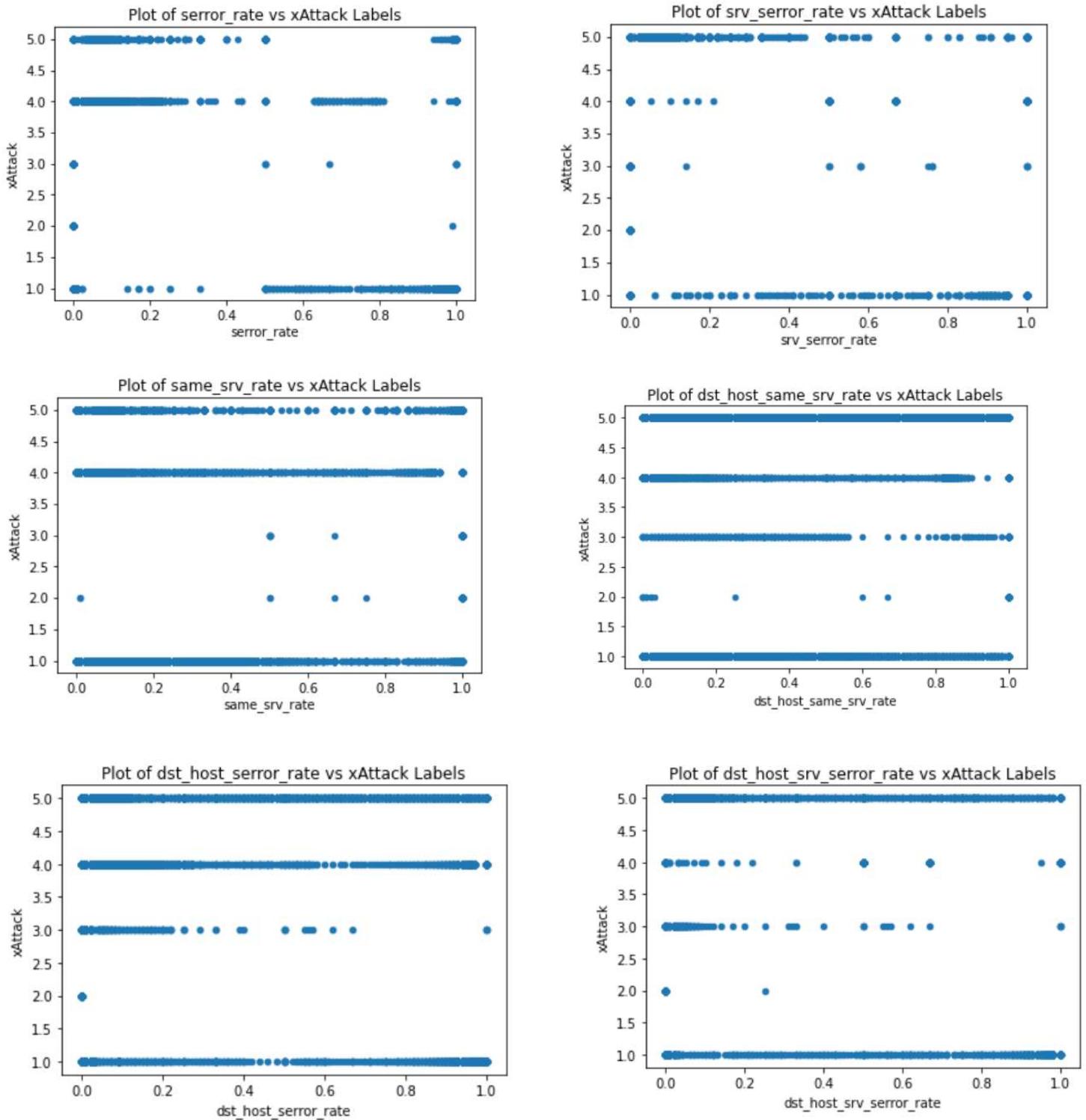


Figure 2. Scatterplots of top 6 strongly correlated features vs xAttack labels

No clear distinction can be seen to distinguish each label. Hence, a linear classifier may not be adequate.

Combining xAttack Labels

As there is a significant class imbalance for some of the classes (u2r and r2l), the labels will be combined into benign (5) and malicious (1-4).

```
train_set['xAttack'] = np.where(train_set['xAttack'] == 5, 0, 1)
test_set['xAttack'] = np.where(test_set['xAttack'] == 5, 0, 1)
```

Python code re-assigning labels to Benign vs Malicious

New xAttack Label	Counts
0 ie benign	67343
1 ie malicious	58630

Table 3. Counts for Benign vs Malicious labels

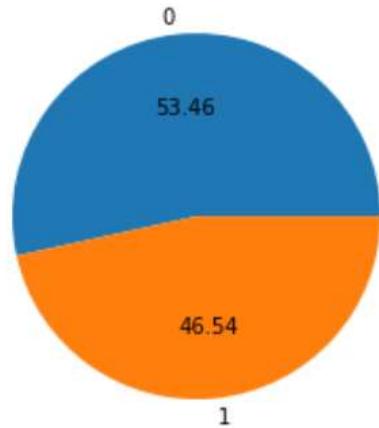


Figure 3. Pie chart showing new distribution of xAttack labels

After adjusting the labels, the number of samples for each class is much more balanced. This will address the class imbalance problem we faced earlier. Hence, our subsequent analysis will be used on this dataset of binary (malicious vs benign) labels.

Correlation Analysis

When we redid the correlation analysis, there are only 2 features which are strongly correlated with the new labels.

```
# Get features which are correlated with xAttack labels ie correlation magnitude of above 0.7
correlations = train_set.corr()
for index, row in (correlations[['xAttack']]).iterrows():
    correlation_with_xattack = row['xAttack']
    if abs(correlation_with_xattack) > 0.7:
        print(index, correlation_with_xattack)

same_srv_rate -0.7519134368764228
dst_host_srv_count -0.7225353705394968
```

Python output of features strongly correlated with binary *xAttack* labels

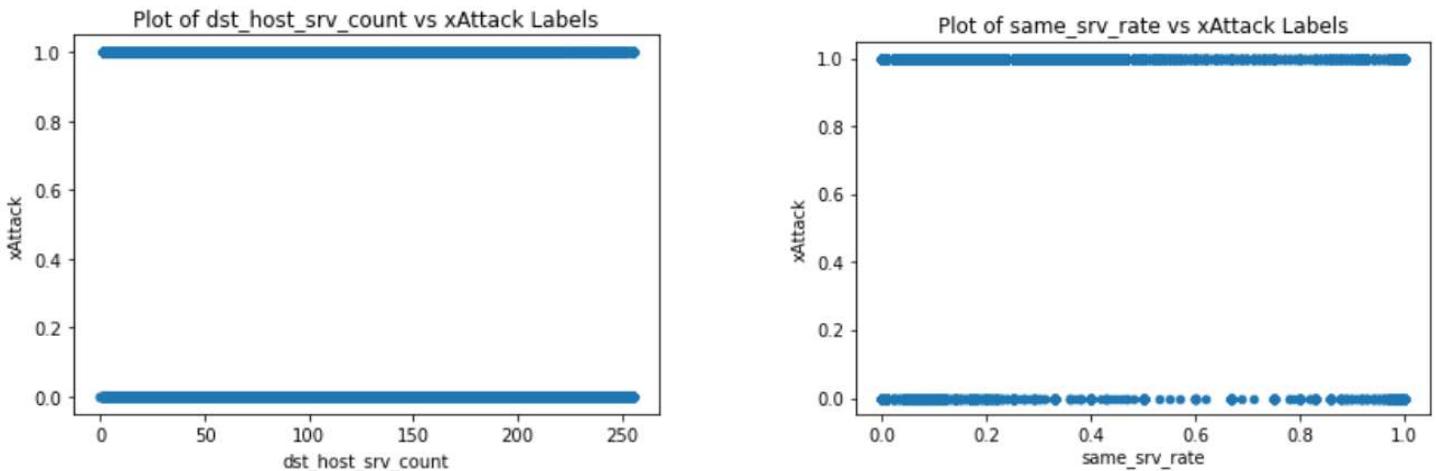


Figure 4. Scatterplots of top 2 strongly correlated features vs xAttack labels

Similar as before, we cannot see any differences between the classes for each single feature.

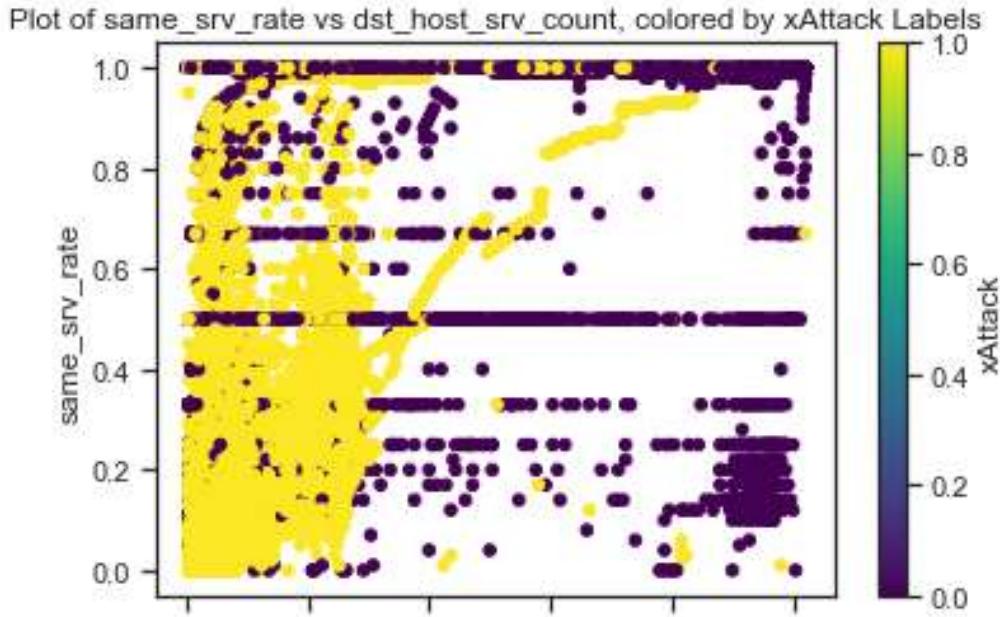


Figure 5. Scatterplot of *same_srv_rate* vs *dst_host_srv_count* colored by *xAttack* labels

However, if we plot *same_srv_rate* vs *dst_host_srv_count*, we can see that most of the malicious points (*xAttack* == 1) are at the left side of the graph, with what appears to be a curved boundary.

Hence, it would be easier to train a model classifying benign vs malicious rather than for all 5 separate classes. This supports our earlier decision to combine the labels, rather than use the 5 separate labels with imbalanced data.

Data Analytics

Correlation Analysis

The full table (of correlations with the xAttack labels) is as follows:

Feature name	Correlation
same_srv_rate	-0.7519134369
dst_host_srv_count	-0.7225353705
dst_host_same_srv_rate	-0.6938028328
logged_in	-0.6901707396
protocol_type	-0.1356434215
srv_diff_host_rate	-0.1193771824
is_guest_login	-0.03927902588
num_access_files	-0.03670050589
su_attempted	-0.02244849665
num_file_creations	-0.02127072945
root_shell	-0.02028540195
hot	-0.01308341431
num_root	-0.01145249134
num_compromised	-0.01019832758
num_shells	-0.009471997357
num_failed_logins	-0.003755073924
urgent	-0.002787032403
is_host_login	-0.002628915973
srv_count	0.0007707123314
dst_bytes	0.004117535029
src_bytes	0.005921322756
land	0.007190758995
duration	0.04878532742
dst_host_srv_diff_host_rate	0.06233233827
dst_host_same_src_port_rate	0.09244443089
wrong_fragment	0.0959054807
diff_srv_rate	0.2036599647

dst_host_diff_srv_rate	0.2428979585
dst_host_error_rate	0.2525627248
error_rate	0.253397111
srv_error_rate	0.2535039621
service	0.2790234199
dst_host_count	0.3750520924
flag	0.5087038065
count	0.5764442653
srv_serror_rate	0.6482888316
serror_rate	0.6506522402
dst_host_serror_rate	0.6518416167
dst_host_srv_serror_rate	0.6549854452
num_outbond_cmd	NA

Table 4. Table of features with their correlations with respect to *xAttack* labels

Highlighted in yellow are features with correlation magnitudes larger than 0.7. Highlighted in green are features with correlation magnitudes larger than 0.65.

Note that *num_outbond_cmd* is NA as it only has 1 value throughout the dataset.

