Real-Time Intrusion Detection System of Network Traffic Data

Introduction:

Intrusion Detection Systems (IDS) have emerged as crucial components in computer and network security. While existing classification methods each have their strengths and weaknesses, there remains significant room for improvement in addressing their limitations. Recent years have seen a proliferation of proposed IDS models and extensive research comparing various classification techniques.

The CIC-IDS- 2017 dataset consists of generated network traffic data using CICF lowMeter, which resembles true real-world data (PCAPs). We can monitor network traffic in real-time and detect if the activity is suspicious or benign. The dataset can be found on this link [1]. The malicious attacks that this dataset captures are Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet, and DDoS. There are 79 features, including connection-related attributes like timestamps, source and destination IPs, ports, protocols, etc.

Background:

In paper [2], Sharafaldin et al. (2018) discuss how previous datasets to analyze network traffic could be more efficient as they lack traffic diversity and volume or cover the variety of attacks to build robust anomaly detection and classification systems. The CIC-IDS2017 dataset produced by the authors of [2] contains realistic traffic patterns with benign and diverse label attack scenarios.

Paper [3] explores the AdaBoost machine-learning technique using the CIC-IDS2017 dataset. The authors use the Synthetic Minority Oversampling Technique (SMOTE) to solve class imbalance as the number of benign data exceeds the author data. Additionally, the authors use Principal component analysis and the SMOTE as preprocessing techniques to significantly enhance the AdaBoost classifier's accuracy. The paper uses metrics like accuracy, precision, recall, and F1 Score to evaluate the algorithm's performance.

In paper [4], the authors used machine-learning classifiers such as Decision Trees, Random Forest, Extra Tree, and Extreme Gradient Boosting. They analyze the performance on the CIC-IDS2017 dataset and got an accuracy of around 99%. This high level of accuracy underscores the model's robustness and effectiveness in detecting network intrusions and countering potential threats.

Problem Definition:

We are developing a real-time network intrusion detection system that employs multiple trained machine learning algorithms for label classification of network attacks.

The motivation to build a real-time machine learning-based intrusion detection system arises from the several critical advantages it offers.

1. Enables early detection of threats through continuous network traffic monitoring
2. Leverages machine learning to identify both known and unknown zero-day threats
3. Provides automated response capabilities to minimize breach impact and reduce operational downtime

4. Helps organizations maintain compliance with security regulations while protecting sensitive data

Methodology

Exploratory Data Analysis and Subsequent Data Preprocessing:

We perform Exploratory Data Analysis over our dataset and use our visualizations to clean and transform the data into features that are more suited to the models that we then utilize.

1. **Column - Renaming:** We print the column names and observe the presence of spaces between words as well as at the start of each name. We remove such spaces and replace space with "_" between words for consistency.

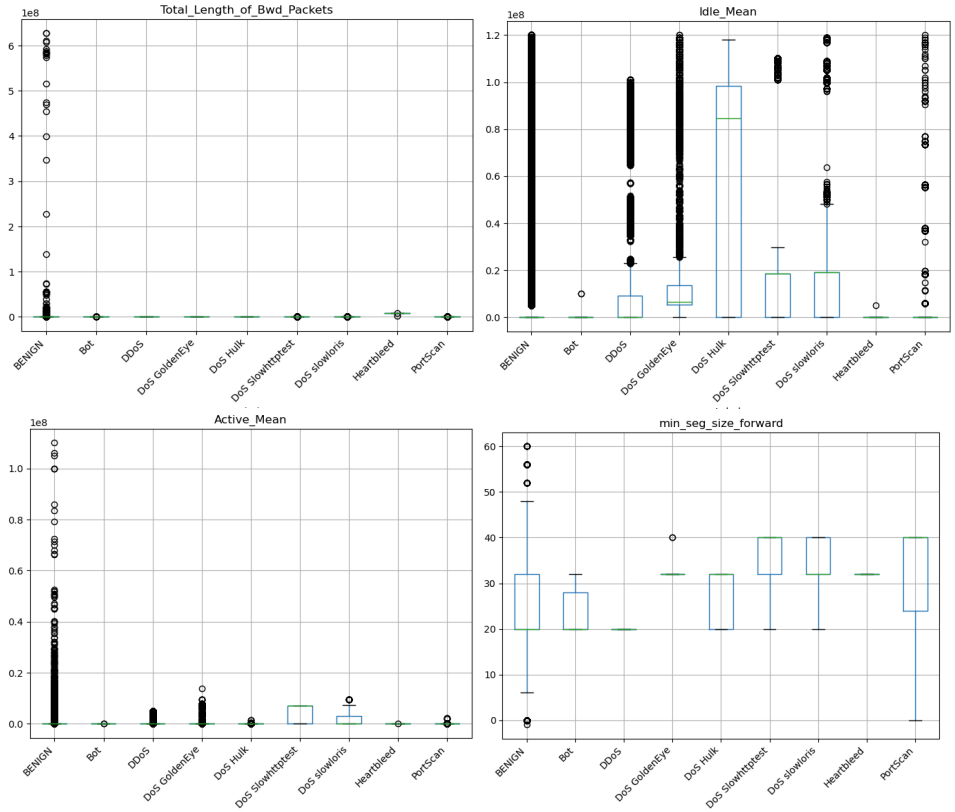
2. **Label Encoding:** We print and observe labels to be strings and thus use a Label Encoder to encode the labels into numbers. We also print the mapping to get a better visualization of this.

