SC1015 A137
Mini Project

Team 9

Josiah Yong (U2221072K)
Yashver Shori(U2220850E)
Anish Gholap (U2221963J)

About the Project

- Growing presence of technology
- Importance of network security
- Need for Network Intrusion Detection System
- Cyber Attacks Increased by 125% through 2021 - 2023

Problem Definition

What is the best machine-learning model for anomaly detection model in a network intrusion detection system?

Dataset Overview (KDD Cup 1999)

- Consists of a wide variety of intrusions simulated in a military network environment

- 41 features
    - Grouped into Basic, Content, Traffic and Traffic between Hosts

- Class Label: Normal or Anomalous

Data Cleaning

1. Checking for Missing Values and filling them if needed
    - To ensure dataset is complete and accurate

2. Checking for Duplicate Rows
    - May create bias

We check for missing values to ensure that the dataset is complete and accurate as it may lead to biased or inaccurate results and errors

```
In [8]: intrusion.isnull().values.any()
Out[8]: False
```

no NAN values in dataset

Next, we check for duplicate rows as it may also create bias

```
In [9]: print(f"Number of duplicate rows: {intrusion.duplicated().sum()}")

Number of duplicate rows: 0
```

Exploratory Data Analysis

Univariate Data Analysis

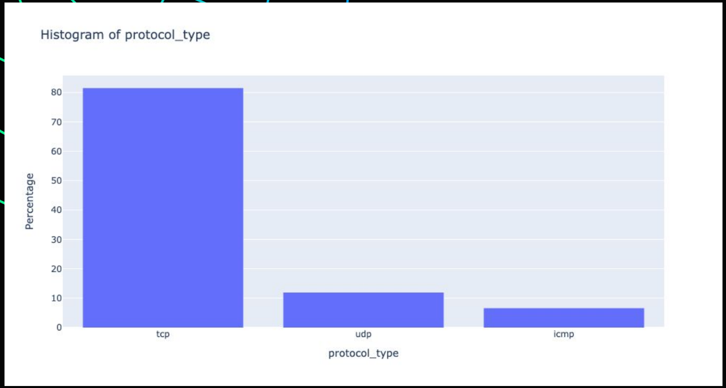
Exploring Categorical Features	Exploring Numerical Features
• Check the number of unique values of 'protocol_type' , 'service', 'flag’ • Plotting histogram to see visualize distribution of data for each variable • Correlation Matrix to analyse correlation between variables	• Visualizing numerical variables with univariate histograms • Statistical dispersion and variation