Hence, there are 7 important features when we consider correlation magnitudes: *same_srv_rate*, *dst_host_srv_count*, *dst_host_same_srv_rate*, *logged_in*, *serror_rate*, *dst_host_serror_rate*, *dst_host_srv_serror_rate*

Feature Ranking using Weka

```
==== Attribute Selection on all input data ====

Search Method:
    Greedy Stepwise (forwards).
    Start set: no attributes
    Merit of best subset found:      0.816

Attribute Subset Evaluator (supervised, Class (numeric): 42 xAttack):
    CFS Subset Evaluator
    Including locally predictive attributes

Selected attributes: 8,12,29,33,35,39 : 6
    wrong_fragment
    logged_in
    same_srv_rate
    dst_host_srv_count
    dst_host_diff_srv_rate
    dst_host_srv_serror_rate
```

Output for attribute selection using CFS Subset Evaluator

Using the CFS Subset Evaluator, the feature ranking (top 6) is: *wrong_fragment, logged_in, same_srv_rate, dst_host_srv_count, dst_host_diff_srv_rate, dst_host_srv_serror_rate.*

```
==== Attribute Selection on all input data ====
```

```
Search Method:
```

```
    Attribute ranking.
```

```
Attribute Evaluator (supervised, Class (nominal): 42 class):  
    Gain Ratio feature evaluator
```

```
Ranked attributes:
```

0.2681	28	srv_error_rate
0.2323	12	logged_in
0.2288	41	dst_host_srv_error_rate
0.2181	27	rerror_rate
0.2158	4	flag
0.2127	6	dst_bytes
0.1975	5	src_bytes
0.1935	30	diff_srv_rate
0.163	29	same_srv_rate
0.1617	40	dst_host_rerror_rate
0.1431	3	service
0.1417	25	serror_rate
0.137	26	srv_serror_rate
0.1367	39	dst_host_srv_serror_rate
0.133	34	dst_host_same_srv_rate
0.1203	31	srv_diff_host_rate
0.1198	35	dst_host_diff_srv_rate
0.1182	33	dst_host_srv_count
0.1178	37	dst_host_srv_diff_host_rate
0.1117	11	num_failed_logins

```
Output for attribute selection using GainRatioAttributeEval with Ranker
```

Using the GainRatioAttributeEval with Ranker, the top 6 features are: *srv_error_rate*, *logged_in*, *dst_host_srv_error_rate*, *rerror_rate*, *flag*, *dst_bytes*.

```
==== Attribute Selection on all input data ====  
  
Search Method:  
    Attribute ranking.  
  
Attribute Evaluator (supervised, Class (nominal): 42 class):  
    Information Gain Ranking Filter  
  
Ranked attributes:  
0.77273      5 src_bytes  
0.692733     6 dst_bytes  
0.484801     3 service  
0.340669     4 flag  
0.337167     33 dst_host_srv_count  
0.334063     35 dst_host_diff_srv_rate  
0.326465     34 dst_host_same_srv_rate  
0.307324     40 dst_host_rerror_rate  
0.26572      41 dst_host_srv_rerror_rate  
0.259631     23 count  
0.25786      30 diff_srv_rate  
0.243368     29 same_srv_rate  
0.230089     12 logged_in  
0.226113     27 rerror_rate  
0.217487     28 srv_rerror_rate  
0.190418     37 dst_host_srv_diff_host_rate  
0.165708     36 dst_host_same_src_port_rate  
0.147335     31 srv_diff_host_rate  
0.122839     38 dst_host_serror_rate  
0.122387     32 dst_host_count  
0.113507     25 serror_rate  
0.108604     39 dst_host_srv_serror_rate  
0.09408      1 duration  
0.092326     26 srv_serror_rate  
0.085731     24 srv_count  
0.040533     2 protocol_type  
0.031069     10 hot  
0.016736     11 num_failed_logins  
0.013958     22 is_guest_login  
0.011118     13 num_compromised  
0.000993     16 num_root  
0.000937     8 wrong_fragment
```

Output for attribute selection using InfoRatioAttributeEval with Ranker

Using the InfoGainAttributeEval with Ranker, the top 6 features are: src_bytes, dst_bytes, service, flag, dst_host_srv_count, dst_host_diff_srv_rate.

Analysis of Variance (Single Factor)

Single-factor ANOVA is done on features that have both higher correlation magnitudes and are ranked highly from Weka.

These features are: *same_srv_rate*, *dst_host_srv_count*, *logged_in*, *dst_host_srv_error_rate*, *dst_bytes*

SUMMARY					
Groups	Count	Sum	Average	Variance	
dst_bytes_benign	58630	2.5E+08	4248.16	3.7E+09	
dst_bytes_malicious	58630	2.2E+09	37524.5	3.5E+13	
ANOVA					
Source of Variation	SS	df	MS	F	P-value
Between Groups	3.2E+13	1	3.2E+13	1.86864	0.17163
Within Groups	2E+18	117258	1.7E+13		
Total	2E+18	117259			

Anova results for the feature *dst_bytes*

SUMMARY					
Groups	Count	Sum	Average	Variance	
same_srv_rate_benign	58630	56833.6	0.96936	0.02079	
same_srv_rate_malicious	58630	17979.4	0.30666	0.15654	
ANOVA					
Source of Variation	SS	df	MS	F	P-value
Between Groups	12874.4	1	12874.4	145202	0
Within Groups	10396.7	117258	0.08867		
Total	23271.1	117259			

Anova results for the feature *same_srv_rate*

SUMMARY				
Groups	Count	Sum	Average	Variance
dst_host_srv_count_benign	58630	1.1E+07	190.325	8570.87
dst_host_srv_count_malicious	58630	1754742	29.9291	2734.17

ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	7.5E+08	1	7.5E+08	133424	0	3.84154
Within Groups	6.6E+08	117258	5652.52			
Total	1.4E+09	117259				

Anova results for the feature *dst_host_srv_count*

SUMMARY				
Groups	Count	Sum	Average	Variance
dst_host_srv_serror_rate_benign	58630	354.1	0.00604	0.00313
dst_host_srv_serror_rate_malicious	58630	34669.6	0.59133	0.23998

ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	10042.2616	1	10042.3	82613	0	3.84154
Within Groups	14253.6289	117258	0.12156			
Total	24295.8905	117259				

Anova results for the feature *dst_host_srv_serror_rate*

SUMMARY				
Groups	Count	Sum	Average	Variance
logged_in_benign	58630	41624	0.70994	0.20593
logged_in_malicious	58630	1995	0.03403	0.03287

ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	13392.9528	1	13393	112170	0	3.84154
Within Groups	14000.4191	117258	0.1194			
Total	27393.3718	117259				

Anova results for the feature *logged_in*

From all the above ANOVA outputs, all the p-values (except for *dst_bytes*) are less than the significance value, 0.05. This means that the null hypothesis (H_0 : Both groups are not different) is rejected. Hence, we can conclude that based on each separate feature, both benign and malicious groups are different.

Hence, these features (except for *dst_bytes*) are significant.

Choice of Features

Ranking (1 - Most Important)	Correlation	CFS Subset Evaluator	Gain Ratio Ranking	Info Gain Ranking
1	same_srv_rate	wrong_fragment	srv_error_rate	src_bytes
2	dst_host_srv_count	logged_in	logged_in	dst_bytes
3	dst_host_same_srv_rate	same_srv_rate	dst_host_srv_error_rate	service
4	logged_in	dst_host_srv_count	error_rate	flag
5	dst_host_srv_error_rate	dst_host_diff_srv_rate	flag	dst_host_srv_count
6	dst_host_error_rate	dst_host_srv_error_rate	dst_bytes	dst_host_diff_srv_rate

Table 5. Overall ranking of top features

The top 4 features shared by all 4 measures are: *dst_bytes*, *dst_host_srv_count*, *logged_in*, *dst_host_srv_error_rate*

However, *dst_bytes* is dropped as it is not significant in the single factor ANOVA.

Hence, the 3 chosen features are: *dst_host_srv_count*, *logged_in*, *dst_host_srv_error_rate*

They are chosen as:

1. They have a high magnitude of correlation with respect to the class labels.
2. Weka has ranked them highly when using the CFS Subset Evaluator, Gain Ratio Ranking and Info Gain Ranking.
3. These features are significant in single-factor ANOVA.

Parallel Coordinates Visualizations

With the top 4 features, we can have an easier time in visualizing the parallel coordinate plots.

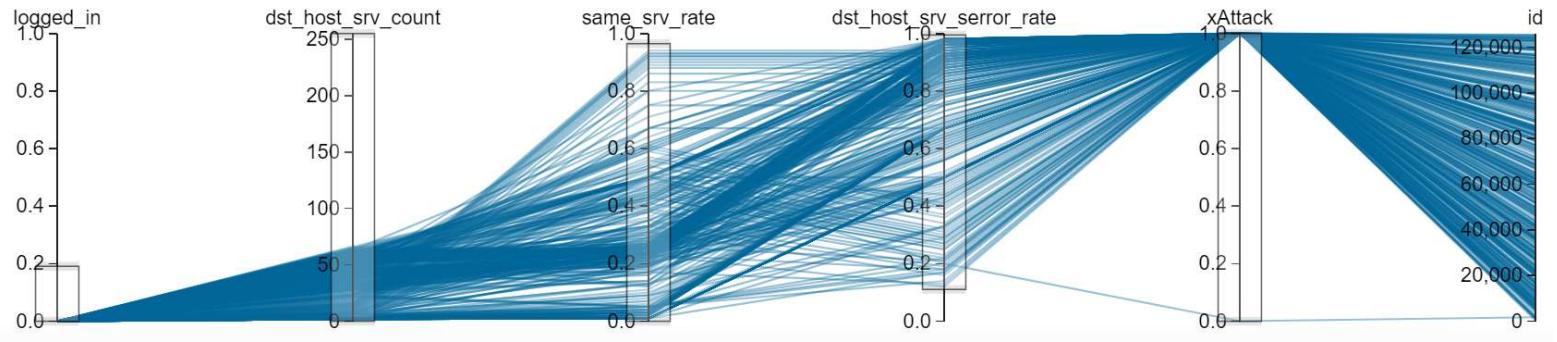


Figure 6. Parallel Coordinate Visualization I

Observe that when `logged_in` (a binary variable) is 0, `same_srv_rate` is slightly below 1 and `dst_host_srv_serror_rate` is more than 0.1, the majority of the points are classified as malicious (`xAttack` = 1).

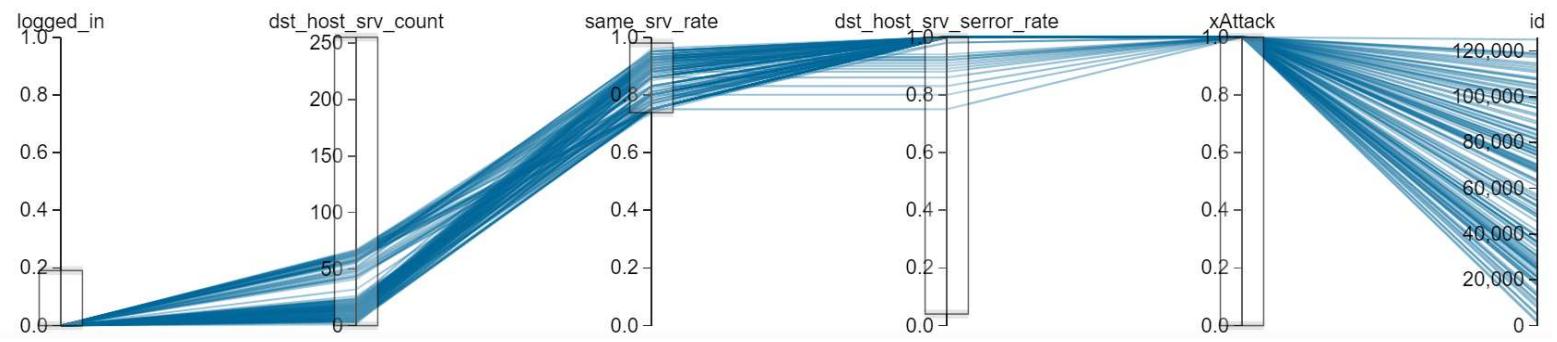


Figure 7. Parallel Coordinate Visualization II

The effect is even more clear, when we further restrict `same_srv_rate` to between 0.75 to below 1.0.