3. **Eliminating missing values:** Removing missing values is important for reducing the size and complexity of your dataset and avoiding errors or biases. df.isna() showed that there seem to be no missing/null values in the dataset.

4. **Nan/ unacceptable values:** Analysis of the columns shows that there are some attributes with infinity/nan values. We observed that only ~1000 rows had an inf value out of ~1.4 million rows, and these were restricted to the Flow_Bytes and Flow_Packets columns. We dropped these rows.

Column	Inf/Nan Count
Flow_Bytes/s	769
Flow_Packets/s	1824

5. **Analysis of input features:**



Plotting the spread of the features versus each label shows how they differ by label. This can inform our decision-making with regard to less and more discriminative features.

6. We plotted the Correlation heatmap and observed 10 columns which had no correlation since they had no variation in their values—each value in the column is the same
   These are the columns:
   1. Bwd_PSH_Flags
   2. Fwd_URG_Flags
   3. Bwd_URG_Flags
   4. CWE_Flag_Count
   5. Fwd_Avg_Bytes/Bulk
   6. Fwd_Avg_Packets/Bulk
   7. Fwd_Avg_Bulk_Rate
   8. Bwd_Avg_Bytes/Bulk
   9. Bwd_Avg_Packets/Bulk
   10. Bwd_Avg_Bulk_Rate

   We dropped these columns as they will not impact the output label.