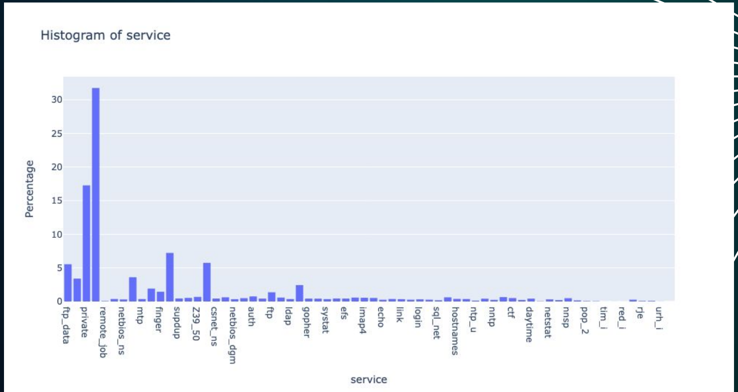
Categorical Features

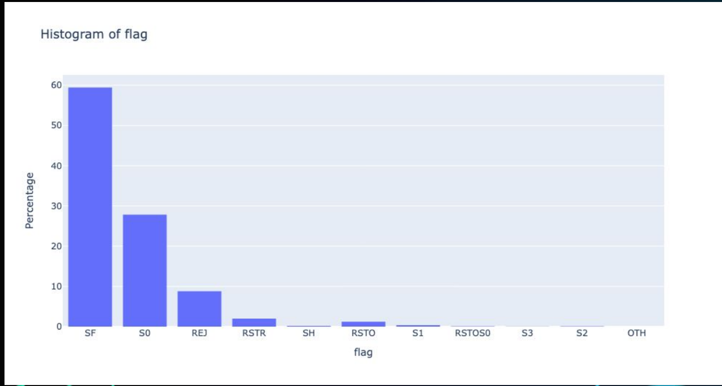
	protocol_type	service	flag
count	25192	25192	25192
unique	3	66	11
top	tcp	http	SF
freq	20526	8003	14973

```
=====Unique values of protocol_type=====
['tcp' 'udp' 'icmp']
Number of unique values: 3

=====Unique values of service=====
['ftp_data' 'other' 'private' 'http' 'remote_job' 'name' 'netbios_ns'
 'eco_i' 'mtp' 'telnet' 'finger' 'domain_u' 'supdup' 'uucp_path' 'Z39_50'
 'smtp' 'csnet_ns' 'uucp' 'netbios_dgm' 'urp_i' 'auth' 'domain' 'ftp'
 'bgp' 'ldap' 'ecr_i' 'gopher' 'vmnet' 'systat' 'http_443' 'efs' 'whois'
 'imap4' 'iso_tsap' 'echo' 'klogin' 'link' 'sunrpc' 'login' 'kshell'
 'sql_net' 'time' 'hostnames' 'exec' 'ntp_u' 'discard' 'nntp' 'courier'
 'ctf' 'ssh' 'daytime' 'shell' 'netstat' 'pop_3' 'nnsp' 'IRC' 'pop_2'
 'printer' 'tim_i' 'pm_dump' 'red_i' 'netbios_ssn' 'rje' 'X11' 'urh_i'
 'http_8001']
Number of unique values: 66

=====Unique values of flag=====
['SF' 'S0' 'REJ' 'RSTR' 'SH' 'RSTO' 'S1' 'RSTOS0' 'S3' 'S2' 'OTH']
Number of unique values: 11
```