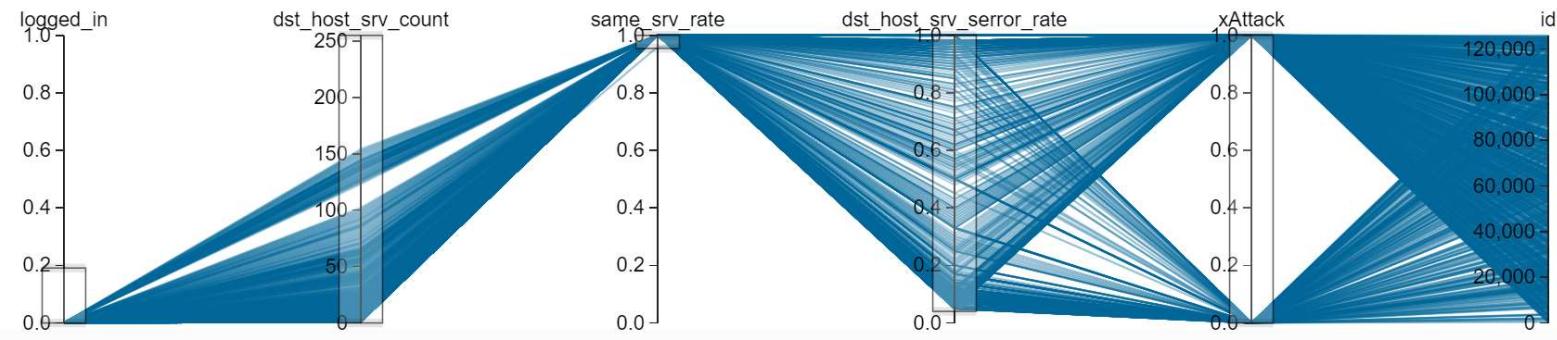


Figure 8. Parallel Coordinate Visualization III

However, when we consider `same_srv_rate` to be 1.0, there are both lots of malicious (`xAttack` = 1) and benign (`xAttack` = 0) samples.

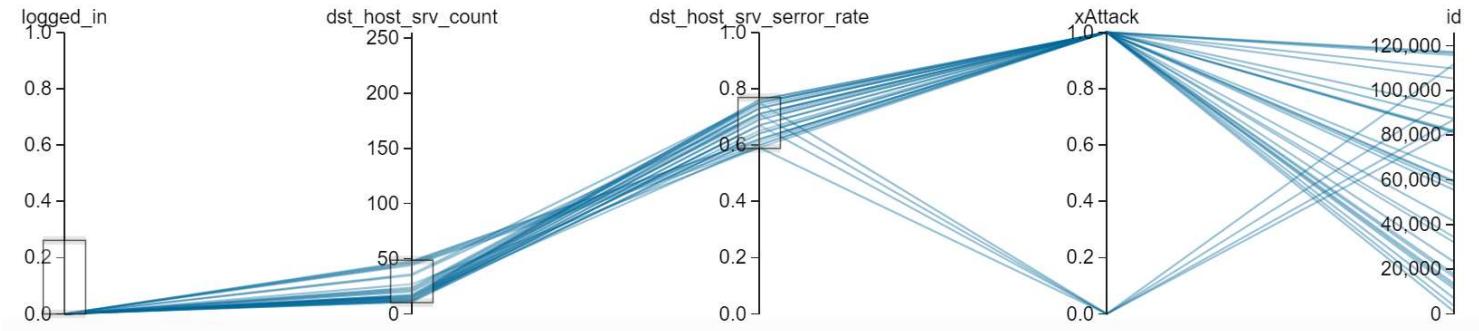


Figure 9. Parallel Coordinate Visualization IV

With the chosen 3 features, we can also see a trend where when $\text{logged_in} = 0$, $\text{dst_host_srv_count}$ is between 25 and 50 and $\text{dst_host_srv_error_rate}$ is between 0.6 and 0.8, the majority of the data points are classified as malicious ($x\text{Attack} = 1$).

Hence, this shows that visually, the 3 chosen features are significant in deciding the class labels.

K-Means Clustering

Using the following R code, the total within cluster sum of squares is obtained with each cluster number.

```
# Load libraries
library(cluster)    # clustering algorithms
library(factoextra) # clustering algorithms & visualization
library(tidyverse)

# Do K-means
# function to compute total within-cluster sum of square
wss <- function(k) {
  kmeans(train_set, k, nstart = 10 )$tot.withinss
}

# Compute and plot wss for k = 1 to k = 15
k.values <- 1:15

# extract wss for 2-15 clusters
wss_values <- map_dbl(k.values, wss)

plot(k.values, wss_values,
  type="b", pch = 19, frame = FALSE,
  xlab="Number of clusters K",
  ylab="Total within-clusters sum of squares")
```

R code for plotting WSS cluster graph

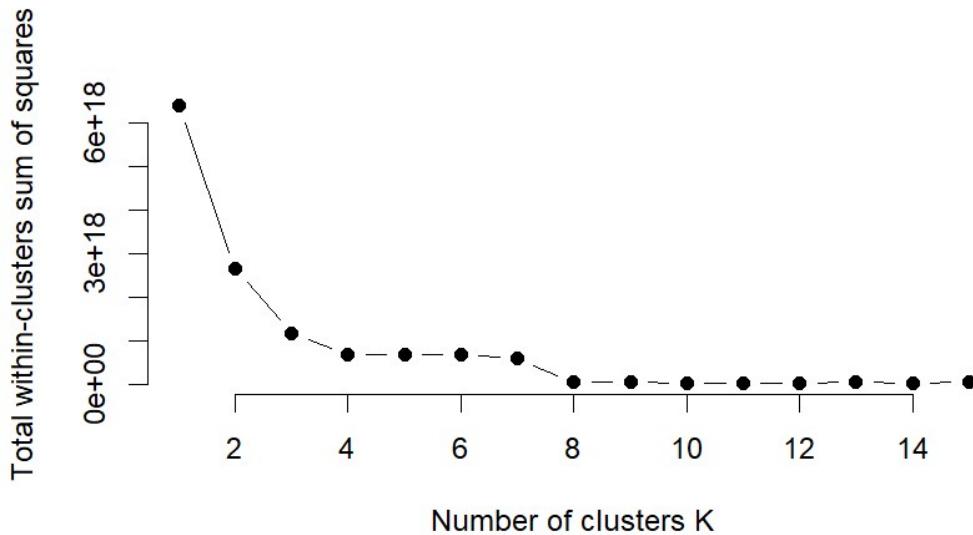


Figure 10. Total within cluster sum of squares for each cluster size

From the above plot, we can see that there is an ‘elbow’ at $k=2$ and $k=8$. The ‘elbow’ can be described as the point where there is a change from a steep negative gradient to a less steep negative gradient.

Hence, this supports our decision to use 2 classes instead of the 5 given labels.

Data Modelling

Regression using R

For regression, we will limit our scope to the top 5 features. This will give us $2^5 - 1 = 31$ possible models. If we had only used our chosen 3 features, we would only have $2^3 - 1 = 8$ possible models. This would not allow us to run at least 10 experiments.

We use the ‘olsrr’ package in R to create all the possible values.

The top 5 features we will use (from *Choice of Features* section) are: *dst_bytes*, *dst_host_srv_count*, *logged_in*, *dst_host_srv_serror_rate*, *same_srv_rate*.

```
library('olsrr')

model <- lm(xAttack ~ dst_bytes + dst_host_srv_count +
             logged_in + dst_host_srv_serror_rate + same_srv_rate,
             data = train_set)
k <- ols_step_all_possible(model)
(k)
```

R code for generating all possible regression models

Index	N	Predictors	R-Square	Adj. R-Square	Mallow's Cp
5	1 1	same_srv_rate	5.653738e-01	5.653704e-01	5.363051e+04
2	2 1	dst_host_srv_count	5.220574e-01	5.220536e-01	7.153006e+04
3	3 1	logged_in	4.763356e-01	4.763315e-01	9.042354e+04
4	4 1	dst_host_srv_serror_rate	4.290059e-01	4.290014e-01	1.099815e+05
1	5 1	dst_bytes	1.695409e-05	9.015894e-06	2.872515e+05
14	6 2	logged_in same_srv_rate	6.544312e-01	6.544257e-01	1.683155e+04
12	7 2	dst_host_srv_count same_srv_rate	6.388481e-01	6.388424e-01	2.327091e+04
10	8 2	dst_host_srv_count logged_in	6.157079e-01	6.157018e-01	3.283310e+04
11	9 2	dst_host_srv_count dst_host_srv_serror_rate	6.105236e-01	6.105174e-01	3.497539e+04
13	10 2	logged_in dst_host_srv_serror_rate	6.070905e-01	6.070845e-01	3.639395e+04
15	11 2	dst_host_srv_serror_rate same_srv_rate	5.806410e-01	5.806344e-01	4.732367e+04
9	12 2	dst_bytes same_srv_rate	5.654223e-01	5.654154e-01	5.361246e+04
6	13 2	dst_bytes dst_host_srv_count	5.220585e-01	5.220509e-01	7.153160e+04
7	14 2	dst_bytes logged_in	4.763401e-01	4.763318e-01	9.042368e+04
8	15 2	dst_bytes dst_host_srv_serror_rate	4.290425e-01	4.290334e-01	1.099684e+05
23	16 3	dst_host_srv_count logged_in same_srv_rate	6.856981e-01	6.856906e-01	3.913212e+03
22	17 3	dst_host_srv_count logged_in dst_host_srv_serror_rate	6.718476e-01	6.718398e-01	9.636602e+03
25	18 3	logged_in dst_host_srv_serror_rate same_srv_rate	6.653025e-01	6.652946e-01	1.234122e+04
20	19 3	dst_bytes logged_in same_srv_rate	6.544566e-01	6.544483e-01	1.682307e+04
24	20 3	dst_host_srv_count dst_host_srv_serror_rate same_srv_rate	6.503581e-01	6.503497e-01	1.8511668e+04
18	21 3	dst_bytes dst_host_srv_count same_srv_rate	6.388669e-01	6.388583e-01	2.326516e+04
16	22 3	dst_bytes dst_host_srv_count logged_in	6.157088e-01	6.156996e-01	3.283471e+04
17	23 3	dst_bytes dst_host_srv_count dst_host_srv_serror_rate	6.105326e-01	6.105233e-01	3.497368e+04
19	24 3	dst_bytes logged_in dst_host_srv_serror_rate	6.071062e-01	6.070968e-01	3.638954e+04
21	25 3	dst_bytes dst_host_srv_serror_rate same_srv_rate	5.806897e-01	5.806797e-01	4.730557e+04
30	26 4	dst_host_srv_count logged_in dst_host_srv_serror_rate same_srv_rate	6.951494e-01	6.951397e-01	9.685807e+00
27	27 4	dst_bytes dst_host_srv_count logged_in same_srv_rate	6.857113e-01	6.857013e-01	3.909767e+03
26	28 4	dst_bytes dst_host_srv_count logged_in dst_host_srv_serror_rate	6.718542e-01	6.718437e-01	9.635901e+03
29	29 4	dst_bytes logged_in dst_host_srv_serror_rate same_srv_rate	6.653284e-01	6.653178e-01	1.233252e+04
30	30 4	dst_bytes dst_host_srv_count dst_host_srv_serror_rate same_srv_rate	6.503774e-01	6.503663e-01	1.851068e+04
31	31 5	dst_bytes dst_host_srv_count logged_in dst_host_srv_serror_rate same_srv_rate	6.951631e-01	6.951510e-01	6.000000e+00

R output for all possible regression models

From the above printout, the model with the highest adjusted R² value is 0.6951510. It is the model with all 5 predictors: *dst_bytes*, *dst_host_srv_count*, *logged_in*, *dst_host_srv_serror_rate*, *same_srv_rate*.

```

> summary(model)

Call:
lm(formula = xAttack ~ dst_bytes + dst_host_srv_count + logged_in +
    dst_host_srv_serror_rate + same_srv_rate, data = train_set)

Residuals:
    Min      1Q  Median      3Q     Max 
-0.88831 -0.07765  0.01758  0.02863  1.02863 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept) 8.798e-01 2.561e-03 343.592 <2e-16 ***
dst_bytes    4.601e-10 1.930e-10  2.384  0.0171 *  
dst_host_srv_count -1.175e-03 1.058e-05 -111.034 <2e-16 ***
logged_in   -2.890e-01 2.124e-03 -136.039 <2e-16 *** 
dst_host_srv_serror_rate 1.696e-01 2.713e-03  62.496 <2e-16 *** 
same_srv_rate -3.198e-01 3.259e-03 -98.142 <2e-16 *** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2754 on 125967 degrees of freedom
Multiple R-squared:  0.6952,    Adjusted R-squared:  0.6952 
F-statistic: 5.745e+04 on 5 and 125967 DF,  p-value: < 2.2e-16

```

R output for model with highest adjusted R-squared value

Fitting this model, we can see that all variables are significant for each t-test. The t-test is a hypothesis test where $H_0: \beta_j = 0$ vs $H_1: \beta_j \neq 0$ for variable j. If H_0 is not rejected, it implies that variable j can be deleted from the model. If H_0 is rejected ($p\text{-value} < 0.05$), it implies that variable j is significant and cannot be deleted from the model.

From the above model summary, all the p-values are lower than 0.05. Hence, we can conclude that all the variables are significant in predicting *xAttack* in a regression model.

