Detecting Distributed Denial Of Service Attacks (DDoS) Using Machine Learning Models

Isha Singhal [*][†]

October 12, 2023

Abstract

The digital landscape of today's world is vulnerable to the widespread threat of Distributed Denial of Service (DDoS) attacks. These attacks have the potential to seriously damage businesses' finances and reputations by interfering with the availability of internet services. Traditional methods of DDoS mitigation, such as rule-based approaches, struggle to keep up with the evolving nature of attacks. In this paper, I have trained and tested several supervised machine learning algorithms for the identification of DDoS attacks to determine the most effective one. I explore the depths of DDoS, obtaining and adjusting a dataset-utilizing principal component analysis (PCA) to reduce the number of features in the model from 80 to 20 while preserving 90% variance in our dataset. By reducing unnecessary features, PCA allowed us to have higher model accuracy and training speed. Overall, the Random Forest model trained with PCA had the best results, obtaining 99.9% accuracy, precision, and recall. The proposed approach exhibits encouraging results, demonstrating its potential to improve DDoS attack detection and thus reinforce network security.

1 Introduction

Distributed Denial of Service (DDoS) attacks involve overloading a target system with an excessive amount of traffic, preventing it from responding to valid user requests. These types of attacks are executed from multiple computers or machines, making them both harder to detect and put an end to. They exploit the fundamental need for the availability of online services and can lead to severe operational, financial, and reputational consequences. Modern DDoS attacks are sophisticated and large-scale, and conventional protection methods like firewalls and intrusion detection systems often fall short in handling them. The attackers are continuously changing their methods to bypass the defense

[*]Advised by: Dr. Maria Konte

[†]Student at Northville High School in Northville, Michigan

mechanisms put in place to prevent DDoS attacks and researchers, in turn, change their approach to prevent new attacks.

There are several reasons why DDoS attacks are difficult to defend against. One reason is because of the scale and volume of the attack. These attacks involve a large volume of traffic that exceeds the target's capacity, making it difficult to filter out attack traffic from real traffic. The scale of attacks can go all the way up to hundreds of gigabits per second.

Another reason is that DDoS attacks are launched from a multitude of sources, usually through the use of botnets. Botnets are networks of computers infected with malware and are under the control of an attacker, making it difficult to identify and block the attacking sources effectively.

DDoS attacks often imitate normal user behavior, making it difficult to figure out whether the traffic is legitimate or not. Thus, it becomes hard to filter out malicious traffic without blocking real users. Lastly, DDoS attackers find and exploit weaknesses in network protocols, infrastructures, or applications, further amplifying the detrimental effects of their attacks.

Machine learning techniques have emerged as a potential solution to detect and prevent DDoS attacks. This is due to their ability to adapt to evolving attack patterns. I have attempted to use various supervised machine learning algorithms to detect DDoS attacks in this paper. Using a dataset with benign and DDoS attacks, I have trained and tested these models and then analyzed the results to figure out which model is most effective in determining and identifying DDoS attacks. The ultimate goal is to improve the detection of DDoS attacks in real-world real-time systems, where intrusion detection systems can be proactively utilized with the appropriate model for the detection and mitigation of these attacks.

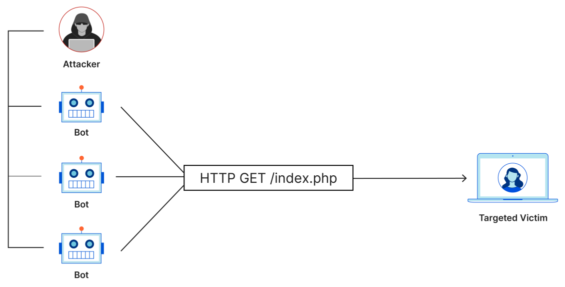


Figure 1: DDoS Attack [Clo23b]

2 Background

2.1 Types of DDoS Attacks

There are several kinds of DDoS Attacks [Imp23]. The most common are listed below:

2.1.1 Volumetric Attacks

This kind of DDoS attack floods the target's network with a large amount of traffic, consuming the available bandwidth and overwhelming the network.

2.1.2 Protocol Attacks

Another kind is a protocol attack, which exploits vulnerabilities in network protocols, such as TCP/IP, DNS, or ICMP, to disrupt the target's services or infrastructure.

2.1.3 Application Layer Attacks

Application layer DDoS attacks focus on certain applications or services and exploit vulnerabilities in the application layer to use up server resources or disrupt the functionality of the application.

2.1.4 Reflective Attacks

Lastly, reflective attacks are when the hacker spoofs the IP address of the source to make requests to servers that will respond with larger replies to the target's IP address, thus amplifying the attack traffic and making it harder to fight against.

2.2 What Harm Do DDoS Attacks Cause

The primary harm DDoS attacks cause is disrupting the availability of services or websites. By overwhelming the target's resources, actual users are unable to access the service or experience slow response times.

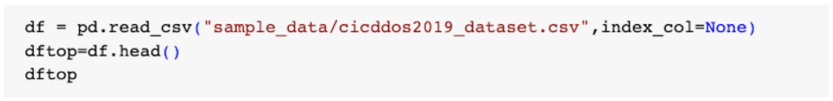
Businesses can experience financial loss from downtime due to DDoS attacks, especially if they rely heavily on online operations for their revenue. In the long term, frequent DDoS attacks can tarnish businesses' reputations, causing customers to lose trust in their ability to provide reliable services. For example, in June 2023, Microsoft's office suite—including Outlook and OneDrive—and cloud computing platform were hit by DDoS attacks, leading to serious service disruptions.

3 Dataset

3.1 What Does the Dataset Include

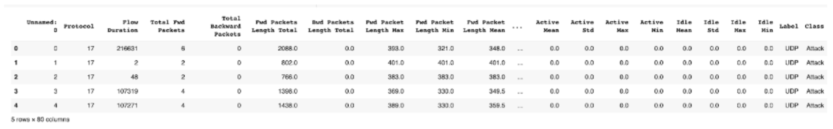
This dataset [Tal23], which shares its feature set with IDS2017, IDS2018, DoS2017, and DDoS 2019 CIC NIDS datasets, includes data from various kinds of DDoS attacks, such as DrDoS, UDP, LDAP, NetBIOS, MSSQL, and many others. It lists 80 features from over 400,000 DDoS attacks in order to provide a large dataset.

Using pandas, the data analysis library of Python, I read the dataset into a pandas data frame with its read_csv() method. In this function, index_col is an optional parameter that specifies the column(s) to be used as the index of the resulting data frame. By default, index_col is set to None, meaning that a new index will be created for the data frame.



```
df = pd.read_csv("sample_data/cicddos2019_dataset.csv",index_col=None)
dftop=df.head()
dftop
```

Figure 2: Reading Dataset Into Pandas Dataframe



	Unnamed: 0	Protocol	Flow Duration	Total Fwd Packets	Total Backward Packets	Fwd Packets Length Total	Bwd Packets Length Total	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	...	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label	Class
0	0	17	216631	6	0	2088.0	0.0	393.0	321.0	348.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	UDP	Attack
1	1	17	2	2	0	802.0	0.0	401.0	401.0	401.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	UDP	Attack
2	2	17	48	2	0	766.0	0.0	383.0	383.0	383.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	UDP	Attack
3	3	17	107319	4	0	1398.0	0.0	369.0	330.0	349.5	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	UDP	Attack
4	4	17	107271	4	0	1438.0	0.0	389.0	330.0	359.5	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	UDP	Attack

5 rows × 80 columns

Figure 3: Listing of Top Five Rows

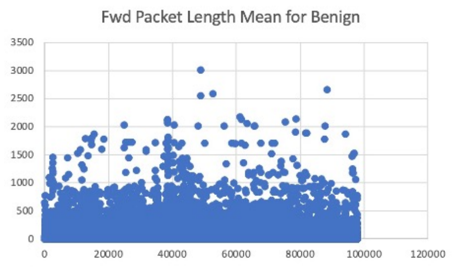
3.2 Analyzing the Dataset

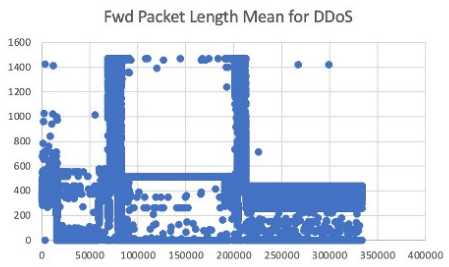
This dataset originally has 80 columns (features). Some of the important features and their significance is explained [Naj23] below:

Originally, the dataset contained 80 columns, including the flow duration, total forward and backward packets, and the average packet length. In this dataset, the label column is called "Class", whose value is either "Attack" or "Benign" and the remaining columns are the features. Some of the important features and their significance is explained below:

3.2.1 Forward packet length mean (fw_pkt_l_avg):

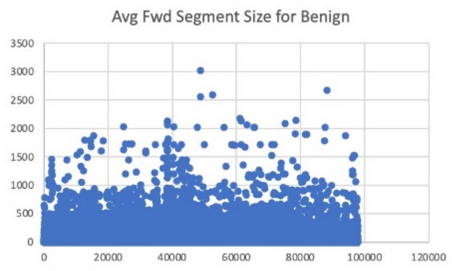
DDoS attacks generate a significantly higher volume of network traffic than normal traffic. By calculating the mean forward packet length, you can identify abnormal patterns where the average packet size deviates from the expected behavior seen in normal traffic.