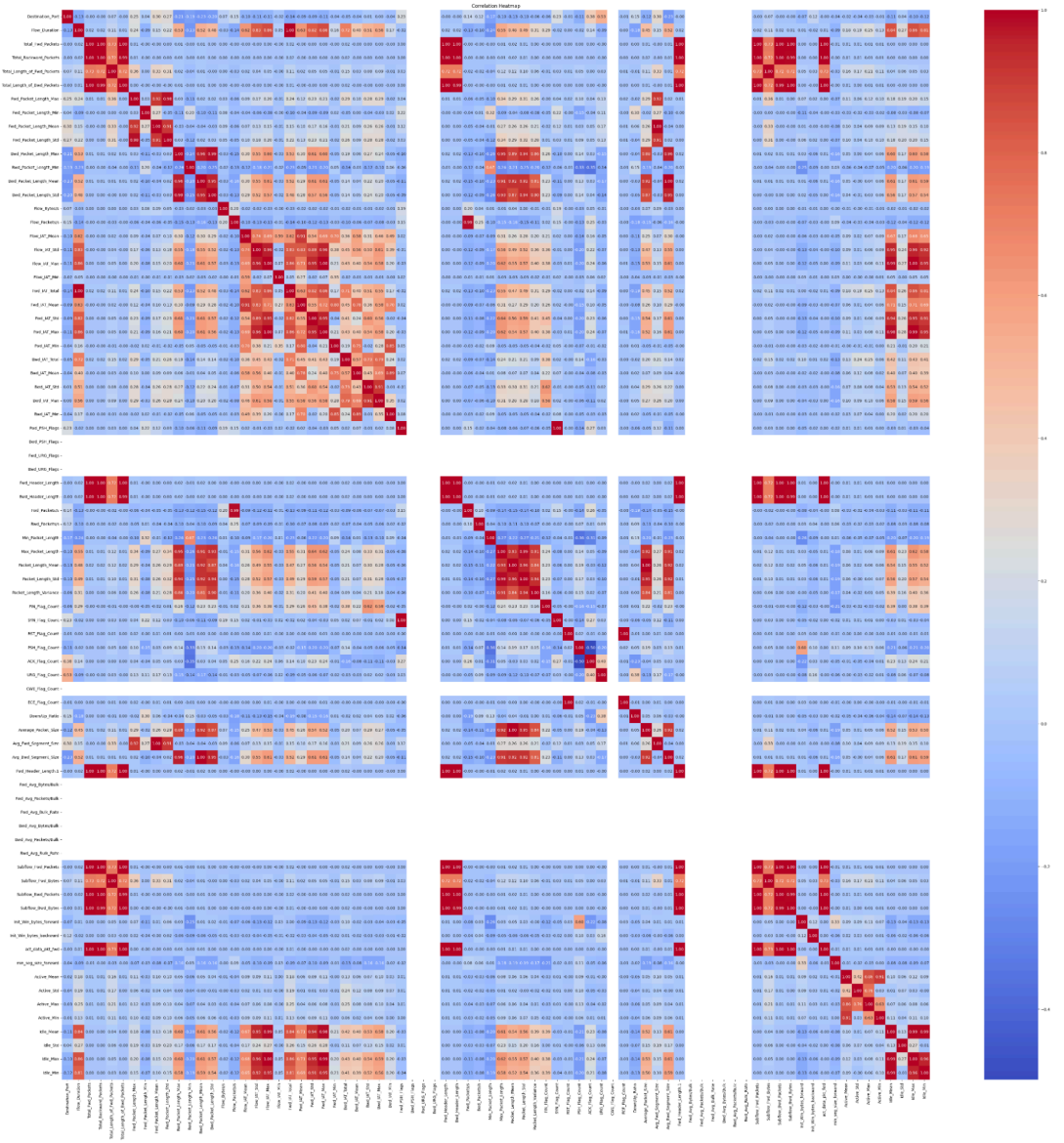
7. We also saw a perfect correlation between 6 pairs of columns. The following columns have the same values.

Column 1	Column 2
Total_Flow_Packets	Subflow_Fwd_Packets
Total_Backward_Packets	Subflow_Bwd_Packets
Total_Length_of_Fwd_Packets	Subflow_Fwd_Bytes
Fwd_Packet_Length_Mean	Avg_Fwd_Segment_Size
Fwd_PSH_Flags	SYN_Flag_Count
Fwd_Header_Length	Fwd_Header_Length.1

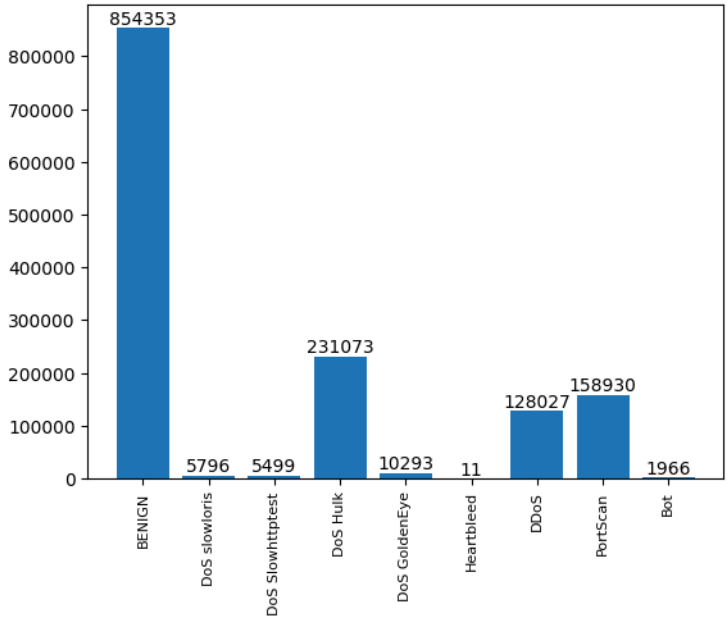
   We dropped one from each pair. Thus, 16 redundant columns can be dropped, so we now have 63 remaining as input.