Stepwise Regression

It is impossible to enumerate all possible models, as we did above, as we would be dealing with $2^{41} - 1$ possible models. Hence, to investigate all 41 features, we will be using stepwise regression.

```
# Do stepwise regression on all features
require('MASS')
full.model <- lm(xAttack ~., data = train_set)
step.model <- stepAIC(full.model, direction = "both",
                      trace = FALSE)
summary(step.model)

Call:
lm(formula = xAttack ~ duration + protocol_type + service + flag +
    src_bytes + land + wrong_fragment + hot + num_failed_logins +
    logged_in + num_compromised + root_shell + su_attempted +
    num_root + num_file_creations + num_access_files + is_guest_login +
    count + srv_count + serror_rate + srv_serror_rate + rerror_rate +
    srv_rerror_rate + same_srv_rate + diff_srv_rate + srv_diff_host_rate +
    dst_host_count + dst_host_srv_count + dst_host_same_srv_rate +
    dst_host_diff_srv_rate + dst_host_same_src_port_rate + dst_host_srv_diff_host_rate +
    dst_host_serror_rate + dst_host_srv_serror_rate + dst_host_rerror_rate +
    dst_host_srv_rerror_rate, data = train_set)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.97632 -0.04576 -0.00898  0.07275  1.27946 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 6.394e-01 7.329e-03 87.243 < 2e-16 ***
duration    -2.075e-06 2.676e-07 -7.756 8.84e-15 ***
protocol_type -1.420e-01 2.092e-03 -67.874 < 2e-16 ***
service     -7.938e-05 4.336e-05 -1.831 0.06715 .
flag         4.209e-02 1.114e-03 37.780 < 2e-16 ***
src_bytes   6.278e-10 1.015e-10  6.188 6.12e-10 ***
land        -5.446e-01 4.243e-02 -12.834 < 2e-16 ***
wrong_fragment 3.090e-01 2.520e-03 122.619 < 2e-16 ***
hot          2.595e-02 5.433e-04 47.761 < 2e-16 ***
num_failed_logins 3.175e-02 1.327e-02  2.392 0.01676 *  
logged_in    -1.545e-01 3.788e-03 -40.804 < 2e-16 ***
num_compromised 8.435e-03 7.389e-04 11.415 < 2e-16 ***
root_shell   1.560e-01 2.053e-02  7.598 3.03e-14 ***
su_attempted -6.092e-02 2.114e-02 -2.883 0.00395 ** 
num_root     -8.396e-03 7.359e-04 -11.409 < 2e-16 ***
num_file_creations -1.102e-02 1.253e-03 -8.799 < 2e-16 ***
num_access_files 3.785e-02 8.206e-03  4.613 3.98e-06 ***
is_guest_login -2.778e-01 1.224e-02 -22.699 < 2e-16 ***
count         3.184e-05 1.152e-05  2.763 0.00572 ** 
srv_count    1.197e-03 1.597e-05 74.982 < 2e-16 ***
serror_rate  -3.368e-01 1.391e-02 -24.206 < 2e-16 ***
srv_serror_rate 2.621e-01 1.500e-02 17.471 < 2e-16 ***
rerror_rate  -2.570e-01 1.471e-02 -17.468 < 2e-16 ***
srv_rerror_rate 3.484e-01 1.498e-02 23.257 < 2e-16 ***
same_srv_rate -5.311e-01 4.689e-03 -113.261 < 2e-16 ***
diff_srv_rate -1.841e-01 4.779e-03 -38.525 < 2e-16 ***
```

```

srv_diff_host_rate      1.152e-01  2.632e-03  43.771 < 2e-16 ***
dst_host_count          5.686e-04  9.088e-06  62.569 < 2e-16 ***
dst_host_srv_count      -1.035e-03 1.544e-05  -67.045 < 2e-16 ***
dst_host_same_srv_rate  2.151e-01  4.597e-03  46.786 < 2e-16 ***
dst_host_diff_srv_rate  1.497e-01  5.249e-03  28.519 < 2e-16 ***
dst_host_same_src_port_rate 3.432e-01  2.785e-03  123.224 < 2e-16 ***
dst_host_srv_diff_host_rate 5.757e-01  6.889e-03  83.563 < 2e-16 ***
dst_host_serror_rate    1.454e-01  8.889e-03  16.356 < 2e-16 ***
dst_host_srv_serror_rate 1.638e-01  1.057e-02  15.498 < 2e-16 ***
dst_host_rerror_rate    3.362e-02  6.092e-03   5.519 3.42e-08 ***
dst_host_srv_rerror_rate 1.188e-01  8.360e-03  14.207 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```

Residual standard error: 0.2105 on 125936 degrees of freedom
Multiple R-squared:  0.8219,    Adjusted R-squared:  0.8218
F-statistic: 1.614e+04 on 36 and 125936 DF,  p-value: < 2.2e-16
```

R output for stepwise regression

Using stepwise regression, we end up with only 35 variables in the model. All the variables above are significant (at 5% significance level) with the exception of *service*.

This model has an adjusted R² value of 0.8218.

Hence, this model is much better than the previous model (which only has 5 variables).

Logistic Regression

As we are doing binary classification, it would be more appropriate to fit a logistic regression model to our features.

Firstly, we fit a logistic model to our top 5 features.

```
logit_model_1 <- glm(formula = xAttack ~ dst_bytes + dst_host_srv_count +
  logged_in + dst_host_srv_serror_rate + same_srv_rate,
  family = binomial(link = "logit"), data = train_set)
summary(logit_model_1)

Call:
glm(formula = xAttack ~ dst_bytes + dst_host_srv_count + logged_in +
  dst_host_srv_serror_rate + same_srv_rate, family = binomial(link = "logit"),
  data = train_set)

Deviance Residuals:
    Min      1Q  Median      3Q     Max 
-3.11675 -0.26265 -0.17592  0.09908  2.89011 

Coefficients:
              Estimate Std. Error z value Pr(>|z|)    
(Intercept)  2.279e+00 2.931e-02 77.74   <2e-16 ***
dst_bytes    1.013e-08 1.777e-08  0.57   0.569    
dst_host_srv_count -8.914e-03 1.163e-04 -76.63   <2e-16 ***
logged_in   -2.511e+00 2.712e-02 -92.59   <2e-16 ***
dst_host_srv_serror_rate 3.407e+00 7.116e-02  47.88   <2e-16 ***
same_srv_rate -1.656e+00 3.445e-02 -48.07   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 174033  on 125972  degrees of freedom
Residual deviance: 59177  on 125967  degrees of freedom
AIC: 59189

Number of Fisher Scoring iterations: 7
```

R output for logistic regression model with top 5 variables

From the output, only the variable *dst_bytes* is not significant in the final logistic model.

J48 Decision Tree

As Weka will split the data within the program itself, we will combine the train and test sets into a single csv file. As the original split was 15% test set, 85% train set, we will use these settings for Weka's train-test split.

Recall that our top 3 chosen features are: *dst_host_srv_count*, *logged_in*, *dst_host_srv_serror_rate*. Using Weka's J48 decision tree, the following tree was obtained.

```
== Run information ==

Scheme:      weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:    chosen_features_weka
Instances:   148516
Attributes:  4
              logged_in
              dst_host_srv_count
              dst_host_srv_serror_rate
              xAttack
Test mode:   split 85.0% train, remainder test

== Classifier model (full training set) ==

J48 pruned tree
-----

logged_in <= 0
|   dst_host_srv_serror_rate <= 0.33
|   |   dst_host_srv_count <= 165
|   |   |   dst_host_srv_count <= 20
|   |   |   |   dst_host_srv_serror_rate <= 0.02: malicious (23098.0/4571.0)
|   |   |   |   dst_host_srv_serror_rate > 0.02
|   |   |   |   |   dst_host_srv_serror_rate <= 0.06: malicious (32.0/14.0)
|   |   |   |   |   dst_host_srv_serror_rate > 0.06: benign (221.0/74.0)
|   |   |   dst_host_srv_count > 20
|   |   |   |   dst_host_srv_serror_rate <= 0.14
|   |   |   |   dst_host_srv_serror_rate <= 0
|   |   |   |   |   dst_host_srv_count <= 88: malicious (7226.0/2732.0)
|   |   |   |   |   dst_host_srv_count > 88
|   |   |   |   |   |   dst_host_srv_count <= 91
|   |   |   |   |   |   |   dst_host_srv_count <= 89: malicious (72.0/27.0)
|   |   |   |   |   |   |   dst_host_srv_count > 89: benign (209.0/89.0)
|   |   |   |   |   dst_host_srv_count > 91
|   |   |   |   |   |   dst_host_srv_count <= 130: malicious (2196.0/861.0)
|   |   |   |   |   |   dst_host_srv_count > 130
|   |   |   |   |   |   |   dst_host_srv_count <= 131: benign (101.0/25.0)
|   |   |   |   |   |   |   dst_host_srv_count > 131: malicious (1457.0/509.0)
|   |   |   dst_host_srv_serror_rate > 0
|   |   |   |   dst_host_srv_count <= 96
|   |   |   |   |   dst_host_srv_count <= 27: benign (69.0/16.0)
|   |   |   |   |   dst_host_srv_count > 27
|   |   |   |   |   |   dst_host_srv_serror_rate <= 0.01: benign (76.0/34.0)
|   |   |   |   |   |   dst_host_srv_serror_rate > 0.01: malicious (230.0/70.0)
|   |   |   |   dst_host_srv_count > 96
|   |   |   |   |   dst_host_srv_serror_rate <= 0.02: benign (134.0/4.0)
```

```

| | | | | dst_host_srv_serror_rate > 0.02
| | | | | | dst_host_srv_count <= 133: benign (7.0)
| | | | | | dst_host_srv_count > 133
| | | | | | | dst_host_srv_serror_rate <= 0.03: benign (5.0/2.0)
| | | | | | | dst_host_srv_serror_rate > 0.03: malicious (16.0)
| | | | dst_host_srv_serror_rate > 0.14: malicious (83.0/1.0)
dst_host_srv_count > 165
| dst_host_srv_serror_rate <= 0.02
| | dst_host_srv_count <= 254
| | | dst_host_srv_serror_rate <= 0: benign (10034.0/1853.0)
| | | dst_host_srv_serror_rate > 0
| | | | dst_host_srv_count <= 250: benign (75.0/5.0)
| | | | dst_host_srv_count > 250: malicious (71.0/15.0)
| | | dst_host_srv_count > 254: benign (5793.0/1642.0)
| | dst_host_srv_serror_rate > 0.02: malicious (180.0/5.0)
| dst_host_srv_serror_rate > 0.33: malicious (37310.0/61.0)
logged_in > 0
| dst_host_srv_count <= 254
| | dst_host_srv_count <= 252
| | | dst_host_srv_count <= 203: benign (17747.0/2875.0)
| | | dst_host_srv_count > 203
| | | | dst_host_srv_serror_rate <= 0.01: benign (3781.0/803.0)
| | | | dst_host_srv_serror_rate > 0.01
| | | | | dst_host_srv_serror_rate <= 0.02
| | | | | | dst_host_srv_count <= 244: malicious (85.0)
| | | | | | dst_host_srv_count > 244: benign (8.0)
| | | | | dst_host_srv_serror_rate > 0.02: benign (18.0/1.0)
| dst_host_srv_count > 252
| | dst_host_srv_serror_rate <= 0: benign (529.0/172.0)
| | dst_host_srv_serror_rate > 0
| | | dst_host_srv_serror_rate <= 0.02: malicious (133.0/18.0)
| | | dst_host_srv_serror_rate > 0.02
| | | | dst_host_srv_count <= 253: benign (5.0)
| | | | dst_host_srv_count > 253: malicious (4.0)
dst_host_srv_count > 254
| dst_host_srv_serror_rate <= 0: benign (35028.0/405.0)
| dst_host_srv_serror_rate > 0
| | dst_host_srv_serror_rate <= 0.02: benign (2449.0/141.0)
| | dst_host_srv_serror_rate > 0.02
| | | dst_host_srv_serror_rate <= 0.03: malicious (7.0)
| | | dst_host_srv_serror_rate > 0.03
| | | | dst_host_srv_serror_rate <= 0.04: benign (20.0)
| | | | dst_host_srv_serror_rate > 0.04: malicious (7.0/1.0)

```

Number of Leaves : 37

Size of the tree : 73

Time taken to build model: 0.67 seconds

```
== Evaluation on test split ==
```

Time taken to test model on test split: **0.01** seconds

```
== Summary ==
```

Correctly Classified Instances	19754	88.6744 %
Incorrectly Classified Instances	2523	11.3256 %
Kappa statistic	0.7733	
Mean absolute error	0.1716	
Root mean squared error	0.2926	
Relative absolute error	34.3719 %	
Root relative squared error	58.561 %	
Total Number of Instances	22277	

```
== Detailed Accuracy By Class ==
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
benign	0.878	0.104	0.902	0.878	0.890	0.774	0.949	0.940	benign
malicious	0.896	0.122	0.871	0.896	0.883	0.774	0.949	0.938	malicious
Weighted Avg.	0.887	0.113	0.887	0.887	0.887	0.774	0.949	0.939	