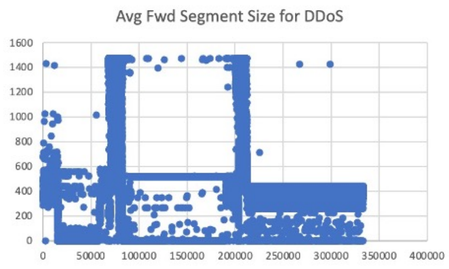




3.2.2 Forward segment size average (fwd_seg_avg):

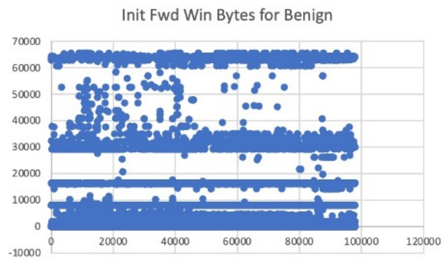
Some DDoS attacks involve packet fragmentation, where attackers split their payloads into smaller segments to evade detection. Analyzing the forward segment size average can help identify unusually small or fragmented packets that might be part of such attacks.

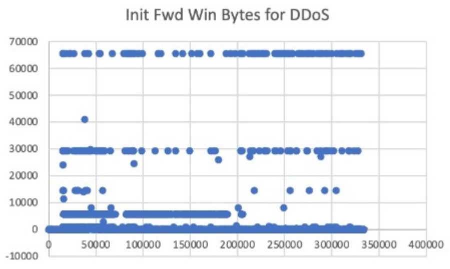




3.2.3 Initial forward window bytes (fw_win_byt):

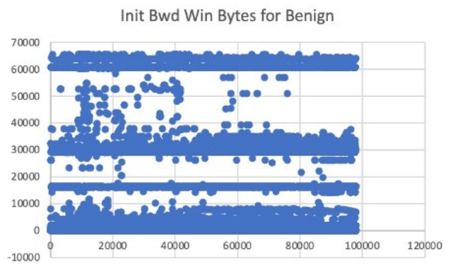
The Initial Forward Window Bytes metric refers to the total number of bytes sent from the client to the server during the initial connection establishment. Unusually large values can be indicative of an attack, as DDoS attacks often involve rapid and massive connection attempts to overwhelm the target server or network. Some DDoS attacks, such as SYN flood attacks, involve flooding the target server with a large number of connection requests. Monitoring the number of bytes sent in the initial window in the forward direction can help identify such flooding patterns and differentiate them from legitimate connection attempts.

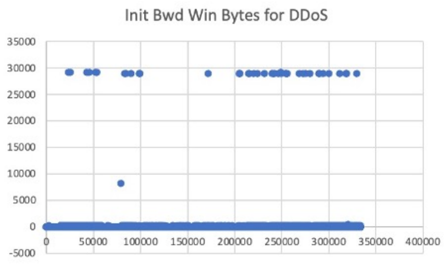




3.2.4 Initial backward window bytes (bw_win_byt):

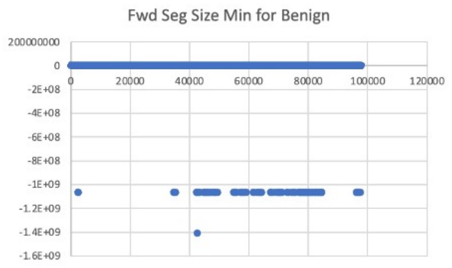
Some DDoS attacks involve asymmetric communication, where the attacker sends large amounts of data to the server during the initial connection phase to consume server resources and establish connections without intending to complete them. Monitoring the Initial Backward Window Bytes in tangent with the Initial Forward Window Bytes can help detect asymmetric communication attempts.

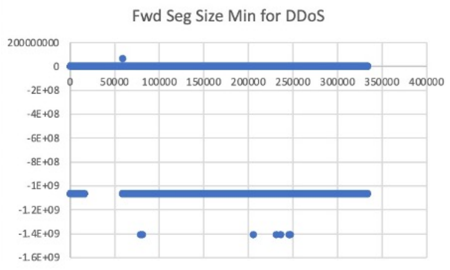




3.2.5 Forward segment size min (fw_seg_min):

DDoS attacks often generate traffic with distinct characteristics compared to regular network traffic. A significant number of small segments can be abnormal behavior in the context of legitimate connections, making it a potential indicator of malicious activity.





3.3 Various Types of Attacks in the Dataset

I plotted the counts of various types of attacks (the column is called "Label") present in this dataset vs their names to see the spread of various DDoS attacks using the matplotlib Python library.

```
import matplotlib.pyplot as plt

# Count the occurrences of each label
label_counts = df['Label'].value_counts()

# Create a bar plot to visualize the label counts
plt.figure(figsize=(10, 6))
label_counts.plot(kind='bar')
plt.title('Comparison of Label Column')
plt.xlabel('Label')
plt.ylabel('Count')
plt.show()
```

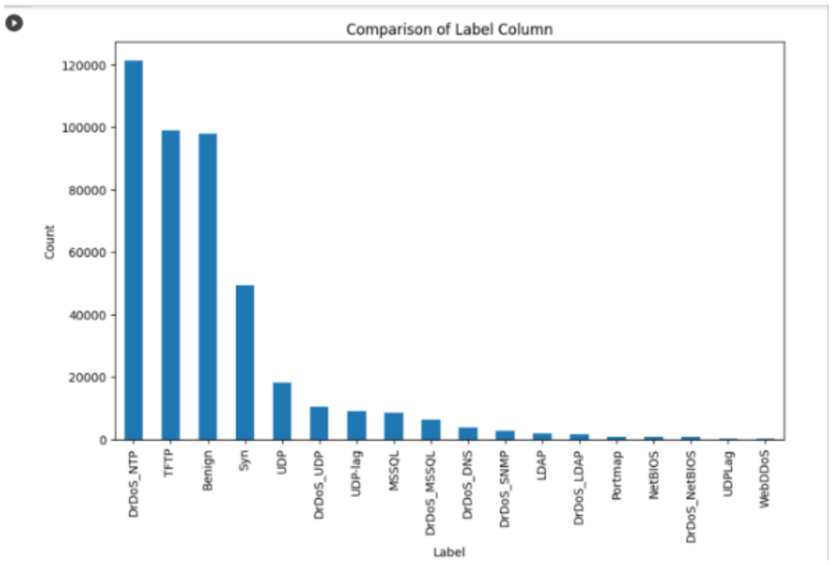
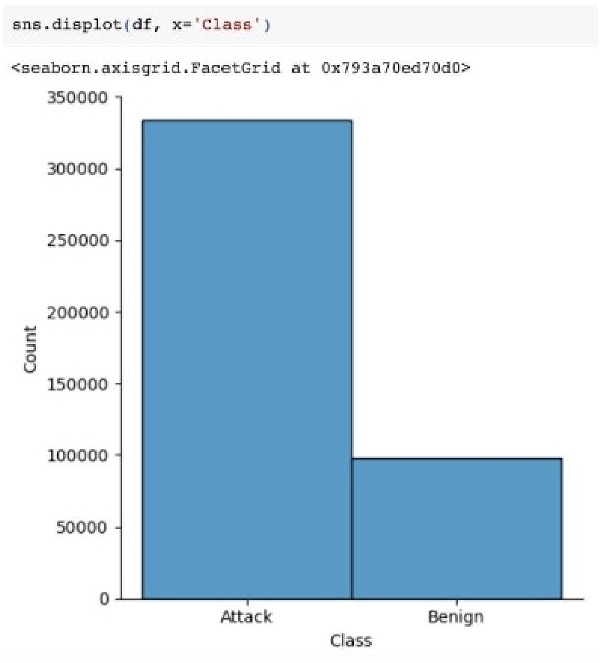


Figure 4: Plotting Types of DDoS Attacks Against Their Count

3.4 Checking Imbalance in the Dataset

The next step I took involved checking for imbalances in the dataset. This dataset is imbalanced towards malicious traffic with the ratio being 3:1 for Attack as compared to Benign as shown in the plot below.



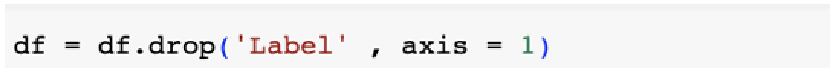
```
sns.displot(df, x='Class')
```

```
<seaborn.axisgrid.FacetGrid at 0x793a70ed70d0>
```

Figure 5: Plotting Attack Versus Benign

4 Preparing the Dataset for Learning

Before machine learning, I dropped the unwanted feature called "Label" which simply classified the type of attack each row belonged to.

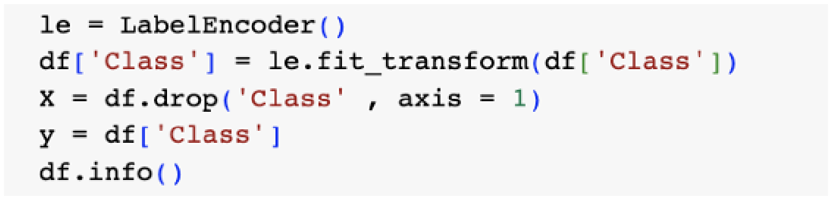


```
df = df.drop('Label' , axis = 1)
```