8. **Standard Scaling—normalization:** Since the scale of the features varies greatly - we need to transform data into a common scale, so no single column dominates.
   Normalization: Scales numerical data to a range of 0 to 1 or -1 to 1. It's useful when the data's distribution is unknown or not normal. Normalization can reduce duplication and remove undesirable characteristics from a dataset. However, it's more likely to be affected by outliers because of its restricted range.
   Standardization: Scales data values so they have a mean of zero and a standard deviation of one. Standardization is less likely to be affected by outliers than normalization.
   We use standardization to scale the data.

9. Based on analyzing the correlation heatmap, we applied PCA to reduce the number of dimensions from 63 and to remove features of high correlation. We select the minimum number of components that explain 95% variance in our data.



10. **Principal Component Analysis:** We use the Principal Component Analysis algorithm to reduce features. The algorithm reduces redundant and unnecessary features while preserving crucial information. This reduces complexity of data, improves model performance due to reduced computational requirements. It can also help visualize data better by transforming it into a 2 or 3-D space. Moreover, it can also identify useful features.



11. **SMOTE:**
    As shown above, we can see that the dataset is highly imbalanced. We use SMOTE to deal with this issue. SMOTE stands for Synthetic Minority Oversampling Technique. It is widely used to deal with the issue of class imbalance in datasets. Class imbalance is a case where one class (the minority class) has very few samples in the dataset compared to another class (the majority class). This can lead to biased predictions by the machine-learning model trained on such data. SMOTE solves this problem by generating synthetic samples for the minority class. It does so by interpolating between existing minority class samples and their nearest neighbors. The algorithm selects a minority class sample and one of its K nearest neighbors, then generates a new sample along the line segment connecting them.

```
Original class distribution: Counter({'BENIGN': 1817189, 'DoS Hulk': 184342, 'PortScan': 126768, 'DDoS': 102311, 'DoS GoldenEye': 8219, 'FTP-
Patator': 6363, 'SSH-Patator': 4769, 'DoS slowloris': 4630, 'DoS Slowhttptest': 4390, 'Bot': 1533, 'Web Attack \ufffd Brute Force': 1200, 'Web Att
ack \ufffd XSS': 533, 'Infiltration': 29, 'Web Attack \ufffd Sql Injection': 15, 'Heartbleed': 9})
Class distribution after SMOTE: Counter({'BENIGN': 1817189, 'SSH-Patator': 1817189, 'DoS Hulk': 1817189, 'PortScan': 1817189, 'DDoS': 181718
9, 'DoS slowloris': 1817189, 'Web Attack \ufffd Brute Force': 1817189, 'DoS GoldenEye': 1817189, 'FTP-Patator': 1817189, 'DoS Slowhttptest': 18171
89, 'Bot': 1817189, 'Web Attack \ufffd XSS': 1817189, 'Infiltration': 1817189, 'Web Attack \ufffd Sql Injection': 1817189, 'Heartbleed': 1817189})
```

Machine Learning :

1. **Choice of models:**

    a. **Unsupervised: Gaussian Mixture Models (GMM):** A Gaussian mixture model is a soft clustering technique used in unsupervised learning to determine the probability that a given data point belongs to a cluster. It's composed of several Gaussians, each identified by $k \in \{1,\ldots, K\}$, where K is the number of clusters in a data set. GMMs are useful for unsupervised classification, feature engineering, and anomaly detection by identifying outliers based on low probabilities. They can effectively model multimodal data, making them suitable for applications such as image segmentation and density estimation. Their versatility across various fields, including computer vision and speech recognition, further enhances their utility in tackling diverse analytical challenges.

    b. **Logistic Regression:** Logistic regression is a statistical method used for binary classification tasks, where the outcome variable is categorical and typically takes on two values, such as "yes/no," "0/1," or "true/false." Logistic regression works by modeling the relationship between the independent variables (predictors) and the log-odds of the dependent

variable. The log-odds are transformed using the logistic (or sigmoid) function, which maps any real-valued number to a value between 0 and 1. This transformation enables logistic regression to predict probabilities, which can then be converted into binary outcomes using a threshold. Logistic regression would be a good starter point as a baseline model.