```
== Confusion Matrix ==
```

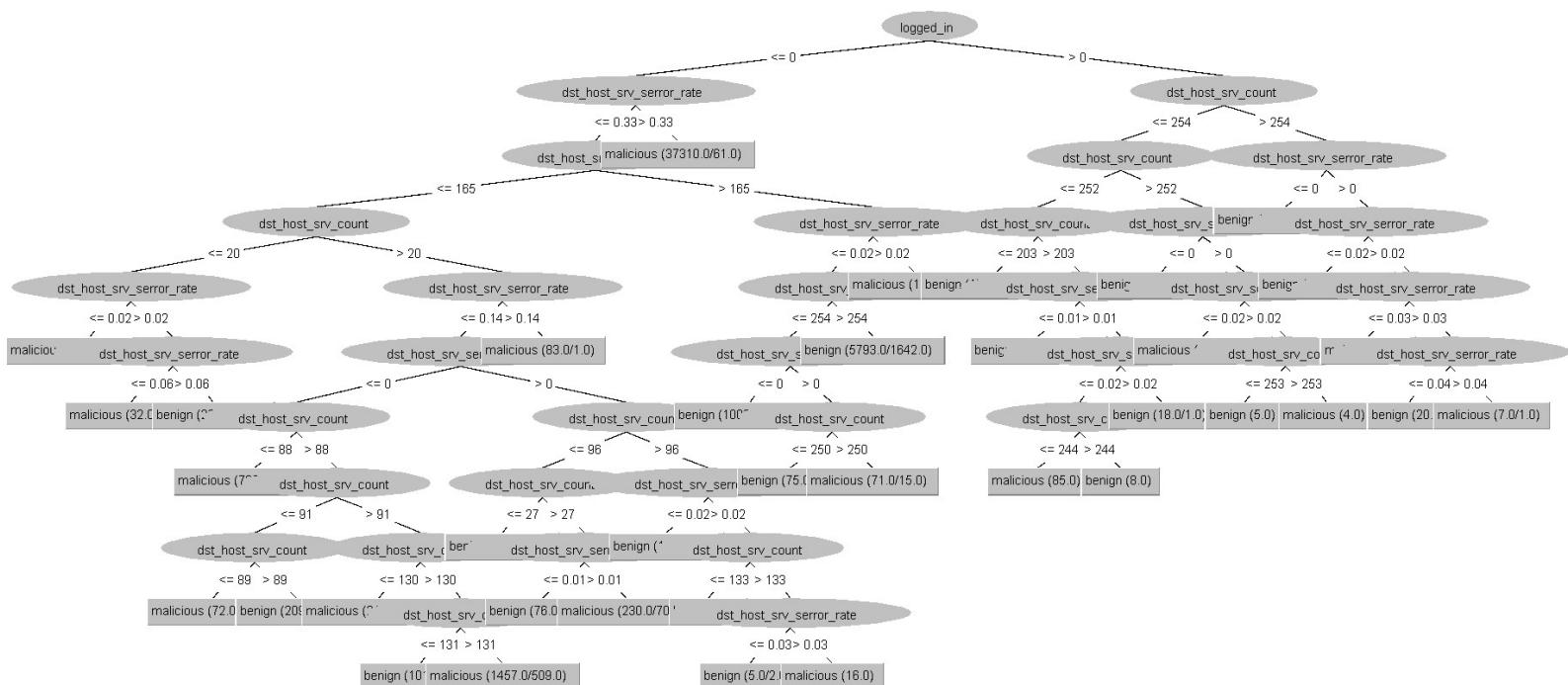
a	b	<-- classified as
10194	1410	a = benign
1113	9560	b = malicious

Weka output for decision tree

Above is Weka's print out for the J48 decision tree.

Weka's decision tree has accuracy 88.67%, recall 89.6% and precision 87.1%.

Weka's decision tree can be visualized below.



Manual Implementation of Decision Tree

Using Python, the decision tree is implemented in a series of if-else statements. The code can be found in Appendix A.

Using our tree, we split the dataset to obtain a 15% test set. Subsequently, we obtained our classification metrics on this test set (Appendix B).

Accuracy: 0.9012883242806482

Precision: 0.8722222222222222

Recall: 0.9232623156271088

Decision tree metrics

	Predicted Benign	Predicted Malicious
Actual Benign	10501	796
Actual Malicious	1403	9577

Confusion matrix for decision tree

The manually implemented decision tree has similar metrics to Weka's decision tree. The differences could be due to how the test sets were obtained for each implementation. Both Weka and our implementation obtained the test set randomly.

Naive Bayes

Using Weka's Naive Bayes classifier, the following model was obtained.

```
== Run information ==

Scheme:      weka.classifiers.bayes.NaiveBayes
Relation:    chosen_features_weka
Instances:   148516
Attributes:  4
              logged_in
              dst_host_srv_count
              dst_host_srv_serror_rate
              xAttack
Test mode:   split 85.0% train, remainder test

== Classifier model (full training set) ==

Naive Bayes Classifier

          Class
Attribute      benign  malicious
                  (0.52)  (0.48)
=====
logged_in
  mean           0.7165  0.0646
  std. dev.     0.4507  0.2458
  weight sum    77053   71463
  precision     1       1

dst_host_srv_count
  mean           193.6543 39.4691
  std. dev.     91.1676 66.4627
  weight sum    77053   71463
  precision     1       1

dst_host_srv_serror_rate
  mean           0.0056  0.5162
  std. dev.     0.0545  0.4953
  weight sum    77053   71463
  precision     0.01   0.01

Time taken to build model: 0.12 seconds

== Evaluation on test split ==

Time taken to test model on test split: 0.04 seconds
```

== Summary ==

Correctly Classified Instances	19283	86.5601 %
Incorrectly Classified Instances	2994	13.4399 %
Kappa statistic	0.7297	
Mean absolute error	0.1662	
Root mean squared error	0.3353	
Relative absolute error	33.2913 %	
Root relative squared error	67.1249 %	
Total Number of Instances	22277	

== Detailed Accuracy By Class ==

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.913	0.186	0.842	0.913	0.876	0.733	0.941	0.940	benign
	0.814	0.087	0.896	0.814	0.853	0.733	0.940	0.936	malicious
Weighted Avg.	0.866	0.139	0.868	0.866	0.865	0.733	0.940	0.938	

== Confusion Matrix ==

a	b	<-- classified as
10596	1008	a = benign
1986	8687	b = malicious

Weka output for Naive Bayes model

Weka's Naive Bayes has accuracy 86.56%, recall 81.4% and precision 89.6%.

Manual Implementation of Naive Bayes

Using the summary statistics obtained from Weka, a manual implementation of Naive Bayes was possible (Appendix C).

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

↑ ↑
Likelihood Class Prior Probability
↓ ↓
Posterior Probability Predictor Prior Probability

Our posterior probability [P(classification | data)] is what we are interested in.

Our likelihood [P(data | classification)] and prior [P(classification)] can be obtained from Weka's summary statistics. We can calculate the likelihood by assuming a Gaussian distribution.

Define X1: *logged_in*, X2: *dst_host_srv_count*, X3: *dst_host_srv_serror_rate*

The pseudo code is as follows:

1. Obtain prior class probabilities, mean and standard deviation from Weka.
2. For each row,
 - a. Obtain $P(\text{class}=0 | X1, X2, X3) \approx P(X1|\text{class}=0) * P(X2|\text{class}=0) * P(X3|\text{class}=0) * P(\text{class}=0)$.
 - b. Obtain $P(\text{class}=1 | X1, X2, X3) \approx P(X1|\text{class}=1) * P(X2|\text{class}=1) * P(X3|\text{class}=1) * P(\text{class}=1)$.
 - c. Return predicted class as class with higher probability. Note that we are ignoring the denominator as they are the same for both posterior probabilities.

Subsequently, we obtained our classification metrics on this test set (Appendix D).

Accuracy: 0.885217937783364

Precision: 0.8756969813497404

Recall: 0.8781451846139015

Naive Bayes metrics

	Predicted Benign	Predicted Malicious
Actual Benign	10611	1264
Actual Malicious	1293	9109

Confusion matrix for Naive Bayes

The manually implemented Naive Bayes algorithm has similar metrics to Weka's implementation. The differences could be due to how the test sets were obtained for each implementation. Both Weka and our implementation obtained the test set randomly.

Multi Layer Perceptron

Using Weka's Multi Layer Perceptron, the following model was obtained.

```
== Run information ==

Scheme:      weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
Relation:    chosen_features_weka
Instances:   148516
Attributes:  4
              logged_in
              dst_host_srv_count
              dst_host_srv_serror_rate
              xAttack
Test mode:   split 85.0% train, remainder test

== Classifier model (full training set) ==

Sigmoid Node 0
  Inputs  Weights
  Threshold  2.9386866537532312
  Node 2  -3.5497231078100104
  Node 3  -8.42281183346259
Sigmoid Node 1
  Inputs  Weights
  Threshold  -2.9386866537532312
  Node 2  3.5497231078100104
  Node 3  8.422811833462587
Sigmoid Node 2
  Inputs  Weights
  Threshold  -6.178574965522042
  Attrib logged_in  -2.1528845325622834
  Attrib dst_host_srv_count  -2.78772303592114
  Attrib dst_host_srv_serror_rate  -5.280187278884462
Sigmoid Node 3
  Inputs  Weights
  Threshold  -8.399289908252548
  Attrib logged_in  -42.89292787638223
  Attrib dst_host_srv_count  17.919887555684497
  Attrib dst_host_srv_serror_rate  56.427914833209606
Class benign
  Input
  Node 0
Class malicious
  Input
  Node 1

Time taken to build model: 17.41 seconds

== Evaluation on test split ==

Time taken to test model on test split: 0.03 seconds
```

== Summary ==

Correctly Classified Instances	19631	88.1223 %
Incorrectly Classified Instances	2646	11.8777 %
Kappa statistic	0.7621	
Mean absolute error	0.1818	
Root mean squared error	0.3018	
Relative absolute error	36.4195 %	
Root relative squared error	60.4104 %	
Total Number of Instances	22277	

== Detailed Accuracy By Class ==

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.881	0.118	0.890	0.881	0.885	0.762	0.945	0.941	benign
	0.882	0.119	0.872	0.882	0.877	0.762	0.945	0.944	malicious
Weighted Avg.	0.881	0.119	0.881	0.881	0.881	0.762	0.945	0.942	

== Confusion Matrix ==

a	b	<-- classified as
10222	1382	a = benign
1264	9409	b = malicious

Weka output for Multi Layer Perceptron

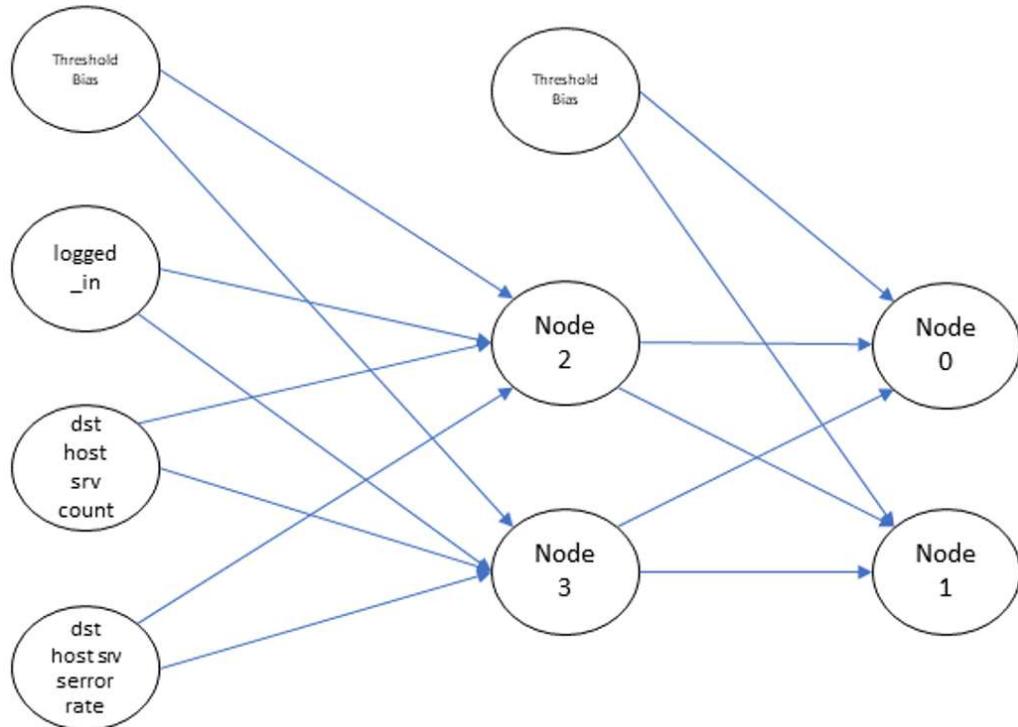


Figure 11. Neural network architecture

The neural network can be visualized as above.

Manual Implementation of Multi Layer Perceptron

To manually implement the MLP, we extract the obtained weights from Weka and create our own feed-forward network. Our activation function uses the logistic function (Appendix E).

Note that we had to normalize our features before putting them into the feed-forward network.

Subsequently, we obtained our classification metrics on this test set (Appendix F).