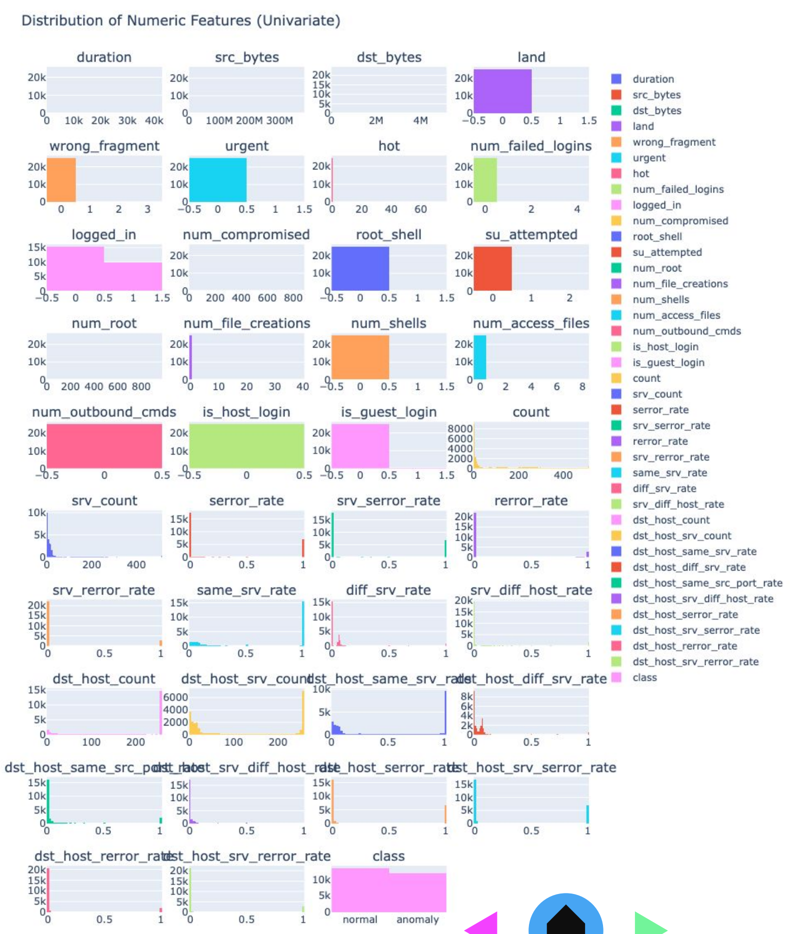
Categorical Variables



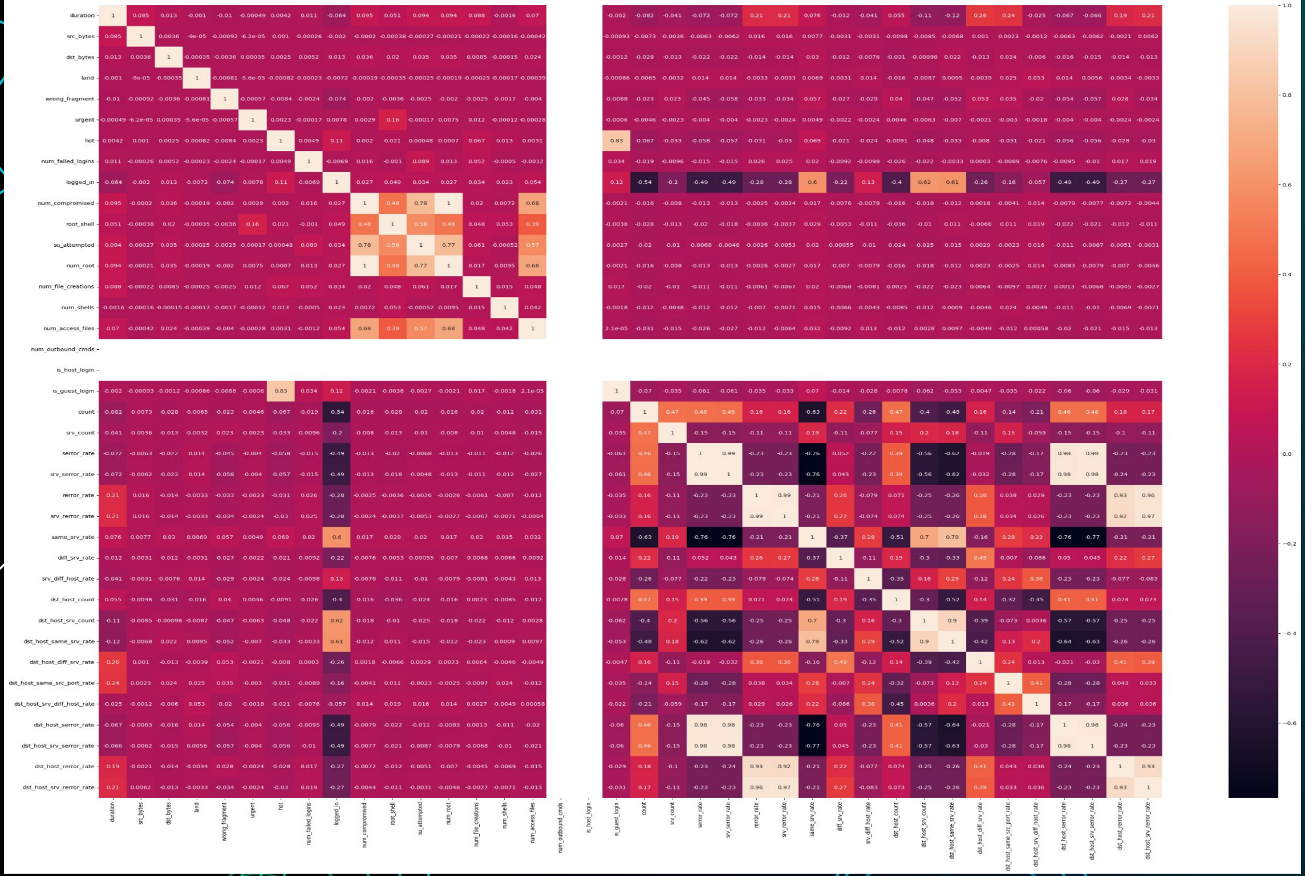




Numerical Variables



Correlation Matrix



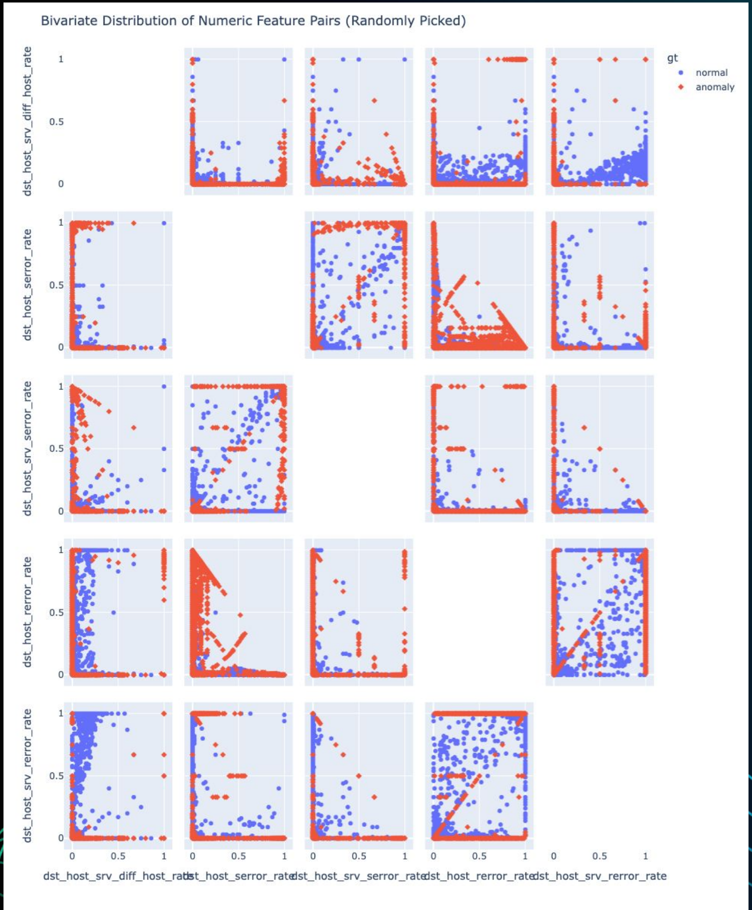
```
=====Statistical dispersion and variation=====
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	...	dst_h