c. **Decision Trees:** A decision tree classifier is a machine learning algorithm that uses a tree-like structure to classify data based on its attributes. The tree has a root node, branches, internal nodes, and leaf nodes. The algorithm uses a divide-and-conquer strategy to recursively split the data into node segments until all records are classified. The algorithm uses the values of the data's attributes to make a class-label prediction. The leaf nodes indicate the class label. Decision trees are inexpensive to construct, fast at classifying unknown records, and simple to interpret. They also ensure a comprehensive analysis of the consequences of each branch. Due to the high number of features, Decision Trees could help explain the dependence of the output on the features.

d. **Random Forest:** A progression we wanted to experiment with was using Random Forests in addition to Decision Trees. Statistically Random Forests offer higher accuracy over Decision Trees and generalize better. This is because Random Forests are an ensemble of trees, where each tree is trained on a different sample of data, and a subset of features is considered at each split. They traditionally also give a better feature space resolution, resulting in a more refined decision boundary. We compare the performance of Random Forests with Decision Trees as well as with other models.

2. **Train Test Split:** We split the dataset into training and testing sets. 80% of the data is used for training, and 20% of the data is used for testing. We gradually increased the dataset through the course of this project, with this final report having twice the training samples than the midterm report.

3. **Model Training and Hyperparameter Tuning :** Hyperparameter tuning is the process of finding the optimal hyperparameters for a machine learning algorithm to maximize its performance. Hyperparameters are configuration variables that control the learning process and are set before training begins, unlike model parameters which are learned from data. The importance of hyperparameter tuning lies in its ability to:
    1. Improve model performance by optimizing the training process
    2. Enhance generalization to unseen data
    3. Balance the bias-variance tradeoff
    4. Reduce computational costs and training time

    The process typically involves defining a search space for hyperparameters, evaluating model performance using cross-validation, and selecting the best-performing configuration.
    To optimize our models for best performance, we fine tuned with different combinations of hyperparameters.

    For RandomForest, the best performing algorithm, we tuned the following parameters:
    1) n estimators (number of trees in the forest) More trees can capture more features and build a more robust model
    2) max depth : Maximum depth till which the decision tree splits data. A higher depth means more accuracy but can cause overfit- ting if the trees split too fine.

3) minimum samples split : Minimum samples needed in a node for it to be split
4) max_features : is the size of the random subsets of features to consider when splitting a node.

We used GridSearchCV for tuning hyperparameters. We ultimately chose the following combination : n_estimators = 100, max_depth = 70, minimum_samples = 4, and boot-strapping set to True.

4. **Model evaluation and metrics :**
Accuracy and F1 Score :
Accuracy measures the overall correctness of a model's predictions across all classes, calculated as (True Positives + True Negatives) / Total Predictions. Ranges from 0 to 1, with 1 being perfect accuracy. It is easy to interpret and works well for balanced datasets where classes are roughly equally represented. However, it can be misleading for imbalanced datasets and doesn't distinguish between types of errors (false positives vs false negatives).

F1 Score: Calculated as a harmonic mean of precision and recall. Also Ranges from 0 to 1, with 1 being the best score. Balances precision (minimizing false positives) and recall (minimizing false negatives). Useful for imbalanced datasets and when false positives/negatives have different costs. It penalizes extreme imbalances between precision and recall.

Results + Discussion[a]

A. Classification Report (Accuracy, Precision, Recall, F1-score)

```
Accuracy: 0.997756269714415

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    454131
           1       0.74      0.71      0.73       423
           2       1.00      1.00      1.00     25714
           3       0.97      0.98      0.98      2074
           4       1.00      1.00      1.00     45782
           5       0.98      0.99      0.99      1109
           6       0.98      0.99      0.99      1166
           7       0.99      1.00      1.00      1572
           8       1.00      1.00      1.00         2
           9       0.50      0.43      0.46         7
          10       0.99      0.99      0.99     32036
          11       0.99      0.99      0.99      1128
          12       0.75      0.73      0.74       307
          13       0.14      0.17      0.15         6
          14       0.39      0.42      0.41       119

    accuracy                           1.00    565576
   macro avg       0.83      0.83      0.83    565576
weighted avg       1.00      1.00      1.00    565576
```