```
Accuracy: 0.7097005880504557
Precision: 0.749537251272559
Recall: 0.6009646600500881
```

MLP metrics

	Predicted Benign	Predicted Malicious
Actual Benign	9331	4302
Actual Malicious	2165	6479

Confusion matrix for MLP

The manually implemented MLP algorithm has similar metrics to Weka's implementation. The differences could be due to how the test sets were obtained for each implementation. Both Weka and our implementation obtained the test set randomly.

Recurrent Neural Network

Using Keras and TensorFlow, a simple RNN network was created for this binary classification problem.

2 model architectures were tested; a simple model and a complex model. The architectures are as follows:

```
Model: "sequential_9"
```

Layer (type)	Output Shape	Param #
<hr/>		
lstm_17 (LSTM)	(None, 1, 100)	41600
<hr/>		
dropout_6 (Dropout)	(None, 1, 100)	0
<hr/>		
lstm_18 (LSTM)	(None, 100)	80400
<hr/>		
dropout_7 (Dropout)	(None, 100)	0
<hr/>		
dense_4 (Dense)	(None, 32)	3232
<hr/>		
dropout_8 (Dropout)	(None, 32)	0
<hr/>		
dense_5 (Dense)	(None, 1)	33
<hr/>		
Total params: 125,265		
Trainable params: 125,265		
Non-trainable params: 0		

Figure 12. Architecture of complex RNN model

```
Model: "sequential_10"
```

Layer (type)	Output Shape	Param #
<hr/>		
lstm_19 (LSTM)	(None, 1, 100)	41600
<hr/>		
dense_6 (Dense)	(None, 1, 1)	101
<hr/>		
Total params: 41,701		
Trainable params: 41,701		
Non-trainable params: 0		

Figure 13. Architecture of simple RNN model

To see the notebook with the implementation code, please see Appendix G. To see the raw Python code, see Appendix H.

The models are trained using 50 epochs and batch size of 64.

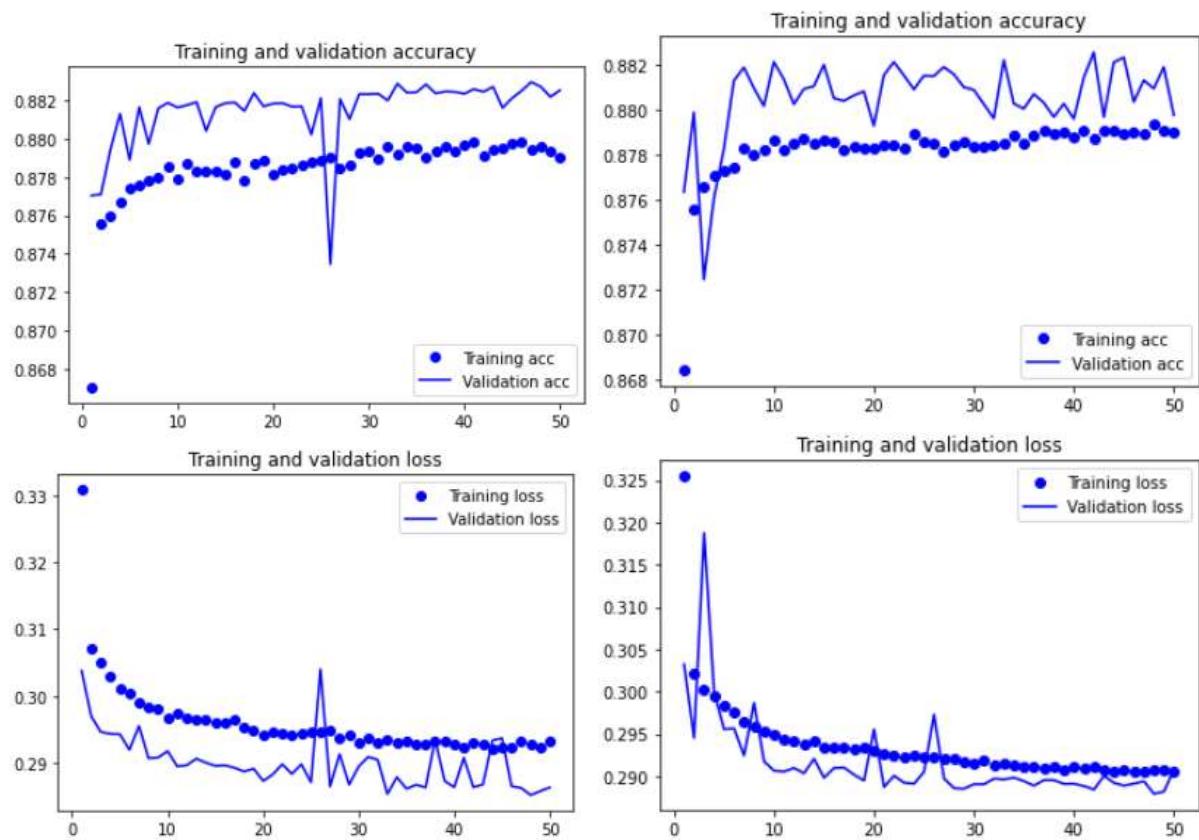


Figure 14. Loss and accuracy graphs for complex RNN model

Figure 15. Loss and accuracy graphs for simple RNN model

	Predicted Benign	Predicted Malicious
Actual Benign	22360	2749
Actual Malicious	2995	20907

Table 6. Confusion matrix for simple RNN model

	Predicted Benign	Predicted Malicious
Actual Benign	22174	2572
Actual Malicious	3181	21084

Table 7. Confusion matrix for complex RNN model

	Simple RNN Model	Complex RNN Model
Accuracy	0.882801819999592	0.8826181877537695
Precision	0.8746966781022508	0.8689058314444673
Recall	0.8837926952992898	0.891274940818397

Table 8. Classification metrics for RNN models

From the loss and accuracy graphs (figures 14 and 15), we can see that both accuracy and loss graphs have converged quickly and stabilized.

The error rates have decreased exponentially to a very low value over various epochs.

The classification metrics (obtained on the test set) are also quite good. The RNN classification metrics are comparable to the various models implemented on Weka.

References

- [1] Botes, F., Leenen, L. and De La Harpe, R. (2017). Ant Colony Induced Decision Trees for Intrusion Detection. In: 16th European Conference on Cyber Warfare and Security. ACPI (June 12, 2017), pp.74-83. Retrieved from <https://github.com/InitRoot/NSLKDD-Dataset>
- [2] NSL-KDD dataset. Canadian Institute for Cybersecuriuty, University of New Brunswick. Retrieved from <https://www.unb.ca/cic/datasets/nsl.html>

Appendix

A. Decision Tree Function

```
def classification_tree(logged_in, dst_host_srv_count, dst_host_srv_serror_rate):
    # Rename variables for easier splitting
    log = logged_in
    count = dst_host_srv_count
    error = dst_host_srv_serror_rate
    rate = dst_host_srv_serror_rate

    # Do splitting logic
    if log > 0:
        if count <= 254:
            if count <= 252:
                if count <= 203:
                    return('benign')
                else: #count > 203
                    if error <= 0.01:
                        return('benign')
                    elif error > 0.01:
                        if error <= 0.02:
                            if count <= 244:
                                return('malicious')
                            elif count > 244:
                                return('benign')
                            elif error > 0.02:
                                return('benign')
                        elif count > 252:
                            if error <= 0:
                                return('benign')
                            elif error > 0:
                                if error <= 0.02:
                                    return('malicious')
                                elif error > 0.02:
                                    if count <= 253:
                                        return('benign')
                                    elif count > 253:
                                        return('malicious')
                            elif count > 254:
                                if error <= 0:
                                    return('benign')
                                elif error > 0:
                                    if error <= 0.02:
                                        return('benign')
                                    elif error > 0.02:
                                        if error <= 0.03:
                                            return('malicious')
                                        elif error > 0.03:
                                            if error <= 0.04:
                                                return('benign')
                                            elif error > 0.04:
                                                return('malicious')

            if log <= 0:
                if error <= 0.33:
```

```

if count <= 165:
    if count <= 20:
        if error <= 0.02:
            return('malicious')
        elif error > 0.02:
            if error <= 0.02:
                return('malicious')
            elif error > 0.02:
                if error <= 0.06:
                    return('malicious')
                elif error > 0.06:
                    return('benign')
    elif count > 20:
        if error <= 0.14:
            if error <= 0:
                if count <= 88:
                    return('malicious')
                elif count > 88:
                    if count <= 91:
                        if count <= 89:
                            return('malicious')
                        elif count > 89:
                            return('benign')
                    if count > 91:
                        if count <= 130:
                            return('malicious')
                        if count > 130:
                            if count <= 131:
                                return('benign')
                            elif count > 131:
                                return('malicious')
            if error > 0:
                if count <= 96:
                    if count <= 27:
                        return('benign')
                    elif count > 27:
                        if error <= 0.01:
                            return('benign')
                        elif error > 0.01:
                            return('benign')
                if count > 96:
                    if error <= 0.02:
                        return('benign')
                    elif error > 0.02:
                        if count <= 133 :
                            return('benign')
                        elif count > 133:
                            if error <= 0.03:
                                return('benign')
                            elif error > 0.03:
                                return('malicious')
            elif error > 0.14:
                return('malicious')
    elif count > 165:
        if error <= 0.02:
            if count <= 254:
                if rate <= 0:
                    return('benign')
                elif rate > 0:

```

```
    if count <= 250:
        return('benign')
    elif count > 250:
        return('malicious')
    elif count > 254:
        return('benign')
    elif error > 0.02:
        return('malicious')
    elif error > 0.33:
        return('malicious')

    return('No match found')
```

B. Classification Metrics of Decision Tree

```
# Training set is 85% of dataset, test set is 15% of dataset
tree_test_set = dataset.head(int(0.15 * len(dataset)))
true_positive, true_negative, false_positive, false_negative = 0, 0, 0, 0

for index, row in tree_test_set.iterrows():
    logged_in = row['logged_in']
    srv_count = row['dst_host_srv_count']
    error_rate = row['dst_host_srv_serror_rate']
    actual_label = row['xAttack']

    predicted_label = classification_tree(logged_in = logged_in, dst_host_srv_count = srv_count,
                                            dst_host_srv_serror_rate = error_rate)

    # Error checking
    if predicted_label not in ['malicious', 'benign']:
        print(predicted_label)

    # Calculate confusion matrix values
    if actual_label == 'malicious' and predicted_label == 'malicious':
        true_positive += 1
    elif actual_label == 'benign' and predicted_label == 'benign':
        true_negative += 1
    elif actual_label == 'malicious' and predicted_label == 'benign':
        false_negative += 1
    elif actual_label == 'benign' and predicted_label == 'malicious':
        false_positive += 1
    else:
        print('error')

# Print confusion matrix
data = {'Predicted Benign': [true_negative, false_positive],
        'Predicted Malicious': [false_negative, true_positive]}
df = pd.DataFrame(data, index = ['Actual Benign',
                                  'Actual Malicious',
                                  ])
])
```

	Predicted Benign	Predicted Malicious
Actual Benign	10501	796
Actual Malicious	1403	9577

```
# Print other classification metrics
print('Accuracy:', (true_positive + true_negative) / len(tree_test_set))
print('Precision:', (true_positive) / (true_positive + false_positive))
print('Recall:', (true_positive) / (true_positive + false_negative))

Accuracy: 0.9012883242806482
Precision: 0.8722222222222222
Recall: 0.9232623156271088
```

C. Naive Bayes Function

```
from math import sqrt
from math import pi
from math import exp

# Read in test set data
dataset = pd.read_csv('chosen_features_weka.csv')
nb_test_set = dataset.head(int(0.15 * len(dataset)))

# Calculate the Gaussian probability distribution function for x
def calculate_probability(x, mean, stdev):
    exponent = exp(-((x-mean)**2 / (2 * stdev**2)))
    return (1 / (sqrt(2 * pi) * stdev)) * exponent

# Define statistics from Weka
weka_stats = {
    'class_0_prior': 0.52, 'class_1_prior': 0.48,

    'class_0_logged_in_mean': 0.7165, 'class_0_logged_in_sd': 0.4507,
    'class_1_logged_in_mean': 0.0646, 'class_1_logged_in_sd': 0.2458,

    'class_0_count_mean': 193.6543, 'class_0_count_sd': 91.1676,
    'class_1_count_mean': 39.4691, 'class_1_count_sd': 66.4627,

    'class_0_error_mean': 0.0056, 'class_0_error_sd': 0.0545,
    'class_1_error_mean': 0.5162, 'class_1_error_sd': 0.4953
}

def nb_pred(test_row, weka_stats):