Figure 6: Deleting Unwanted Feature

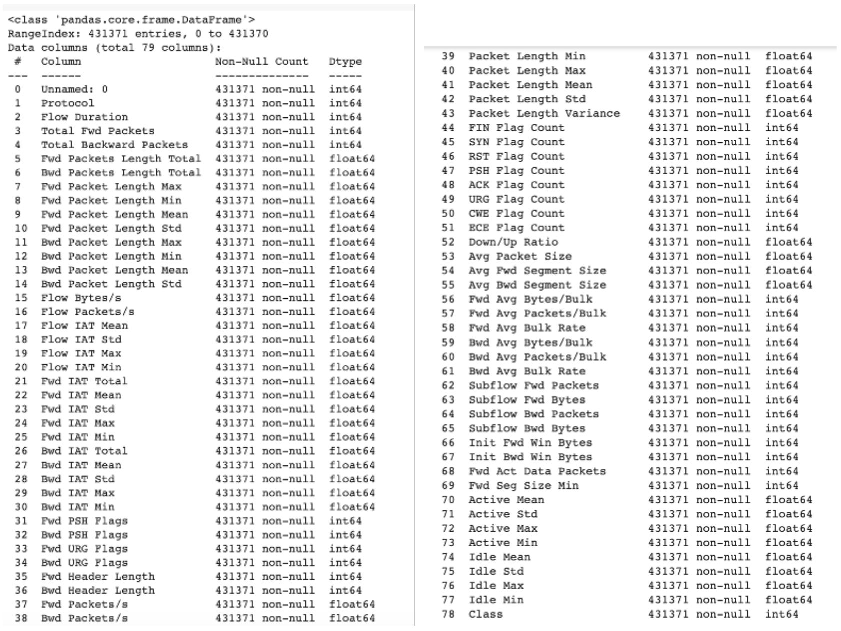
4.1 Label Encoding

Before the machine learning models can be used to train the dataset, the labels need to be encoded. Label encoding is used to convert categorical (A string variable consisting of only a few different values) columns into numerical ones so that they can be fitted by machine learning models that only take numerical data. It is an important data pre-processing step in a machine-learning project. Here the label called "Class" is converted from having the values "Attack" to 0 and "Benign" to 1 by using the LabelEncoder class from the scikit-learn library. Then I separated the data into the features and labels and put them into different arrays, called X and y respectively.



```
le = LabelEncoder()
df['Class'] = le.fit_transform(df['Class'])
X = df.drop('Class' , axis = 1)
y = df['Class']
df.info()
```

Figure 7: Label Encoding [Chu23]

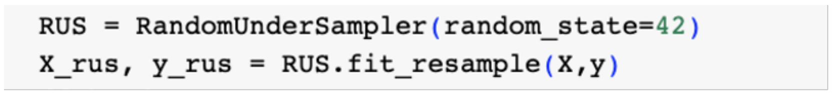


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 431371 entries, 0 to 431370
Data columns (total 79 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   Unnamed: 0                431371 non-null  int64
 1   Protocol                  431371 non-null  int64
 2   Flow Duration             431371 non-null  int64
 3   Total Fwd Packets         431371 non-null  int64
 4   Total Backward Packets    431371 non-null  int64
 5   Fwd Packets Length Total  431371 non-null  float64
 6   Bwd Packets Length Total  431371 non-null  float64
 7   Fwd Packet Length Max     431371 non-null  float64
 8   Fwd Packet Length Min     431371 non-null  float64
 9   Fwd Packet Length Mean    431371 non-null  float64
 10  Fwd Packet Length Std     431371 non-null  float64
 11  Bwd Packet Length Max     431371 non-null  float64
 12  Bwd Packet Length Min     431371 non-null  float64
 13  Bwd Packet Length Mean    431371 non-null  float64
 14  Bwd Packet Length Std     431371 non-null  float64
 15  Flow Bytes/s              431371 non-null  float64
 16  Flow Packets/s            431371 non-null  float64
 17  Flow IAT Mean             431371 non-null  float64
 18  Flow IAT Std              431371 non-null  float64
 19  Flow IAT Max              431371 non-null  float64
 20  Flow IAT Min              431371 non-null  float64
 21  Fwd IAT Total             431371 non-null  float64
 22  Fwd IAT Mean              431371 non-null  float64
 23  Fwd IAT Std               431371 non-null  float64
 24  Fwd IAT Max               431371 non-null  float64
 25  Fwd IAT Min               431371 non-null  float64
 26  Bwd IAT Total             431371 non-null  float64
 27  Bwd IAT Mean              431371 non-null  float64
 28  Bwd IAT Std               431371 non-null  float64
 29  Bwd IAT Max               431371 non-null  float64
 30  Bwd IAT Min               431371 non-null  float64
 31  Fwd PSH Flags             431371 non-null  int64
 32  Bwd PSH Flags             431371 non-null  int64
 33  Fwd URG Flags             431371 non-null  int64
 34  Bwd URG Flags             431371 non-null  int64
 35  Fwd Header Length         431371 non-null  int64
 36  Bwd Header Length         431371 non-null  int64
 37  Fwd Packets/s             431371 non-null  float64
 38  Bwd Packets/s             431371 non-null  float64
 39  Packet Length Min         431371 non-null  float64
 40  Packet Length Max         431371 non-null  float64
 41  Packet Length Mean        431371 non-null  float64
 42  Packet Length Std         431371 non-null  float64
 43  Packet Length Variance    431371 non-null  float64
 44  FIN Flag Count            431371 non-null  int64
 45  SYN Flag Count            431371 non-null  int64
 46  RST Flag Count            431371 non-null  int64
 47  PSH Flag Count            431371 non-null  int64
 48  ACK Flag Count            431371 non-null  int64
 49  URG Flag Count            431371 non-null  int64
 50  CWE Flag Count            431371 non-null  int64
 51  ECE Flag Count            431371 non-null  int64
 52  Down/Up Ratio             431371 non-null  float64
 53  Avg Packet Size           431371 non-null  float64
 54  Avg Fwd Segment Size      431371 non-null  float64
 55  Avg Bwd Segment Size      431371 non-null  float64
 56  Fwd Avg Bytes/Bulk        431371 non-null  int64
 57  Fwd Avg Packets/Bulk      431371 non-null  int64
 58  Fwd Avg Bulk Rate         431371 non-null  int64
 59  Bwd Avg Bytes/Bulk        431371 non-null  int64
 60  Bwd Avg Packets/Bulk      431371 non-null  int64
 61  Bwd Avg Bulk Rate         431371 non-null  int64
 62  Subflow Fwd Packets       431371 non-null  int64
 63  Subflow Fwd Bytes         431371 non-null  int64
 64  Subflow Bwd Packets       431371 non-null  int64
 65  Subflow Bwd Bytes         431371 non-null  int64
 66  Init Fwd Win Bytes        431371 non-null  int64
 67  Init Bwd Win Bytes        431371 non-null  int64
 68  Fwd Act Data Packets      431371 non-null  int64
 69  Fwd Seg Size Min          431371 non-null  int64
 70  Active Mean               431371 non-null  float64
 71  Active Std                431371 non-null  float64
 72  Active Max                431371 non-null  float64
 73  Active Min                431371 non-null  float64
 74  Idle Mean                 431371 non-null  float64
 75  Idle Std                  431371 non-null  float64
 76  Idle Max                  431371 non-null  float64
 77  Idle Min                  431371 non-null  float64
 78  Class                     431371 non-null  int64
```

Figure 8: Columns in the Data Frame

4.2 Random Undersampling of Data

Since this dataset is imbalanced, this bias in the training dataset can influence the machine learning algorithms, leading them to ignore the minority class entirely. In our case, this would lead to ignoring benign data, which would lead to an incorrect model. In order to mitigate this problem, I randomly resampled the training dataset. The method I used is called random undersampling which deletes some data from the majority category, leaving me with 97,831 rows, for both the benign and DDoS types.

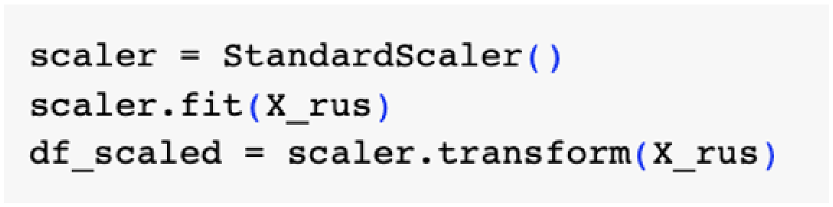


```
RUS = RandomUnderSampler(random_state=42)
X_rus, y_rus = RUS.fit_resample(X,y)
```

Figure 9: Random Undersampling of Data [Bro21]

4.3 Scaling the Data

The dataset contains features of various dimensions and scales together. Different scales of the features affect the modeling of a dataset adversely, which leads to a biased outcome of predictions. So, it is necessary to scale the data before modeling. To accomplish this, I used standardization, which is a scaling method making the data scale-free by resizing distribution of values so that mean of observed values is 0 and standard deviation is 1. I have used StandardScaler from scikit-learn to do this on the randomly undersampled features array.

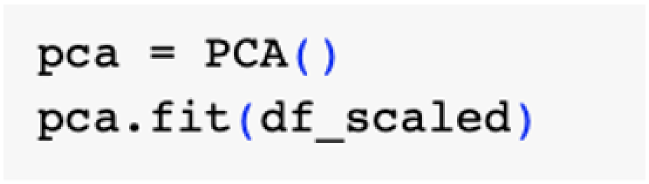


```
scaler = StandardScaler()
scaler.fit(X_rus)
df_scaled = scaler.transform(X_rus)
```

Figure 10: Scaling of Data [Kha21b]

4.4 Performing PCA on Data

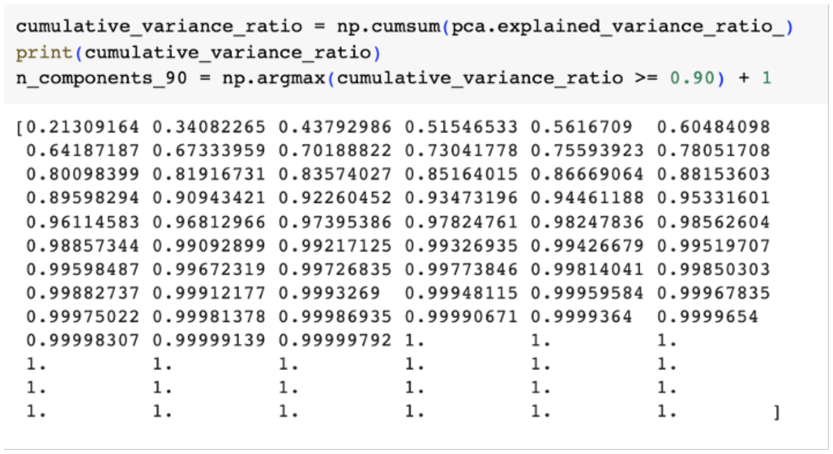
Many datasets, like the one I am using for this paper, have many variables or features, known as high-dimensional data. This can lead to computational inefficiencies and difficulties in visualization. To mitigate this issue, I performed Principal Component Analysis, also known as PCA. It can reduce the number of dimensions while retaining the most important information, making data more manageable and interpretable. This makes it an important tool for data analysis, feature selection, and enhancing the performance of machine learning algorithms. I performed PCA on the scaled dataset containing features using the PCA class from the scikit-learn library.