Max Proportion	9.196570e-01	3.916323e-01	5.388218e-01	0.999921	0.991108	0.99996	0.979359	0.999087	0.605232	0.989203	...	
Variance	7.217581e+06	5.811983e+12	7.890897e+09	0.000079	0.067715	0.00004	4.640585	0.002063	0.238936	108.521223	...	

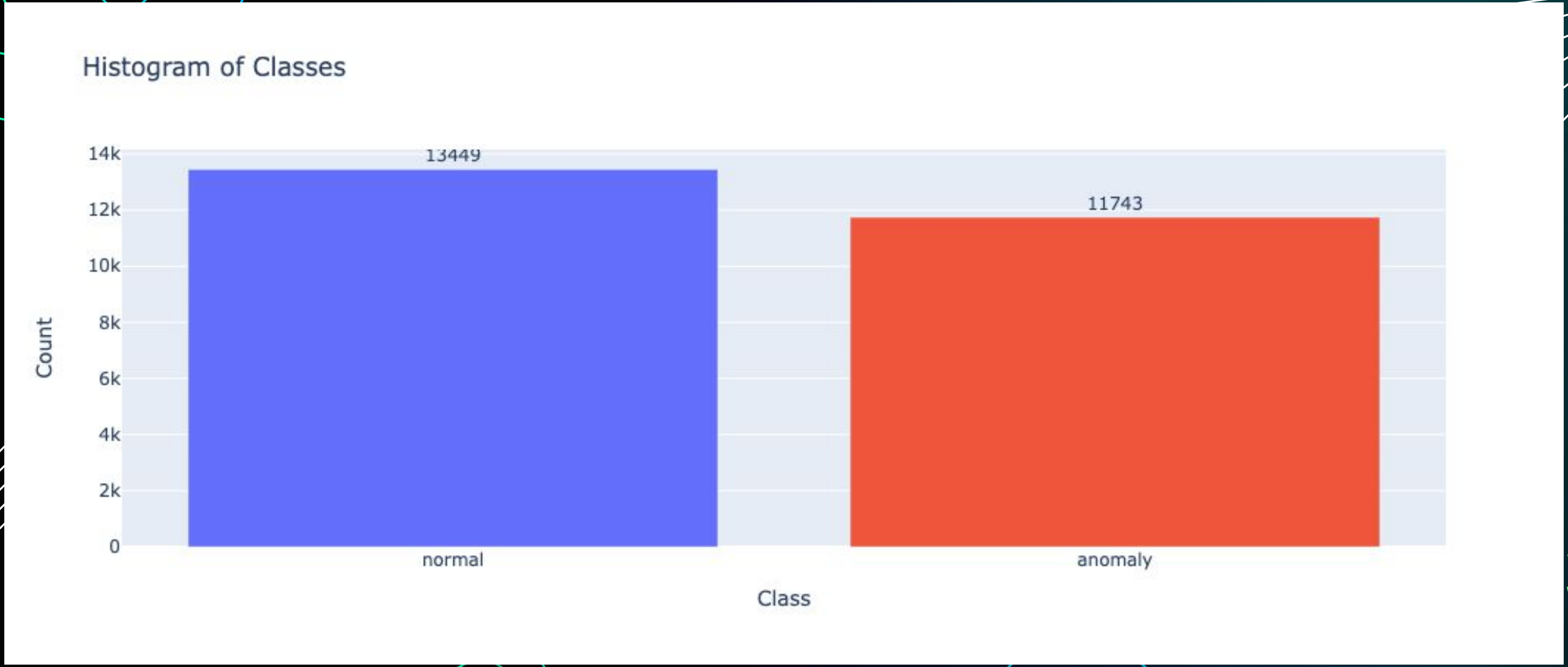
2 rows × 39 columns