    # Calculate class 0 probability
    class_0_prob = weka_stats['class_0_prior']
    class_0_prob *= calculate_probability(test_row['logged_in'], weka_stats['class_0_logged_in_mean'],
                                          weka_stats['class_0_logged_in_sd'])

    class_0_prob *= calculate_probability(test_row['dst_host_srv_count'],
                                           weka_stats['class_0_count_mean'],
                                           weka_stats['class_0_count_sd'])

    class_0_prob *= calculate_probability(test_row['dst_host_srv_error_rate'],
                                           weka_stats['class_0_error_mean'],
                                           weka_stats['class_0_error_sd'])

    # Calculate class 1 probability
    class_1_prob = weka_stats['class_1_prior']
    class_1_prob *= calculate_probability(test_row['logged_in'], weka_stats['class_1_logged_in_mean'],
                                          weka_stats['class_1_logged_in_sd'])

    class_1_prob *= calculate_probability(test_row['dst_host_srv_count'],
                                           weka_stats['class_1_count_mean'],
                                           weka_stats['class_1_count_sd'])

    class_1_prob *= calculate_probability(test_row['dst_host_srv_error_rate'],
                                           weka_stats['class_1_error_mean'],
                                           weka_stats['class_1_error_sd'])

    # Return classification
    if class_0_prob > class_1_prob:
        return('benign')
    else:
        return('malicious')
```

D. Classification Metrics of Naive Bayes

```
true_positive, true_negative, false_positive, false_negative = 0, 0, 0, 0

for index, row in nb_test_set.iterrows():
    actual_label = row['xAttack']
    predicted_label = nb_pred(test_row = row, weka_stats = weka_stats)

    # Error checking
    if predicted_label not in ['malicious', 'benign']:
        print(predicted_label)

    # Calculate confusion matrix values
    if actual_label == 'malicious' and predicted_label == 'malicious':
        true_positive += 1
    elif actual_label == 'benign' and predicted_label == 'benign':
        true_negative += 1
    elif actual_label == 'malicious' and predicted_label == 'benign':
        false_negative += 1
    elif actual_label == 'benign' and predicted_label == 'malicious':
        false_positive += 1
    else:
        print('error')

# Print confusion matrix
data = {'Predicted Benign':[true_negative, false_positive],
        'Predicted Malicious':[false_negative, true_positive]}
df = pd.DataFrame(data, index =['Actual Benign',
                                'Actual Malicious',
                                ])
df
```

Predicted Benign Predicted Malicious

Actual Benign	10611	1264
Actual Malicious	1293	9109

```
# Print other classification metrics
print('Accuracy:', (true_positive + true_negative) / len(nb_test_set))
print('Precision:', (true_positive) / (true_positive + false_positive))
print('Recall:', (true_positive) / (true_positive + false_negative))
Accuracy: 0.885217937783364
Precision: 0.8756969813497404
Recall: 0.8781451846139015
```

E. Multi Layer Perceptron Function

```
from scipy.stats import logistic

def logistic_function(x):
    return (logistic.cdf(x))

def mlp_classifier(row):
    log_in, count, error = row['logged_in'], row['dst_host_srv_count'], row['dst_host_srv_serror_rate']

    node_2 = -6.178574965522042 + (-2.1528845325622834 * log_in) + (-2.78772303592114 * count) +
(-5.28018727884462 * error)
    node_3 = -8.399289908252548 + (-42.89292787638223 * log_in) + (17.919887555684497 * count) +
(56.427914833209606 * error)

    node_2 = logistic_function(node_2)
    node_3 = logistic_function(node_3)

    node_0 = 2.9386866537532312 + (-3.5497231078100104 * node_2) + (-8.42281183346259 * node_3)
    node_1 = -2.9386866537532312 + (3.5497231078100104 * node_2) + (8.422811833462587 * node_3)

    node_0 = logistic_function(node_0)
    node_1 = logistic_function(node_1)

    if node_0 > node_1:
        return('benign')
    else:
        return('malicious')
```

F. Classification Metrics of Multi Layer Perceptron

```
# Read in test set data
dataset = pd.read_csv('chosen_features_weka.csv')

# Normalize dataset
from sklearn import preprocessing
scaler = preprocessing.MinMaxScaler()
dataset[['logged_in', 'dst_host_srv_count', 'dst_host_srv_serror_rate']] =
scaler.fit_transform(dataset[['logged_in', 'dst_host_srv_count', 'dst_host_srv_serror_rate']])

# Extract test set
mlp_test_set = dataset.sample(int(0.15 * len(dataset)))

true_positive, true_negative, false_positive, false_negative = 0, 0, 0, 0

for index, row in mlp_test_set.iterrows():
    actual_label = row['xAttack']
    predicted_label = mlp_classifier(row)

    # Error checking
    if predicted_label not in ['malicious', 'benign']:
        print(predicted_label)

    # Calculate confusion matrix values
    if actual_label == 'malicious' and predicted_label == 'malicious':
        true_positive += 1
    elif actual_label == 'benign' and predicted_label == 'benign':
        true_negative += 1
    elif actual_label == 'malicious' and predicted_label == 'benign':
        false_negative += 1
    elif actual_label == 'benign' and predicted_label == 'malicious':
        false_positive += 1
    else:
        print('error')

# Print confusion matrix
data = {'Predicted Benign':[true_negative, false_positive],
        'Predicted Malicious':[false_negative, true_positive]}
df = pd.DataFrame(data, index =[ 'Actual Benign',
                                 'Actual Malicious',
                                 ])
df
```

	Predicted Benign	Predicted Malicious
Actual Benign	9331	4302
Actual Malicious	2165	6479

```
# Print other classification metrics
print('Accuracy:', (true_positive + true_negative) / len(mlp_test_set))
print('Precision:', (true_positive) / (true_positive + false_positive))
print('Recall:', (true_positive) / (true_positive + false_negative))

Accuracy: 0.7097005880504557
Precision: 0.749537251272559
Recall: 0.6009646600500881
```

G. Recurrent Neural Network Notebook

10/13/21, 6:18 PM

STL1 - P1 - LSTM.ipynb - Colaboratory

```
1 # Upload chosen_features_weka.csv
2 from google.colab import files
3 files.upload()

Choose Files chosen_feat...s_weka.csv
• chosen_features_weka.csv(application/vnd.ms-excel) - 2782898 bytes, last modified: 10/12/2021 - 1
done
Saving chosen_features_weka.csv to chosen_features_weka.csv
{'chosen_features_weka.csv': b'logged_in,dst_host_srv_count,dst_host_srv_error_rate,'

1 # Define a function to print accuracy graphs
2 def print_graphs(history):
3     acc = history.history['accuracy']
4     val_acc = history.history['val_accuracy']
5     loss = history.history['loss']
6     val_loss = history.history['val_loss']
7     epochs = range(1, len(acc) + 1)
8
9     plt.plot(epochs, acc, 'bo', label='Training acc')
10    plt.plot(epochs, val_acc, 'b', label='Validation acc')
11    plt.title('Training and validation accuracy')
12    plt.legend()
13
14    plt.figure()
15
16    plt.plot(epochs, loss, 'bo', label='Training loss')
17    plt.plot(epochs, val_loss, 'b', label='Validation loss')
18    plt.title('Training and validation loss')
19    plt.legend()
20
21    plt.show()
```

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Load dataset
6 dataset = pd.read_csv('chosen_features_weka.csv')
7
8 # Change labels to binary
9 dataset['xAttack'] = np.where(dataset['xAttack'] == 'benign', 0, 1)
10 dataset.head()
```

```

logged_in dst_host_srv_count dst_host_srv_serror_rate xAttack

1 # Split into test and train set
2 from sklearn.model_selection import train_test_split
3 X, y = dataset[['logged_in', 'dst_host_srv_count', 'dst_host_srv_serror_rate']], dataset
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=
5
6 # Reshape dataset
7 X_train = X_train.values.reshape(-1,·1,·3)
8 X_test = X_test.values.reshape(-1, 1, 3)
9 y_train = y_train.values.reshape(-1, 1, 1)
10 y_test = y_test.values.reshape(-1, 1, 1)

```

```

1 import tensorflow
2 from tensorflow.keras.models import Sequential
3 from tensorflow.keras.layers import Dense, Dropout, LSTM
4
5 model_complex = tensorflow.keras.models.Sequential()
6 model_complex.add(tensorflow.keras.layers.LSTM(100, input_shape=(1, 3), activation='relu'))
7 model_complex.add(Dropout(0.2))
8
9 model_complex.add(LSTM(100))
10 model_complex.add(Dropout(0.2))
11 model_complex.add(Dense(32, activation='relu'))
12 model_complex.add(Dropout(0.2))
13
14
15 model_complex.add(tensorflow.keras.layers.Dense(1, activation='sigmoid'))
16 model_complex.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

```

```
1 model_complex.summary()
```

Model: "sequential_11"

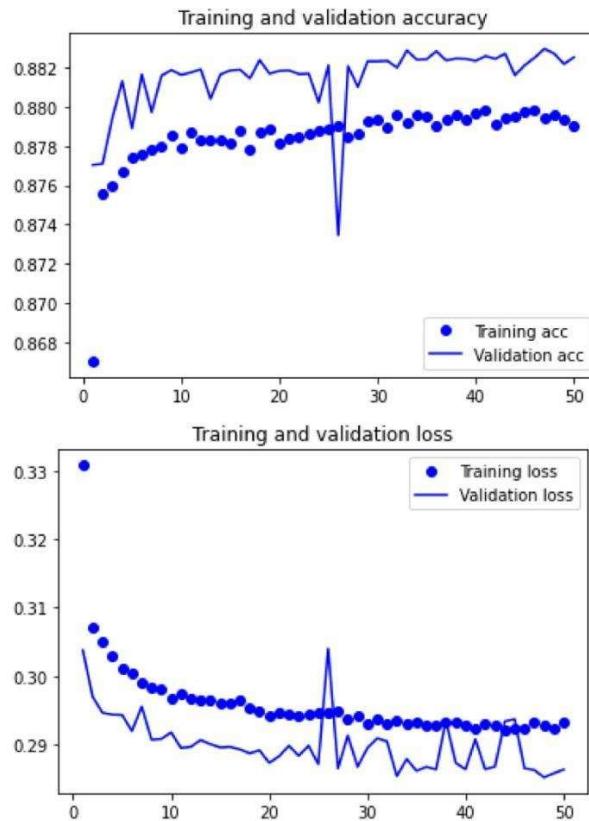
Layer (type)	Output Shape	Param #
<hr/>		
lstm_20 (LSTM)	(None, 1, 100)	41600
<hr/>		
dropout_9 (Dropout)	(None, 1, 100)	0
<hr/>		
lstm_21 (LSTM)	(None, 100)	80400
<hr/>		
dropout_10 (Dropout)	(None, 100)	0
<hr/>		
dense_7 (Dense)	(None, 32)	3232
<hr/>		
dropout_11 (Dropout)	(None, 32)	0
<hr/>		
dense_8 (Dense)	(None, 1)	33
<hr/>		
Total params: 125,265		
Trainable params: 125,265		
Non-trainable params: 0		

10/13/21, 6:18 PM

STL1 - P1 - LSTM.ipynb - Colaboratory

```
1 history = model_complex.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50
  Epoch 22/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2947 - accuracy: 0.2947
  Epoch 23/50
  1555/1555 [=====] - 13s 8ms/step - loss: 0.2945 - accuracy: 0.2945
  Epoch 24/50
  1555/1555 [=====] - 13s 8ms/step - loss: 0.2946 - accuracy: 0.2946
  Epoch 25/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2938 - accuracy: 0.2938
  Epoch 26/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2945 - accuracy: 0.2945
  Epoch 27/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2941 - accuracy: 0.2941
  Epoch 28/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2951 - accuracy: 0.2951
  Epoch 29/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2940 - accuracy: 0.2940
  Epoch 30/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2944 - accuracy: 0.2944
  Epoch 31/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2936 - accuracy: 0.2936
  Epoch 32/50
  1555/1555 [=====] - 13s 8ms/step - loss: 0.2941 - accuracy: 0.2941
  Epoch 33/50
  1555/1555 [=====] - 13s 8ms/step - loss: 0.2932 - accuracy: 0.2932
  Epoch 34/50
  1555/1555 [=====] - 13s 8ms/step - loss: 0.2937 - accuracy: 0.2937
  Epoch 35/50
  1555/1555 [=====] - 13s 8ms/step - loss: 0.2934 - accuracy: 0.2934
  Epoch 36/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2938 - accuracy: 0.2938
  Epoch 37/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2935 - accuracy: 0.2935
  Epoch 38/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2926 - accuracy: 0.2926
  Epoch 39/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2936 - accuracy: 0.2936
  Epoch 40/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2931 - accuracy: 0.2931
  Epoch 41/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2931 - accuracy: 0.2931
  Epoch 42/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2928 - accuracy: 0.2928
  Epoch 43/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2922 - accuracy: 0.2922
  Epoch 44/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2927 - accuracy: 0.2927
  Epoch 45/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2927 - accuracy: 0.2927
  Epoch 46/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2923 - accuracy: 0.2923
  Epoch 47/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2926 - accuracy: 0.2926
  Epoch 48/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2926 - accuracy: 0.2926
  Epoch 49/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2923 - accuracy: 0.2923
  Epoch 50/50
  1555/1555 [=====] - 12s 8ms/step - loss: 0.2929 - accuracy: 0.2929
```

```
1 print_graphs(history)
```



```
1 model_simple = tensorflow.keras.models.Sequential()
2 model_simple.add(tensorflow.keras.layers.LSTM(100, input_shape=(1, 3), activation='relu')
3 model_simple.add(tensorflow.keras.layers.Dense(1, activation='sigmoid'))
4 model_simple.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
5 model_simple.summary()
```