```
pca = PCA()
pca.fit(df_scaled)
```

Figure 11: Performing Principal Component Analysis on the Data [SK23a]

I have tried retaining 90% of the variance in data using the PCA and determined the minimum number of features which would allow me to keep 90% of variance in data.

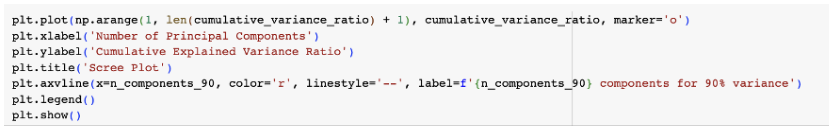


```
cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_)
print(cumulative_variance_ratio)
n_components_90 = np.argmax(cumulative_variance_ratio >= 0.90) + 1
```

```
[0.21309164 0.34082265 0.43792986 0.51546533 0.5616709  0.60484098
 0.64187187 0.67333959 0.70188822 0.73041778 0.75593923 0.78051708
 0.80098399 0.81916731 0.83574027 0.85164015 0.86669064 0.88153603
 0.89598294 0.90943421 0.92260452 0.93473196 0.94461188 0.95331601
 0.96114583 0.96812966 0.97395386 0.97824761 0.98247836 0.98562604
 0.98857344 0.99092899 0.99217125 0.99326935 0.99426679 0.99519707
 0.99598487 0.99672319 0.99726835 0.99773846 0.99814041 0.99850303
 0.99882737 0.99912177 0.9993269  0.99948115 0.99959584 0.99967835
 0.99975022 0.99981378 0.99986935 0.99990671 0.9999364  0.9999654
 0.99998307 0.99999139 0.99999792 1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.        ]
```

Figure 12: Retaining 90% Variance on the Data [SK23a]

A common method for determining the number of features to be retained is a graphical representation known as a scree plot. A Scree Plot is a simple line segment plot that shows the eigenvalues for each individual feature or principal component. For this dataset, using the scree plot, the number of features to keep in order to retain 90% of existing variance in data comes out to be 20 as shown below:



```
plt.plot(np.arange(1, len(cumulative_variance_ratio) + 1), cumulative_variance_ratio, marker='o')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.title('Scree Plot')
plt.axvline(x=n_components_90, color='r', linestyle='--', label=f'{n_components_90} components for 90% variance')
plt.legend()
plt.show()
```

Figure 13

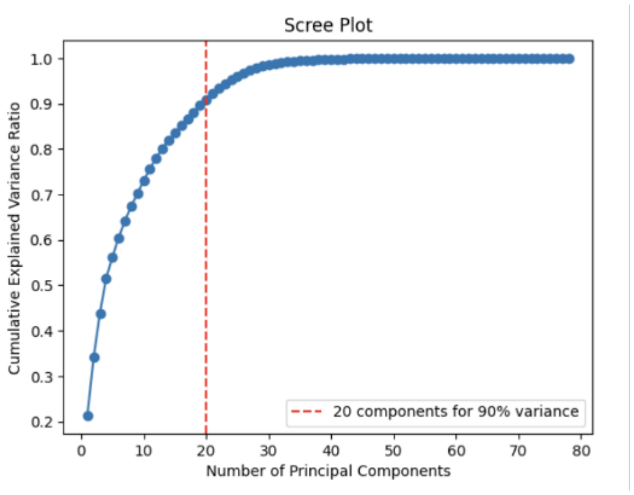
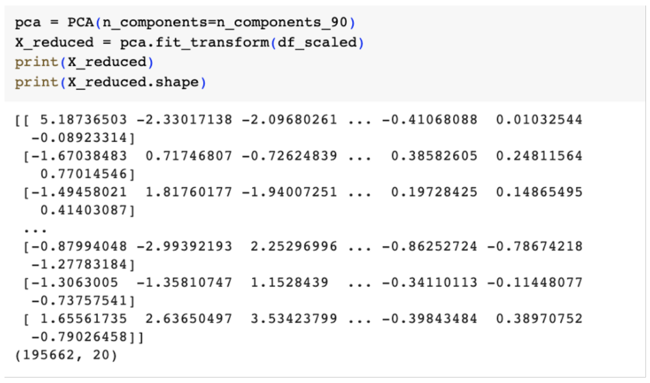


Figure 14: Graphing a Scree Plot [SK23b]



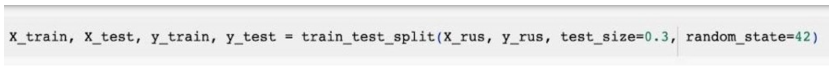
```
pca = PCA(n_components=n_components_90)
X_reduced = pca.fit_transform(df_scaled)
print(X_reduced)
print(X_reduced.shape)
```

```
[[ 5.18736503 -2.33017138 -2.09680261 ... -0.41068088  0.01032544
  -0.08923314]
 [-1.67038483  0.71746807 -0.72624839 ...  0.38582605  0.24811564
   0.77014546]
 [-1.49458021  1.81760177 -1.94007251 ...  0.19728425  0.14865495
   0.41403087]
 ...
 [-0.87994048 -2.99392193  2.25296996 ... -0.86252724 -0.78674218
  -1.27783184]
 [-1.3063005  -1.35810747  1.1528439  ... -0.34110113 -0.11448077
  -0.73757541]
 [ 1.65561735  2.63650497  3.53423799 ... -0.39843484  0.38970752
  -0.79026458]]
(195662, 20)
```

Figure 15

4.5 Splitting the Data

Now, the data must be split into training and testing data; the training data is used to fit the machine learning model so it can differentiate a DDoS attack from benign, while the testing data is used to evaluate the accuracy of the fit machine learning model. I called the train_test_split method of sklearn to split the data into training and testing datasets. I decided on a 70/30 train/test split because it gives enough data to train and test our model and ensures accuracy.



```
X_train, X_test, y_train, y_test = train_test_split(X_rus, y_rus, test_size=0.3, random_state=42)
```

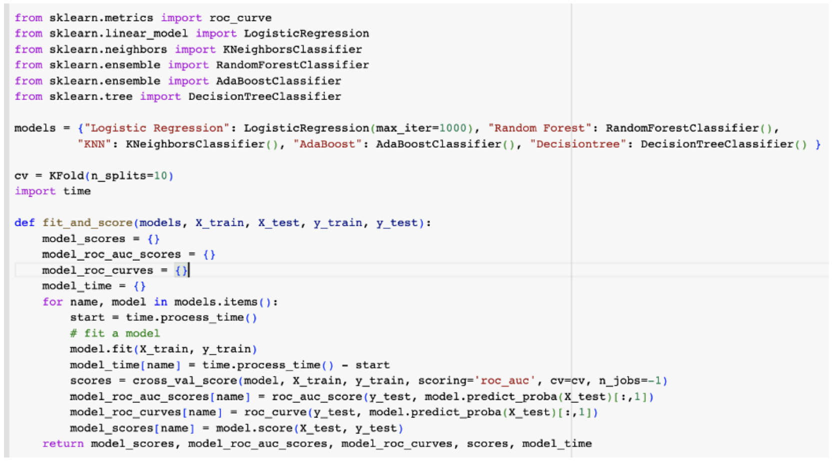
Figure 16: Splitting the Data [Bro20]

5 Processing the Dataset

I used five machine learning models to train the dataset. The models used included Logistic Regression, Random Forest, K-Nearest Neighbors, AdaBoost and Decision Tree.

I created a common method called fit_and_score which takes each of the above five initialized models and uses the scikit-learn method called "fit" to train each machine learning model on the training dataset. I also used the method called cross_val_score from the scikit-learn package which trains and tests a model over multiple folds of your dataset (here number of folds is 10). This cross-validation method provides a better understanding of model performance over the whole dataset instead of just a single train/test split [All22].

Then I used the roc_auc_score function from the sklearn.metrics module that calculates the area under the receiver operating characteristic (ROC) curve for a model on the testing data. The ROC curve is a graphical representation of the performance of a model as its discrimination threshold is varied. The AUC (area under the curve) of the ROC curve is a measure of how well the model can distinguish between the positive and negative classes. The y_test stands for the real values of y. Comparing them with the predicted ones helps determine accuracy of the model [Clo23a].