```python
# Filter out features with high "max proportion" or low "variance"
disp_and_var_T = disp_and_var.T   # Take the transpose
features_remained = disp_and_var_T[(disp_and_var_T['Max Proportion'] < 0.99) & disp_and_var_T['Variance'] > 0.001].index.toli
intrusion = intrusion.loc[:, features_remained]
print(f"After filtering, there are {len(features_remained)} numeric features remained.")
```

Bivariate Data Analysis



Class-Distribution



MACHINE LEARNING

PREPARATORY WORK BEFORE WE BEGIN

Scaling Numerical Numbers

Standardize data before feeding it to the models

Encode Categorical Variables

Make them Compatible with Models

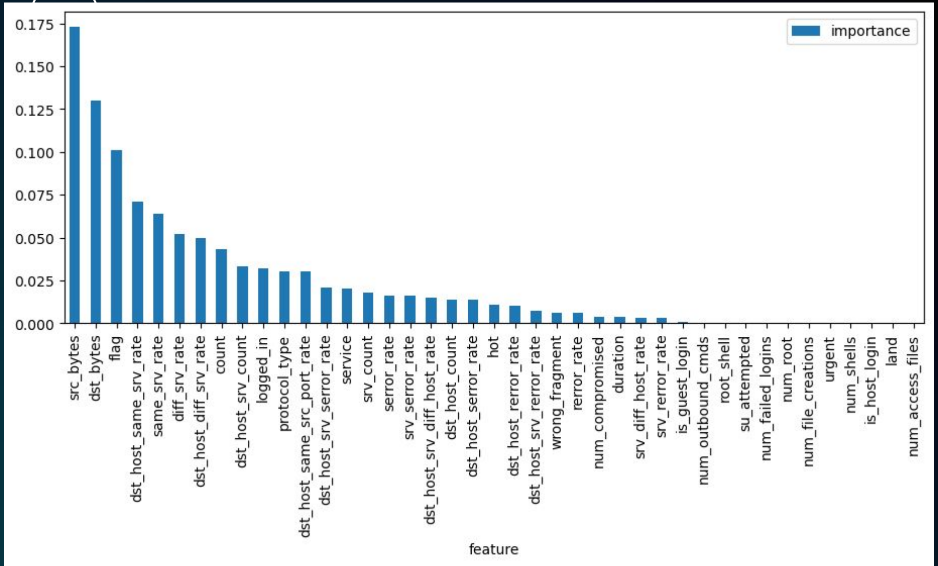
Feature Selection

Filter Data to prevent Over-Fitting

Dataset Partition