Fig. Decision Tree Classification Report

```
Logistic Regression Results:
Accuracy: 0.7999698068463618

Classification Report:
              precision    recall  f1-score   support

           0       0.59      0.56      0.58    363800
           1       0.77      0.68      0.72    363313
           2       0.86      0.91      0.88    362463
           3       0.92      0.91      0.91    363154
           4       0.89      0.79      0.84    363596
           5       0.91      0.83      0.87    363435
           6       0.88      0.82      0.85    362507
           7       0.90      0.98      0.94    364214
           8       1.00      1.00      1.00    363565
           9       0.99      0.83      0.90    364155
          10       0.78      1.00      0.87    364219
          11       0.66      0.97      0.79    362813
          12       0.68      0.05      0.09    363633
          13       0.83      0.71      0.77    364030
          14       0.54      0.95      0.69    362670

    accuracy                           0.80   5451567
   macro avg       0.81      0.80      0.78   5451567
weighted avg       0.81      0.80      0.78   5451567
```

```
Random Forest Results:
Accuracy: 0.9983662673097868

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    454131
           1       0.84      0.65      0.73       423
           2       1.00      1.00      1.00     25714
           3       1.00      0.98      0.99      2074
           4       1.00      1.00      1.00     45782
           5       0.99      0.99      0.99      1109
           6       1.00      1.00      1.00      1166
           7       1.00      1.00      1.00      1572
           8       1.00      1.00      1.00         2
           9       1.00      0.14      0.25         7
          10       0.99      1.00      1.00     32036
          11       0.99      0.99      0.99      1128
          12       0.75      0.76      0.75       307
          13       1.00      0.17      0.29         6
          14       0.37      0.34      0.35       119

    accuracy                           1.00    565576
   macro avg       0.93      0.80      0.82    565576
weighted avg       1.00      1.00      1.00    565576
```

1. **Accuracy**
   Decision Tree: 99.8%
   Logistic Regression: 79.96%
   Random Forest: 99.83%
   Explanation:
   Accuracy measures the proportion of correctly classified instances out of the total instances. The Decision Tree has almost perfect accuracy (99.8%), while Logistic Regression achieves lower accuracy (~79.96%). Random Forest outperforms both these models.

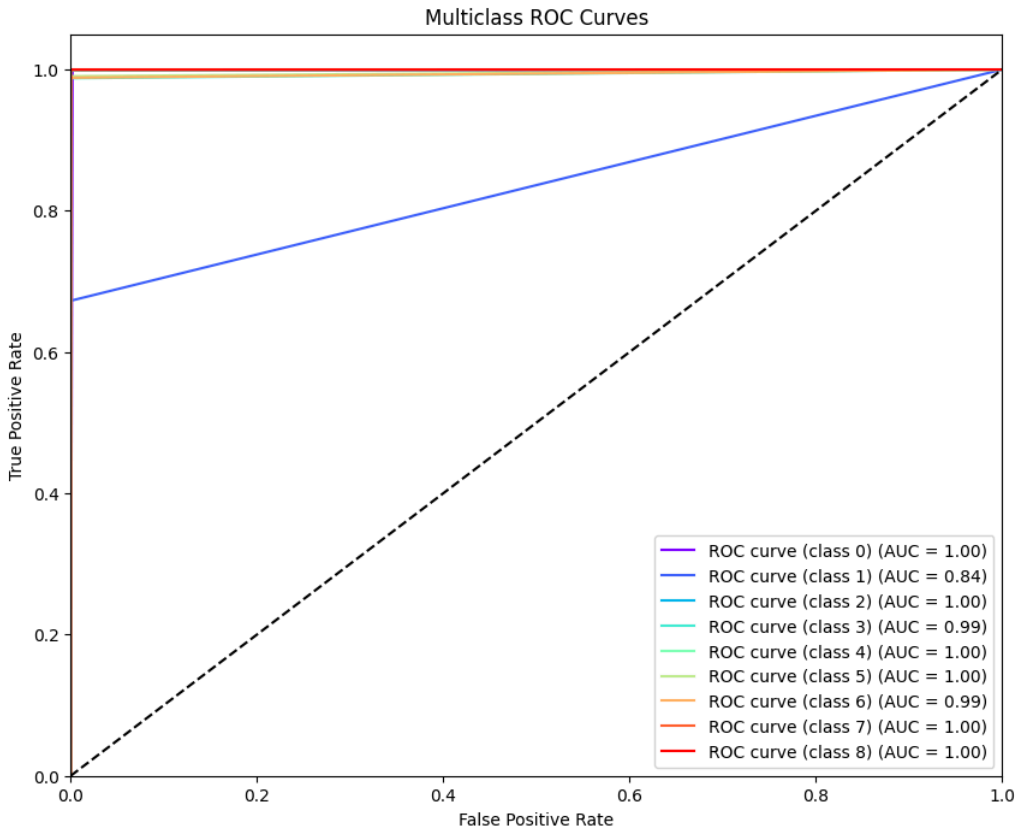
2. **Precision**
   Decision Tree (average): 0.83
   Logistic Regression (average): 0.81
   Random Forest (average): 0.93
   Explanation:
   Precision is the ratio of true positives to the sum of true positives and false positives. It tells us how many of the predicted positive instances were actually correct. Both models have similar average precision, indicating they are equally effective at avoiding false positives. Random forest outperforms the 2 models.