```
from sklearn.metrics import roc_curve
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

models = {"Logistic Regression": LogisticRegression(max_iter=1000), "Random Forest": RandomForestClassifier(),
         "KNN": KNeighborsClassifier(), "AdaBoost": AdaBoostClassifier(), "Decisiontree": DecisionTreeClassifier() }

cv = KFold(n_splits=10)
import time

def fit_and_score(models, X_train, X_test, y_train, y_test):
    model_scores = {}
    model_roc_auc_scores = {}
    model_roc_curves = {}
    model_time = {}
    for name, model in models.items():
        start = time.process_time()
        # fit a model
        model.fit(X_train, y_train)
        model_time[name] = time.process_time() - start
        scores = cross_val_score(model, X_train, y_train, scoring='roc_auc', cv=cv, n_jobs=-1)
        model_roc_auc_scores[name] = roc_auc_score(y_test, model.predict_proba(X_test)[:,1])
        model_roc_curves[name] = roc_curve(y_test, model.predict_proba(X_test)[:,1])
        model_scores[name] = model.score(X_test, y_test)
    return model_scores, model_roc_auc_scores, model_roc_curves, scores, model_time
```

Figure 17: Training the Dataset Using Various Models

The salient features and why the above five machine learning models were selected for DDoS identification are detailed below.

5.1 Logistic Regression

Logistic Regression provides interpretable results, as it estimates the relationship between the input features and the probability of a DDoS attack occurring. This transparency can be valuable for understanding the impact of different features on the classification decision.

Logistic Regression is computationally efficient and relatively fast to train and predict. It can handle large datasets without requiring significant computational resources.

Due to its simplicity and efficiency, Logistic Regression can scale well to large-scale DDoS detection scenarios.

5.2 Random Forest

Random Forest is an ensemble learning technique that combines multiple decision trees, reducing the risk of overfitting and improving generalization on unseen data. This property can be beneficial in handling the diverse and complex patterns often found in DDoS attacks.

Random Forest can rank the importance of features in the dataset, providing insights into which features contribute most to the distinction between normal traffic and DDoS attacks. This information can be valuable for feature engineering and selecting relevant features.

5.3 KNN

DDoS attacks can exhibit non-linear patterns in network traffic data. KNN is a non-parametric algorithm, which means it can capture complex, non-linear relationships between features and target classes, making it potentially effective in detecting such patterns.

KNN can be used for anomaly detection since it classifies data points based on the majority class of their k-nearest neighbors. In the context of DDoS detection, attacks are often considered anomalies compared to normal network traffic, and KNN can help identify these anomalies.

KNN is relatively easy to implement and understand. It does not require a complex training process, making it suitable for quick prototyping and implementation.

5.4 AdaBoost

DDoS attacks can exhibit complex and non-linear patterns in network traffic data. AdaBoost can combine multiple weak classifiers, typically decision trees, to create a more powerful model capable of capturing complex relationships in the data.

AdaBoost assigns higher weights to informative features during training, which can lead to better feature selection and focus on the most relevant features for DDoS detection.

Similar to other ensemble methods, AdaBoost is adaptive and can adapt to changes in the data distribution. This makes it suitable for handling dynamic DDoS attack patterns.

5.5 Decision Tree

Decision trees can implicitly rank the importance of features in the dataset. For DDoS attack detection, certain features or traffic characteristics might have higher relevance in identifying attacks. Decision trees can identify these features at the top levels of the tree, making it more efficient.

DDoS attacks often exhibit complex and non-linear relationships among various features, like traffic volume and packet rates. Decision trees can capture these non-linear relationships by recursively partitioning the feature space into regions that better differentiate between normal and attack traffic.

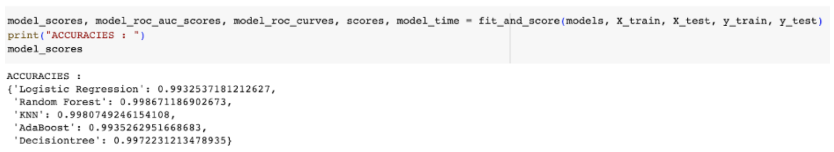
Using the common fit_and_score method, I calculated the accuracy as well as ROC-AUC scores for each of the five supervised machine learning algorithms mentioned above.

5.6 Calculating Accuracy

Accuracy is a metric that shows how a model performs across all classes. It is useful when all classes are of equal importance. It is calculated as the ratio between the number of correct predictions to the total number of predictions.

$$\boldsymbol{Accuracy} = \frac{\boldsymbol{True}_{\boldsymbol{positive}} + \boldsymbol{True}_{\boldsymbol{negative}}}{\boldsymbol{True}_{\boldsymbol{positive}} + \boldsymbol{True}_{\boldsymbol{negative}} + \boldsymbol{False}_{\boldsymbol{positive}} + \boldsymbol{False}_{\boldsymbol{negative}}}$$

Figure 18: Formula for Calculating Accuracy [Doc23]



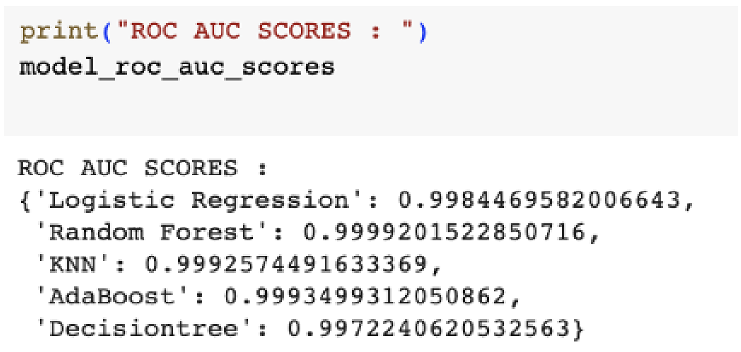
```
model_scores, model_roc_auc_scores, model_roc_curves, scores, model_time = fit_and_score(models, X_train, X_test, y_train, y_test)
print("ACCURACIES : ")
model_scores
```

```
ACCURACIES :
{'Logistic Regression': 0.9932537181212627,
 'Random Forest': 0.998671186902673,
 'KNN': 0.9980749246154108,
 'AdaBoost': 0.9935262951668683,
 'Decisiontree': 0.9972231213478935}
```

Figure 19: Accuracy Values for Various Models

5.7 Calculating ROC-AUC Score

The ROC-AUC is the area under the ROC curve. It is the metric that is used to measure how well the model can distinguish two classes. The score ranges from 0.5 to 1, and the score being 1 is the ideal case where TPR (true positive rate) is 1 and FPR (false positive rate) is 0, which means I correctly classify all positives and negatives.

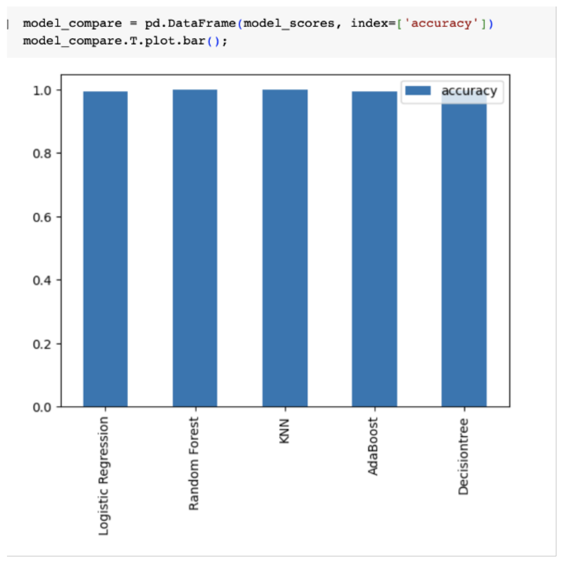


```
print("ROC AUC SCORES : ")
model_roc_auc_scores
```

```
ROC AUC SCORES :
{'Logistic Regression': 0.9984469582006643,
 'Random Forest': 0.9999201522850716,
 'KNN': 0.9992574491633369,
 'AdaBoost': 0.9993499312050862,
 'Decisiontree': 0.9972240620532563}
```

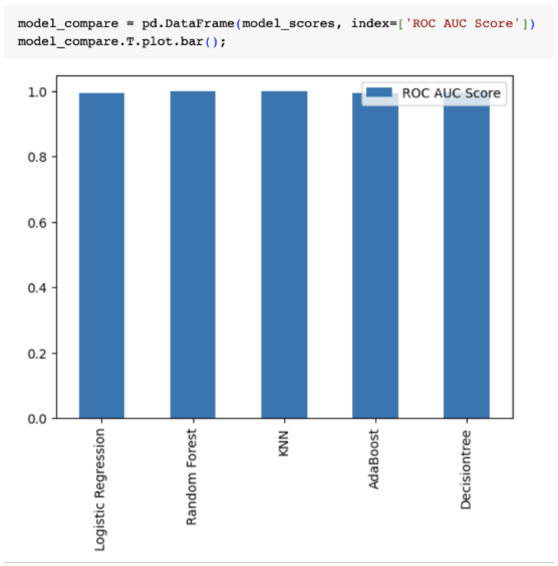
Figure 20: ROC-AUC Scores for Various Models

Comparing the ROC-AUC scores for the five models leads us to the realization that the better option among these classifiers is Random Forest even though all of them perform well. Bar graphs comparing their accuracies and ROC-AUC scores are shown below:



```
model_compare = pd.DataFrame(model_scores, index=['accuracy'])
model_compare.T.plot.bar();
```

Figure 21: Plot of Accuracy Values for Various Models

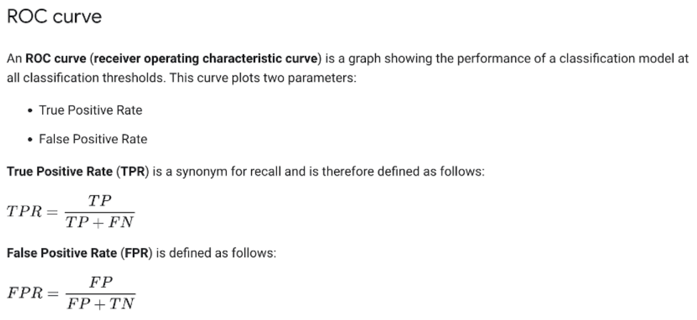


```
model_compare = pd.DataFrame(model_scores, index=['ROC AUC Score'])
model_compare.T.plot.bar();
```

Figure 22: Plot of ROC-AUC Scores for Various Models

5.8 Plotting ROC Curve

To investigate further, I used a ROC curve to evaluate the performance of all five models.



ROC curve

An **ROC curve** (**receiver operating characteristic curve**) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

True Positive Rate (**TPR**) is a synonym for recall and is therefore defined as follows:

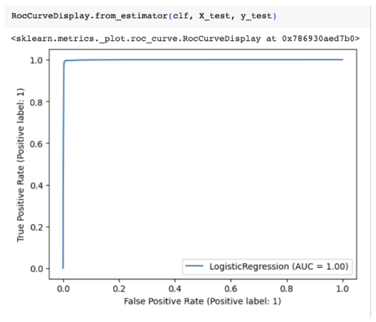
$$TPR = \frac{TP}{TP + FN}$$

False Positive Rate (**FPR**) is defined as follows:

$$FPR = \frac{FP}{FP + TN}$$

Figure 23: Components of ROC Curve [Sch23]

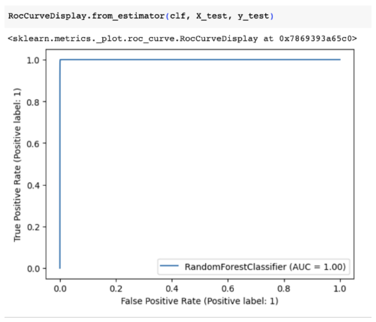
ROC curves for the five machine learning models are displayed below. They are all equally good in terms of the area under the ROC curve which is almost 1 for all of them.