Separate data into Train and Test

Feature Selection



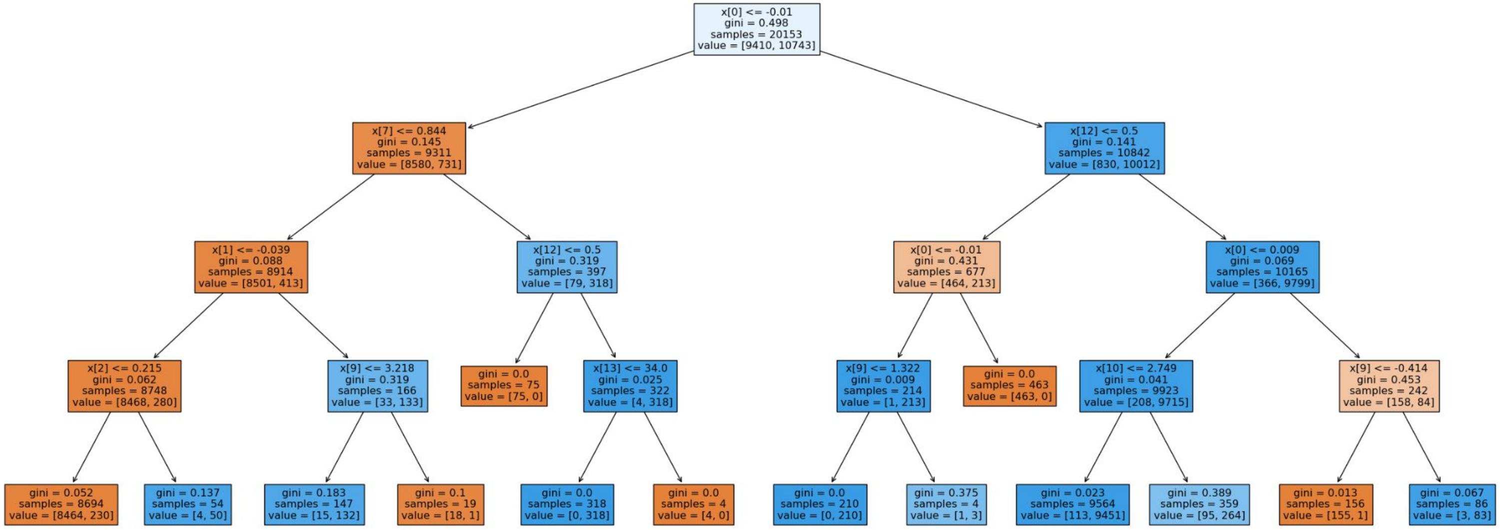
- Used Random Forest Classifier
- Identify which variables are important to train and test

Anomaly Detection Models

MODELS USED

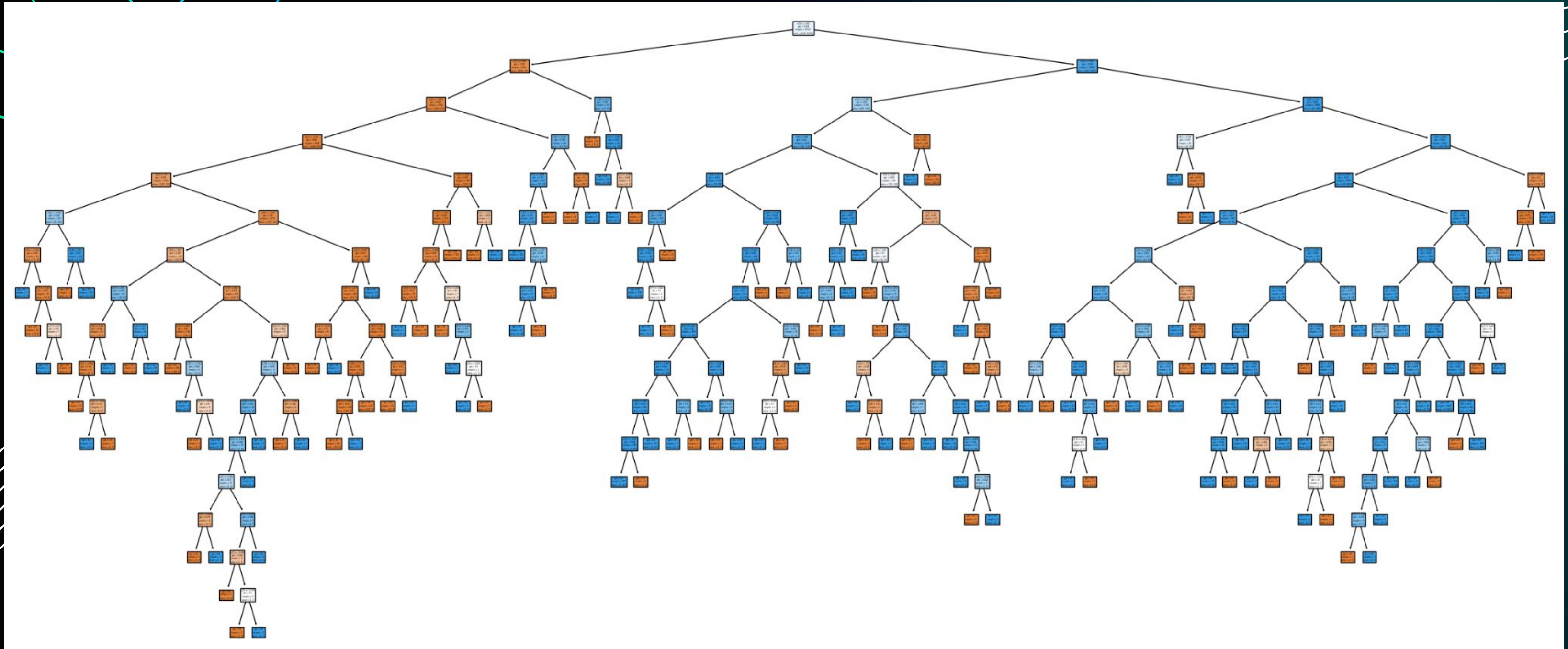
Decision Tree Classifier Model	K Neighbors Classifier Model	Logistics Regression
classify either ***normal*** or ***malicious*** by learning the patterns and behaviors of ***known*** intrusions.	finds the ***K*** nearest data points to a new ***intrusion*** and classifies it as either ***normal*** or ***malicious*** based on the ***majority***	***predicts*** the probability of a ***binary outcome*** based on one or more input variables.

Baseline : Decision-Tree