3. **Recall**
   Decision Tree (average): 0.83
   Logistic Regression (average): 0.81
   Random Forest (average): 0.8

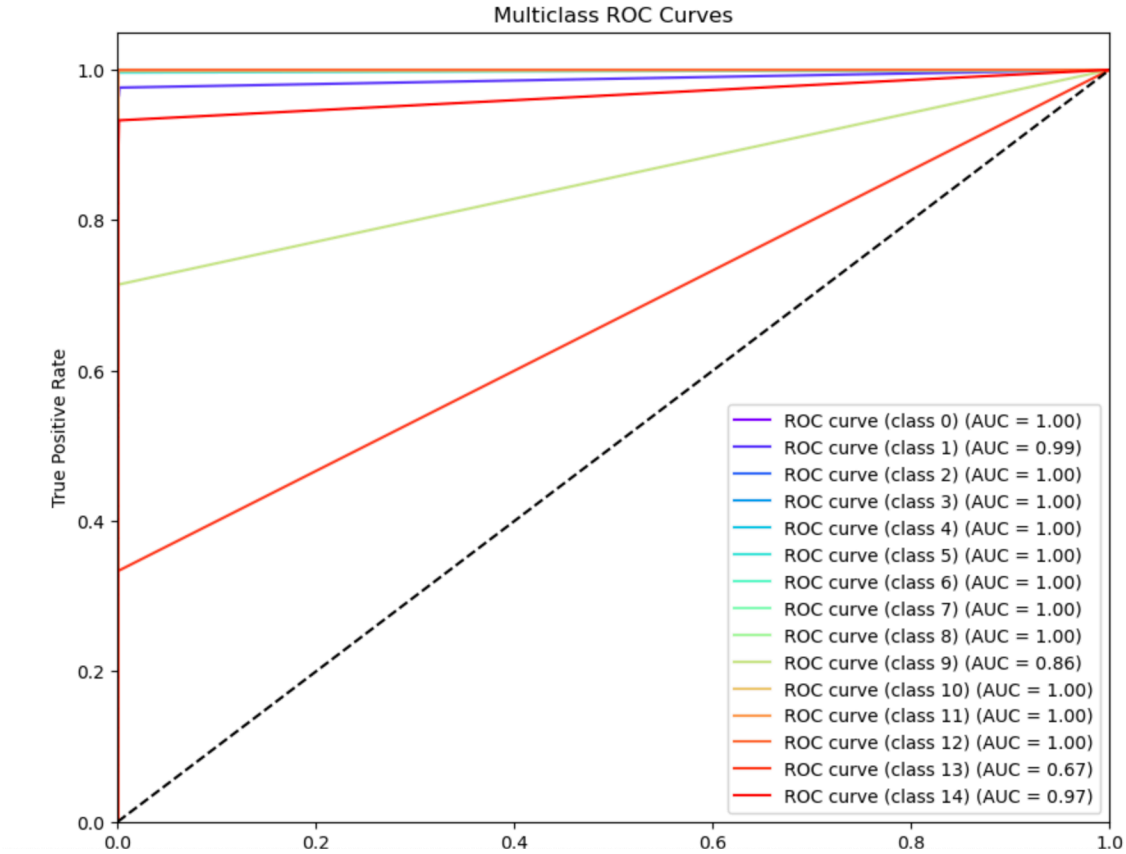
Explanation:
Recall (also known as sensitivity) is the ratio of true positives to the sum of true positives and false negatives. It measures how well the model identifies actual positive cases. The Decision Tree has a higher average recall (83%) compared to Logistic Regression (81%), meaning it is better at detecting positive cases. Random forest has comparable recall to the other 2 models.