```
RocCurveDisplay.from_estimator(clf, X_test, y_test)
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x786930aed7b0>
```

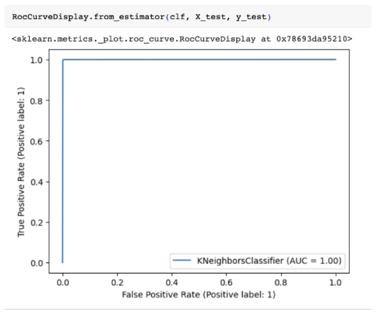
Figure 24: ROC Curve for Logistic Regression



```
RocCurveDisplay.from_estimator(clf, X_test, y_test)
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7869393a65c0>
```

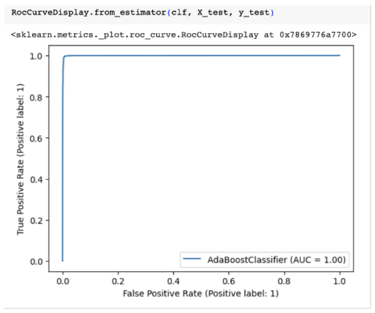
Figure 25: ROC Curve for Random Forest



```
RocCurveDisplay.from_estimator(clf, X_test, y_test)
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x78693da95210>
```

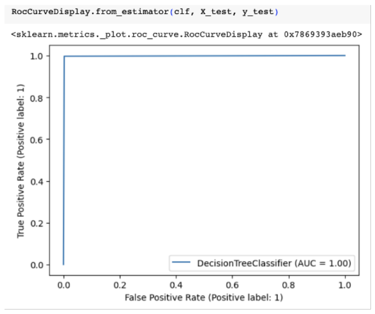
Figure 26: ROC Curve for KNN



```
RocCurveDisplay.from_estimator(clf, X_test, y_test)
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7869776a7700>
```

Figure 27: ROC Curve for AdaBoost



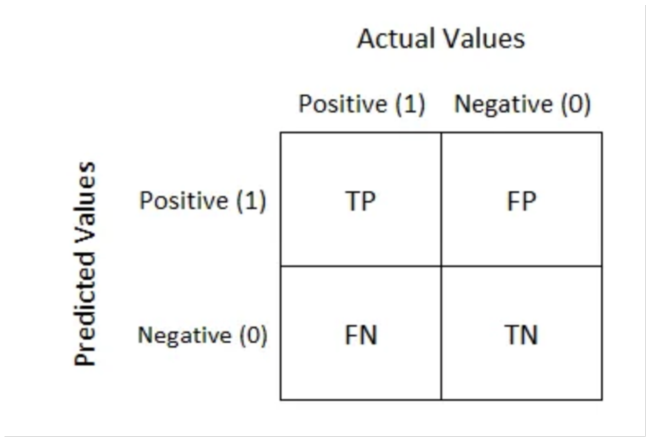
```
RocCurveDisplay.from_estimator(clf, X_test, y_test)
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7869393aeb90>
```

Figure 28: ROC Curve for Decision Tree

5.9 Creating Confusion Matrix

Furthermore, I checked the effectiveness of the models using a confusion matrix. It is a table with 4 different combinations of predicted and actual values.



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 29: Confusion Matrix [Nar21]

The lower the values in the false positive and false negative blocks, the better the effectiveness of the model. The higher the values in the true positive and true negative blocks, the better the effectiveness of the model.

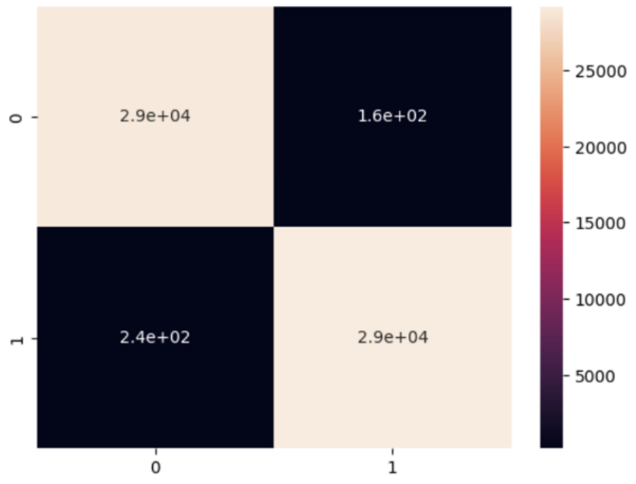


Figure 30: Confusion Matrix for the Logistic Regression Model

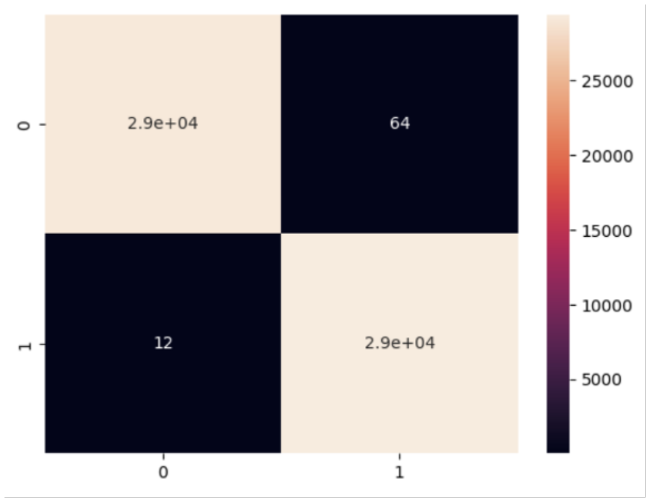


Figure 31: Confusion Matrix for the Random Forest Model

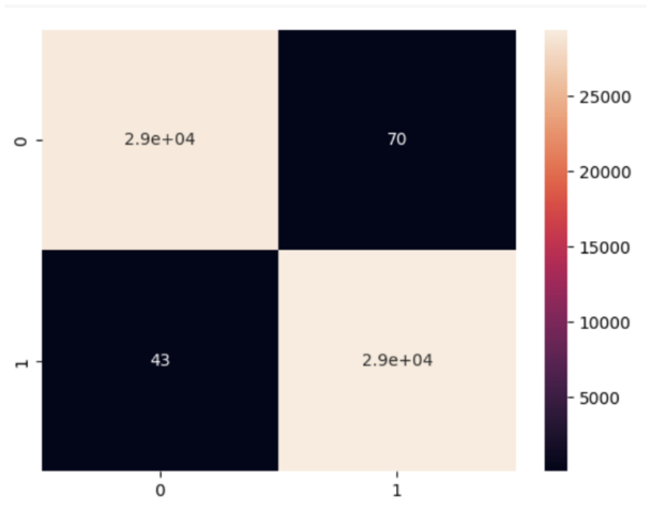


Figure 32: Confusion Matrix for the KNN Model

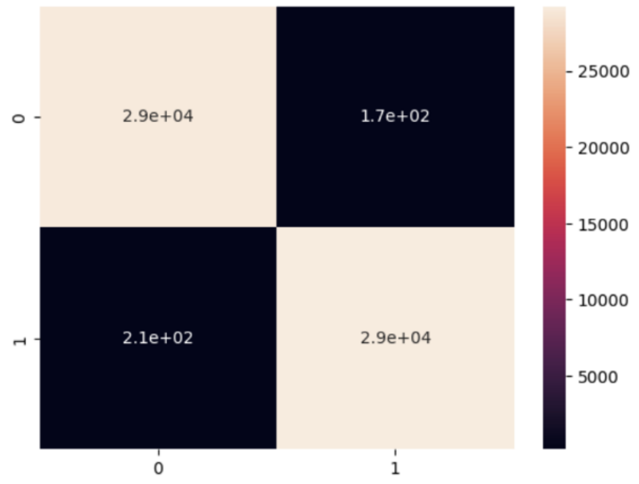


Figure 33: Confusion Matrix for the AdaBoost Model

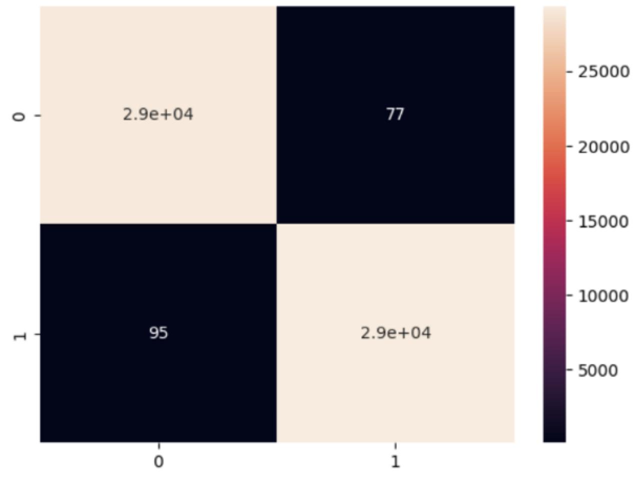
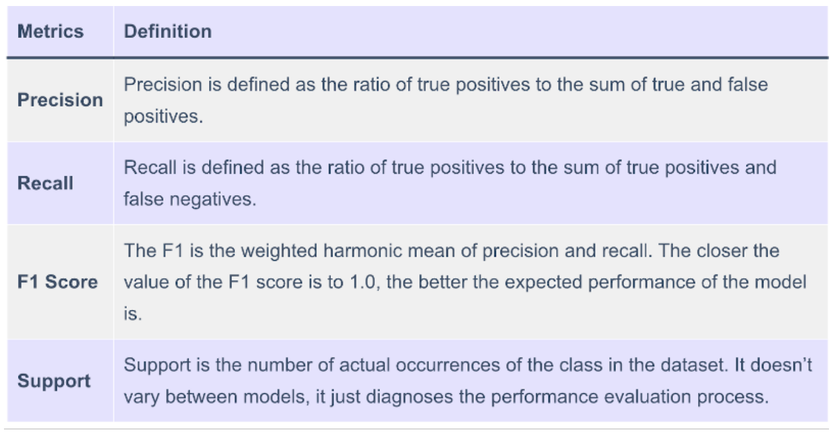


Figure 34: Confusion Matrix for the Decision Tree Model

5.10 Creating Classification Report

I also evaluated the performance of these models using the classification report. A classification report is a performance evaluation metric in machine learning. It is used to show the precision, recall, F1 Score, and support of the trained classification model.

The different metrics of a classification report are described as shown below:



Metrics	Definition
Precision	Precision is defined as the ratio of true positives to the sum of true and false positives.
Recall	Recall is defined as the ratio of true positives to the sum of true positives and false negatives.
F1 Score	The F1 is the weighted harmonic mean of precision and recall. The closer the value of the F1 score is to 1.0, the better the expected performance of the model is.
Support	Support is the number of actual occurrences of the class in the dataset. It doesn't vary between models, it just diagnoses the performance evaluation process.

Figure 35: Components of Classification Report [Kha21a]

Precision is the true positive divided by the sum of the true positive and false positive.

Precision = TruePositives / (TruePositives + FalsePositives)

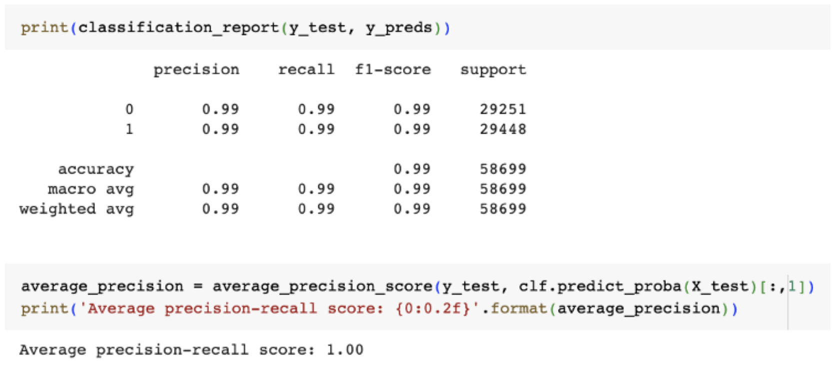
Recall is the true positive divided by the sum of the true positive and false negative.

Recall = TruePositives / (TruePositives + FalseNegatives)

F1 score is two times the recall times the precision over the sum of the recall and precision.

F1 score = 2 * (precision * recall) / (precision + recall)

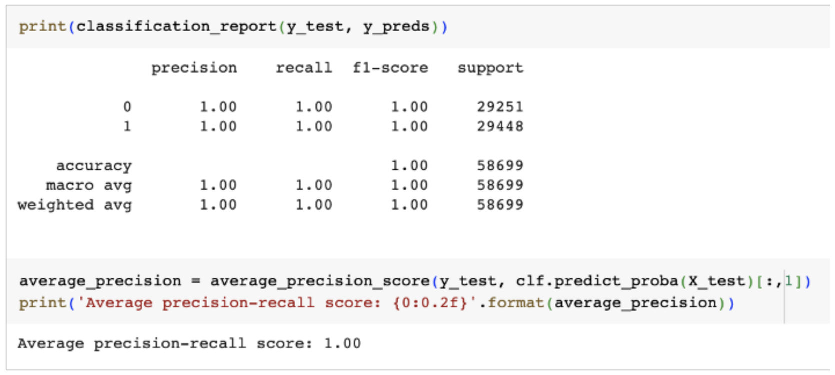
The classification reports for the five supervised machine learning models are:



```
print(classification_report(y_test, y_preds))

              precision    recall  f1-score   support

           0       0.99      0.99      0.99     29251
           1       0.99      0.99      0.99     29448

    accuracy                           0.99     58699
   macro avg       0.99      0.99      0.99     58699
weighted avg       0.99      0.99      0.99     58699
```

```
average_precision = average_precision_score(y_test, clf.predict_proba(X_test)[:,1])
print('Average precision-recall score: {0:0.2f}'.format(average_precision))

Average precision-recall score: 1.00
```

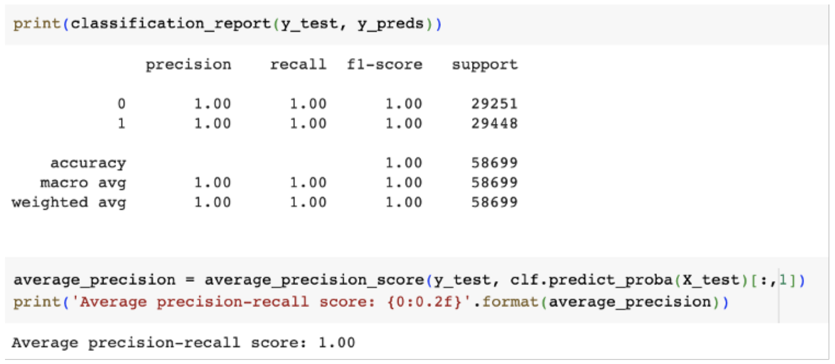
Figure 36: Classification Report for Logistic Regression Model



```
print(classification_report(y_test, y_preds))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     29251
           1       1.00      1.00      1.00     29448

    accuracy                           1.00     58699
   macro avg       1.00      1.00      1.00     58699
weighted avg       1.00      1.00      1.00     58699
```

```
average_precision = average_precision_score(y_test, clf.predict_proba(X_test)[:,1])
print('Average precision-recall score: {0:0.2f}'.format(average_precision))

Average precision-recall score: 1.00
```

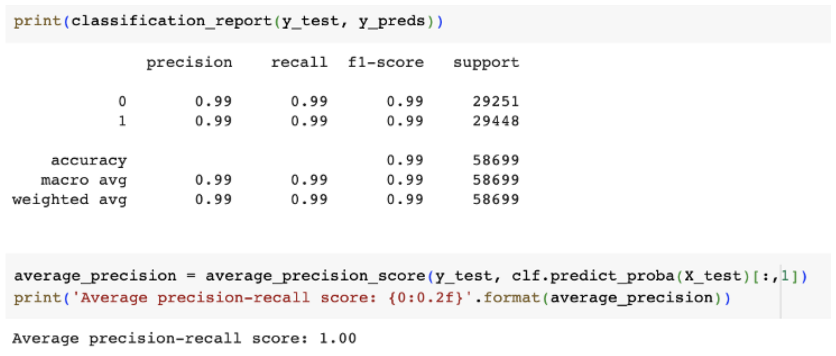
Figure 37: Classification Report for Random Forest Model



```
print(classification_report(y_test, y_preds))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     29251
           1       1.00      1.00      1.00     29448

    accuracy                           1.00     58699
   macro avg       1.00      1.00      1.00     58699
weighted avg       1.00      1.00      1.00     58699
```

```
average_precision = average_precision_score(y_test, clf.predict_proba(X_test)[:,1])
print('Average precision-recall score: {0:0.2f}'.format(average_precision))

Average precision-recall score: 1.00
```

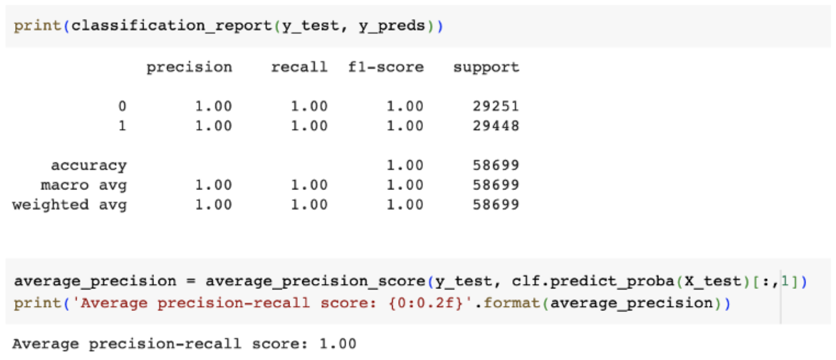
Figure 38: Classification Report for KNN Model



```
print(classification_report(y_test, y_preds))

              precision    recall  f1-score   support

           0       0.99      0.99      0.99     29251
           1       0.99      0.99      0.99     29448

    accuracy                           0.99     58699
   macro avg       0.99      0.99      0.99     58699
weighted avg       0.99      0.99      0.99     58699
```

```
average_precision = average_precision_score(y_test, clf.predict_proba(X_test)[:,1])
print('Average precision-recall score: {0:0.2f}'.format(average_precision))

Average precision-recall score: 1.00
```

Figure 39: Classification Report for AdaBoost Model



```
print(classification_report(y_test, y_preds))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     29251
           1       1.00      1.00      1.00     29448

    accuracy                           1.00     58699
   macro avg       1.00      1.00      1.00     58699
weighted avg       1.00      1.00      1.00     58699
```

```
average_precision = average_precision_score(y_test, clf.predict_proba(X_test)[:,1])
print('Average precision-recall score: {0:0.2f}'.format(average_precision))