Model: "sequential_12"

Layer (type)	Output Shape	Param #
<hr/>		
lstm_22 (LSTM)	(None, 1, 100)	41600
<hr/>		
dense_9 (Dense)	(None, 1, 1)	101
<hr/>		

Total params: 41,701
 Trainable params: 41,701
 Non-trainable params: 0

```
1 history = model_simple.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5
```

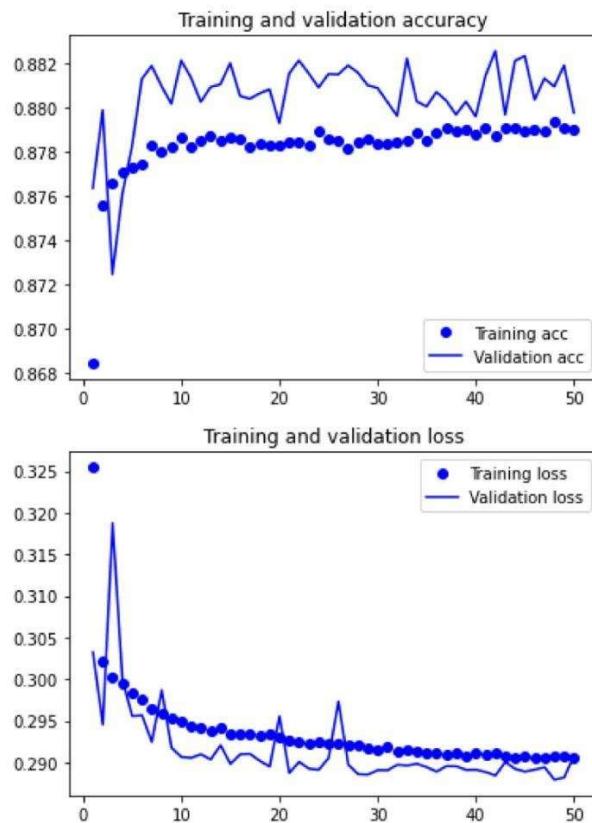
Epoch 1/50
 1555/1555 [=====] - 8s 4ms/step - loss: 0.3354 - accuracy

10/13/21, 6:18 PM

STL1 - P1 - LSTM.ipynb - Colaboratory

```
Epoch 2/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.3037 - accuracy
Epoch 3/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.3007 - accuracy
Epoch 4/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.3005 - accuracy
Epoch 5/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2996 - accuracy
Epoch 6/50
1555/1555 [=====] - 7s 4ms/step - loss: 0.2987 - accuracy
Epoch 7/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2972 - accuracy
Epoch 8/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2971 - accuracy
Epoch 9/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2967 - accuracy
Epoch 10/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2956 - accuracy
Epoch 11/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2956 - accuracy
Epoch 12/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2950 - accuracy
Epoch 13/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2948 - accuracy
Epoch 14/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2944 - accuracy
Epoch 15/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2945 - accuracy
Epoch 16/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2942 - accuracy
Epoch 17/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2936 - accuracy
Epoch 18/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2933 - accuracy
Epoch 19/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2931 - accuracy
Epoch 20/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2930 - accuracy
Epoch 21/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2927 - accuracy
Epoch 22/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2922 - accuracy
Epoch 23/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2921 - accuracy
Epoch 24/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2921 - accuracy
Epoch 25/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2921 - accuracy
Epoch 26/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2919 - accuracy
Epoch 27/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2915 - accuracy
Epoch 28/50
1555/1555 [=====] - 7s 4ms/step - loss: 0.2916 - accuracy
Epoch 29/50
1555/1555 [=====] - 6s 4ms/step - loss: 0.2914 - accuracy
```

```
1 print_graphs(history)
```



▼ Get classification metrics on test set

```

1 # Get metrics for model_simple
2 true_positive, true_negative, false_positive, false_negative = 0, 0, 0, 0
3
4 for index in range(len(X_test)):
5     test_row = X_test[index,:]
6     test_row = test_row.reshape(-1, 1, 3)
7     predicted_prob = model_simple.predict(test_row)[0][0][0]
8
9     predicted_label = None
10    if predicted_prob >= 0.50:
11        predicted_label = 1
12    else:
13        predicted_label = 0
14
15    actual_label = y_test[index][0][0]
16
17    if actual_label == 1 and predicted_label == 1:
18        true_positive += 1
19    elif actual_label == 0 and predicted_label == 0:
20        true_negative += 1
21    elif actual_label == 1 and predicted_label == 0:
22        false_negative += 1
23    elif actual_label == 0 and predicted_label == 1:
24        false_positive += 1

```

10/13/21, 6:18 PM

STL1 - P1 - LSTM.ipynb - Colaboratory

```
24     false_positive += 1
25 else:
26     print('error')
27
28 if index % 1000 == 0:
29     print(index)
30
31 # Print confusion matrix
32 data = {'Predicted Benign':[true_negative, false_positive],
33         'Predicted Malicious':[false_negative, true_positive]}
34 df = pd.DataFrame(data, index =[ 'Actual Benign',
35                         'Actual Malicious',
36                         ])
37 df
```

```
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
1 # Print other classification metrics for model_simple
2 print('Accuracy:', (true_positive + true_negative) / len(X_test))
3 print('Precision:', (true_positive) / (true_positive + false_positive))
4 print('Recall:', (true_positive) / (true_positive+false_negative))

Accuracy: 0.882801819999592
Precision: 0.8746966781022508
Recall: 0.8837926952992898
25000
1
0.06705746
30000
1 # Get metrics for model_complex
2 true_positive, true_negative, false_positive, false_negative = 0, 0, 0, 0
3
4 for index in range(len(X_test)):
5     test_row = X_test[index,]
6     test_row = test_row.reshape(-1, 1, 3)
7     predicted_prob = model_complex.predict(test_row)[0][0]
8
9     predicted_label = None
10    if predicted_prob >= 0.50:
11        predicted_label = 1
12    else:
13        predicted_label = 0
14
15    actual_label = y_test[index][0][0]
16
17    if actual_label == 1 and predicted_label == 1:
18        true_positive += 1
19    elif actual_label == 0 and predicted_label == 0:
20        true_negative += 1
21    elif actual_label == 1 and predicted_label == 0:
22        false_negative += 1
23    elif actual_label == 0 and predicted_label == 1:
24        false_positive += 1
25    else:
26        print('error')
27
```

10/13/21, 6:18 PM

STL1 - P1 - LSTM.ipynb - Colaboratory

```
28 if index % 1000 == 0:  
29     print(index)  
30  
31 # Print confusion matrix  
32 data = {'Predicted Benign':[true_negative, false_positive],  
33         'Predicted Malicious':[false_negative, true_positive]}  
34 df = pd.DataFrame(data, index =[ 'Actual Benign',  
35                     'Actual Malicious',  
36                     ])  
37 df
```

```
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

1 # Print other classification metrics for model_complex
2 print('Accuracy:', (true_positive + true_negative) / len(X_test))
3 print('Precision:', (true_positive) / (true_positive + false_positive))
4 print('Recall:', (true_positive) / (true_positive + false_negative))

↳ Accuracy: 0.8826181877537695
Precision: 0.8689058314444673
Recall: 0.891274940818397

21000
22000
23000
24000
25000
26000
27000
28000
29000
30000
31000
32000
33000
34000
35000
36000
37000
38000
39000
40000
41000
42000
43000
44000
```

✓ 0s completed at 5:49 PM



H. Recurrent Neural Network Code

```
# -*- coding: utf-8 -*-
"""STL1 - P1 - LSTM.ipynb

Automatically generated by Colaboratory.

"""

# Upload chosen_features_weka.csv
from google.colab import files
files.upload()

# Define a function to print accuracy graphs
def print_graphs(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)

    plt.plot(epochs, acc, 'bo', label='Training acc')
    plt.plot(epochs, val_acc, 'b', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()

    plt.figure()

    plt.plot(epochs, loss, 'bo', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()

    plt.show()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load dataset
dataset = pd.read_csv('chosen_features_weka.csv')

# Change labels to binary
dataset['xAttack'] = np.where(dataset['xAttack'] == 'benign', 0, 1)
dataset.head()

# Split into test and train set
from sklearn.model_selection import train_test_split
X, y = dataset[['logged_in', 'dst_host_srv_count', 'dst_host_srv_error_rate']], dataset[['xAttack']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

# Reshape dataset
X_train = X_train.values.reshape(-1, 1, 3)
X_test = X_test.values.reshape(-1, 1, 3)
y_train = y_train.values.reshape(-1, 1, 1)
y_test = y_test.values.reshape(-1, 1, 1)

import tensorflow
```

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM

model_complex = tensorflow.keras.models.Sequential()
model_complex.add(tensorflow.keras.layers.LSTM(100, input_shape=(1, 3), activation='relu',
return_sequences=True))
model_complex.add(Dropout(0.2))

model_complex.add(LSTM(100))
model_complex.add(Dropout(0.2))
model_complex.add(Dense(32, activation='relu'))
model_complex.add(Dropout(0.2))

model_complex.add(tensorflow.keras.layers.Dense(1, activation='sigmoid'))
model_complex.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model_complex.summary()

history = model_complex.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50,
batch_size=64)

print_graphs(history)

model_simple = tensorflow.keras.models.Sequential()
model_simple.add(tensorflow.keras.layers.LSTM(100, input_shape=(1, 3), activation='relu',
return_sequences=True))
model_simple.add(tensorflow.keras.layers.Dense(1, activation='sigmoid'))
model_simple.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model_simple.summary()

history = model_simple.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, batch_size=64)

print_graphs(history)

"""## Get classification metrics on test set"""

# Get metrics for model_simple
true_positive, true_negative, false_positive, false_negative = 0, 0, 0, 0

for index in range(len(X_test)):
    test_row = X_test[index,]
    test_row = test_row.reshape(-1, 1, 3)
    predicted_prob = model_simple.predict(test_row)[0][0][0]

    predicted_label = None
    if predicted_prob >= 0.50:
        predicted_label = 1
    else:
        predicted_label = 0

    actual_label = y_test[index][0][0]

    if actual_label == 1 and predicted_label == 1:
        true_positive += 1
    elif actual_label == 0 and predicted_label == 0:
        true_negative += 1
    elif actual_label == 1 and predicted_label == 0:
        false_negative += 1

```

```

    elif actual_label == 0 and predicted_label == 1:
        false_positive += 1
    else:
        print('error')

    if index % 1000 == 0:
        print(index)

# Print confusion matrix
data = {'Predicted Benign':[true_negative, false_positive],
        'Predicted Malicious':[false_negative, true_positive]}
df = pd.DataFrame(data, index =[ 'Actual Benign',
                                'Actual Malicious',
                                ])
df

# Print other classification metrics for model_simple
print('Accuracy:', (true_positive + true_negative) / len(X_test))
print('Precision:', (true_positive) / (true_positive + false_positive))
print('Recall:', (true_positive) / (true_positive + false_negative))

# Get metrics for model_complex
true_positive, true_negative, false_positive, false_negative = 0, 0, 0, 0

for index in range(len(X_test)):
    test_row = X_test[index,:]
    test_row = test_row.reshape(-1, 1, 3)
    predicted_prob = model_complex.predict(test_row)[0][0]

    predicted_label = None
    if predicted_prob >= 0.50:
        predicted_label = 1
    else:
        predicted_label = 0

    actual_label = y_test[index][0][0]

    if actual_label == 1 and predicted_label == 1:
        true_positive += 1
    elif actual_label == 0 and predicted_label == 0:
        true_negative += 1
    elif actual_label == 1 and predicted_label == 0:
        false_negative += 1
    elif actual_label == 0 and predicted_label == 1:
        false_positive += 1
    else:
        print('error')

    if index % 1000 == 0:
        print(index)

# Print confusion matrix
data = {'Predicted Benign':[true_negative, false_positive],
        'Predicted Malicious':[false_negative, true_positive]}
df = pd.DataFrame(data, index =[ 'Actual Benign',
                                'Actual Malicious',
                                ])

```

```
df

# Print other classification metrics for model_complex
print('Accuracy:', (true_positive + true_negative) / len(X_test))
print('Precision:', (true_positive) / (true_positive + false_positive))
print('Recall:', (true_positive) / (true_positive + false_negative))
```