```
dt = DecisionTreeClassifier(max_depth=4) #take an arbitrary value first
```

Decision-Tree (After-Optimisation)



```
Best is trial 12 with value: 0.9964278626711649.
2023-04-21 22:24:23,662] Trial 29 finished with value: 0.994244889859099 and parameters: {'max_depth': 23, 'max_features':
```

K Nearest Neighbours (After-Optimisation)

```
KNN_model = KNeighborsClassifier(n_neighbors=study_KNN.best_trial.params['KNN_n_neighbors'])

# Measure the time taken to fit the model
start_time = time.time()
KNN_model.fit(X_train, Y_train)
fit_timeknn = time.time() - start_time

# Measure the accuracy of the model on the training and test set
KNN_train, KNN_test = KNN_model.score(X_train, Y_train), KNN_model.score(X_test, Y_test)

# Measure the time taken to generate predictions
start_time = time.time()
y_pred = KNN_model.predict(X_test)
predict_timeknn = time.time() - start_time

# Calculate the precision score
precisionKnn = precision_score(Y_test, y_pred, average='macro')
```

Logistics Regression (After-Optimisation)

```
lg_model = LogisticRegression(C=study_lg.best_trial.params['C'], penalty=study_lg.best_trial.params['penalty'])

# Measure the time taken to fit the model
start_time = time.time()
lg_model.fit(X_train, Y_train)
fit_timelg = time.time() - start_time

# Measure the accuracy of the model on the training and test set
lg_train, lg_test = lg_model.score(X_train, Y_train), lg_model.score(X_test, Y_test)

# Measure the time taken to generate predictions
start_time = time.time()
y_pred = lg_model.predict(X_test)
predict_timelg = time.time() - start_time

# Calculate the precision score
precisionlg = precision_score(Y_test, y_pred, average='macro')
```

Comparisons of Model

Model	Train Score	Test Score	Precision	Time to Fit	Time Predict	Optimisation time
KNN	0.984866	0.981544	0.981189	0.0910194	0.375084	0.479107
Logistic Regression	0.939513	0.934908	0.935269	0.124027	0.000999928	3.93516
Decision Tree	1	0.994642	0.994515	0.0500114	0.00100017	1.75239

Can we improve?

Conclusion

Learned to use different machine learning models and How to Optimise.

- K-Nearest Neighbors Classifier
- Random Forest Classifier
- Logistics Regression

Outcome of Project

- Makers could look into our results and improve their network intrusion systems

- Aim towards more secure networks

Data-driven Insights

- **Feature Selection necessary for testing on different networks**
- **For this Data, SRC_bytes most important feature**
- Decision-Tree most accurate
- KNN is the fastest

Thank You

Josiah Yong (U2221072K) , ,