Average precision-recall score: 1.00
```

Figure 40: Classification Report for Decision Tree Model

The closer the values of precision, recall, and f1-score to 1, the better the effectiveness of the model.

Finally, I evaluated the performance of all five supervised machine learning models using the precision recall curve.

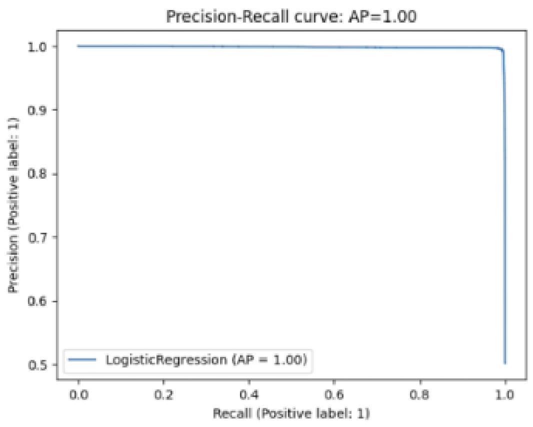


Figure 41: Precision Recall Curve for the Logistic Regression Model

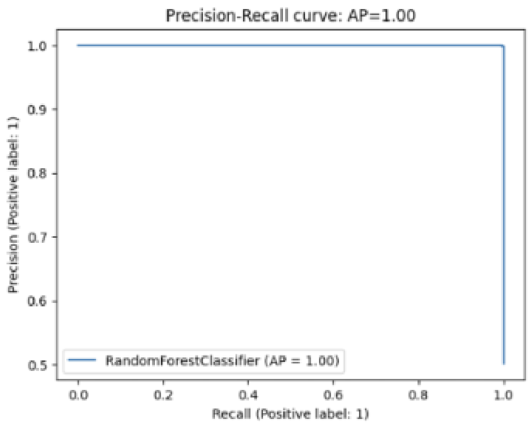


Figure 42: Precision Recall Curve for the Random Forest Model

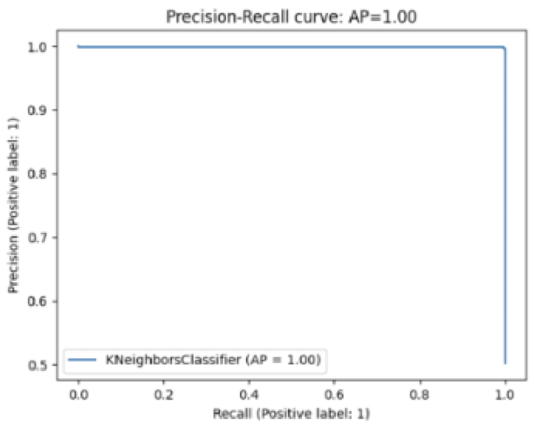


Figure 43: Precision Recall Curve for the KNN Model

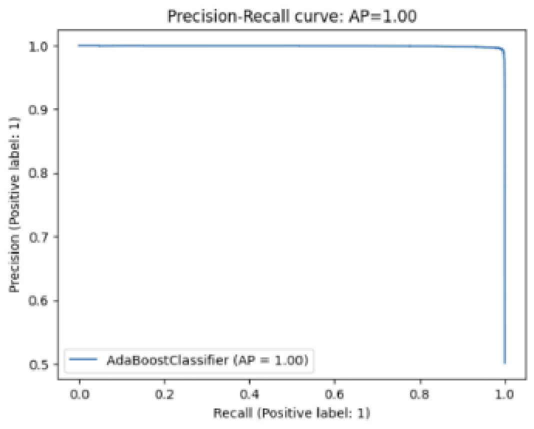


Figure 44: Precision Recall Curve for the AdaBoost Model

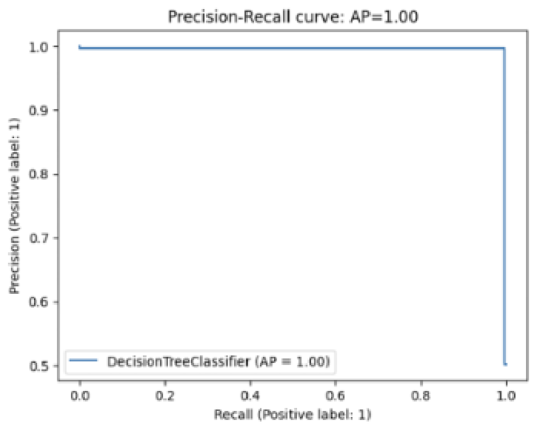


Figure 45: Precision Recall Curve for the Decision Tree Model

6 Conclusion and Further Analysis

I achieved 99% accuracy for all of the supervised machine learning models. Overall, I recommend Random Forest over Decision Tree based on the confusion matrix made from our dataset. As shown in Figures 18-45 above, I utilized various performance indicators to compare the models, including confusion matrices, precision recall curves, ROC curves, and ROC-AUC scores. The three most commonly used performance indicators are accuracy, recall, and precision. F1 score provided additional information on the model performance.

It must be noted that this study was conducted on various types of DDoS attacks currently known. In the future, if new types of DDoS attacks emerge, this analysis will have to be performed again as the new types may not be detected that well using Random Forest. Additionally, my research shows that supervised learning models are effective in identifying DDoS attacks, but they need pre-labeled datasets and training, which is unavailable for not-yet-known attacks.

Due to new types of DDoS attacks occurring every day, there are large differences between known and lab-based train datasets and real-time DDoS attacks. This causes the false-negative rate to be higher than anticipated. Further research is needed on accurately detecting DDoS attacks under real scenarios and different types of test datasets.

Aside from conducting research to find the limitations of existing methods and datasets, I propose that the focus should also be on developing new types of DDoS attacks to anticipate potential future attacks.

References

[All22] Stephen Allwright. Using cross_val_score in sklearn, simply explained. https://stephenallwright.com/cross_val_score-sklearn/, 2022.

[Bro20] Jason Brownlee. Train-test split for evaluating machine learning algorithms. https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms, 2020.

[Bro21] Jason Brownlee. Random oversampling and undersampling for imbalanced classification. https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/, 2021.

[Cho23] Jean-Christopher Chouinard. How to use classification report in scikit-learn (python). https://www.jcchouinard.com/classification-report-in-scikit-learn/, 2023.

[Chu23] Aakarsha Chugh. Label encoding in python. https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/, 2023.

[Clo23a] Saturn Cloud. How to improve your model's performance with sklearn roc_auc_score. https://saturncloud.io/blog/how-to-improve-your-models-performance-with-sklearn-rocaucscore, 2023.

[Clo23b] Cloudflare. What is a distributed denial-of-service (ddos) attack? https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/, 2023.

[Clo23c] Cloudflare. What is an application layer ddos attack? https://www.netscout.com/what-is-ddos/application-layer-attacks, 2023.

[Doc23] Hasty.Ai Documentation. Accuracy. https://hasty.ai/docs/mp-wiki/metrics/accuracy, 2023.

[Goo23a] Google. Classification: Roc curve and auc — machine learning — google for developers. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc#:~:text=An%20ROC%20curve%20, 2023.

[Goo23b] Google. Classification: Roc curve and auc — machine learning — google for developers. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc, 2023.

[Hui23] Purva Huilgol. Precision and recall: Essential metrics for machine learning (2023 update). https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning, 2023.

[Imp23] Imperva. Ddos attack types & mitigation methods: Imperva. https://www.imperva.com/learn/ddos/ddos-attacks, 2023.

[Kha21a] Aman Kharwal. Classification report in machine learning: Aman kharwal. https://thecleverprogrammer.com/2021/07/07/classification-report-in-machine-learning, 2021.

[Kha21b] Aman Kharwal. Standardscaler in machine learning: Aman kharwal. https://thecleverprogrammer.com/2020/09/22/standardscaler-in-machine-learning, 2021.

[Man20] Sanchita Mangale. Scree plot. https://sanchitamangale12.medium.com/scree-plot-733ed72c8608, 2020.

[Mik19] Bartosz Mikulski. Pca-how to choose the number of components? https://www.mikulskibartosz.name/pca-how-to-choose-the-number-of-components, 2019.

[Naj23] et al Najafimehr, Mohammad. Ddos attacks and machine-learning-based detection methods: A survey and taxonomy. https://onlinelibrary.wiley.com/doi/full/10.1002/eng2.12697, 2023.

[Nar21] Sarang Narkhede. Understanding confusion matrix. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62, 2021.

[Net23] Palo Alto Networks. What is a denial of service attack (dos)? https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos, 2023.

[One23] OneLogin. What is a ddos attack: Types, prevention & remediation. https://www.onelogin.com/learn/ddos-attack, 2023.

[Pan22] Pankaj. Numpy.cumsum() in python. https://www.digitalocean.com/community/tutorials/numpy-cumsum-in-python, 2022.

[Sch23] Frank Schoonjans. Roc curve analysis. https://www.medcalc.org/manual/roc-curves.php, 2023.

[SK23a] Paula Villasante Soriano and Cansu Kebabci. Principal component analysis (pca) in python: Sklearn example. https://statisticsglobe.com/principal-component-analysis-python, 2023.

[SK23b] Paula Villasante Soriano and Cansu Kebabci. Scree plot for pca explained: Tutorial, example & how to interpret. https://statisticsglobe.com/scree-plot-pca, 2023.

[Ste20] Doug Steen. Precision-recall curves. https://medium.com/@douglaspsteen/precision-recall-curves-d32e5b290248, 2020.

[Tal23] Md Alamin Talukder. Cic-ddos2019 dataset. https://data.mendeley.com/datasets/ssnc74xm6r/1, 2023.

[Zen23] Zenarmor. Dos and ddos attacks. what are their differences? https://www.zenarmor.com/docs/network-security-tutorials/dos-vs-ddos-attacks, 2023.