4. **F1 Score**
   Decision Tree (average): 0.83
   Logistic Regression (average): 0.78
   Random Forest (average): 0.82
   Explanation:
   The F1 score is the harmonic mean of precision and recall, providing a balance between the two metrics. A higher F1 score indicates better overall performance in terms of both precision and recall. The Decision Tree has a higher F1 score (96%) compared to Logistic Regression (83%), indicating it performs better overall in balancing precision and recall.

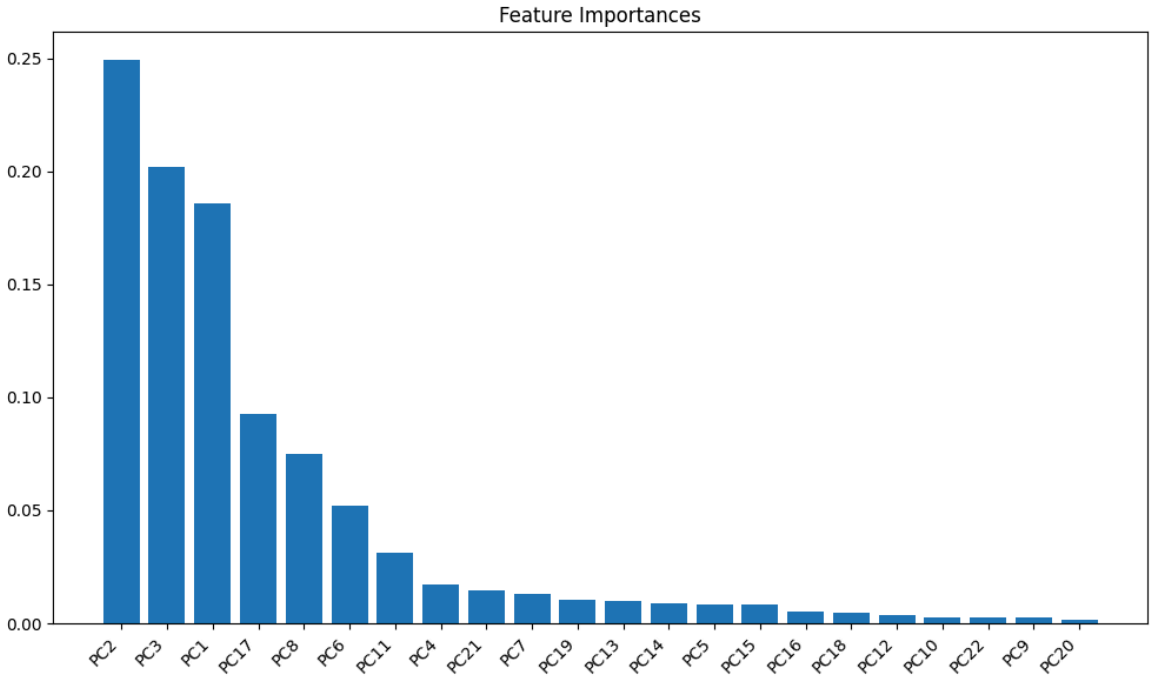
B. Multiclass ROC Curves



1. We selected the Random Forest algorithm to perform further analysis as it gave the best accuracy.
2. The ROC curves evaluate how well the model distinguishes between classes. Classes 0, 2, 4, 7, and 8 have an AUC (Area Under the Curve) of 1.00, indicating perfect classification.
3. Class 1 has an AUC of 0.84, showing that the model struggles more with this class compared to others.
4. Most other classes (e.g., Class 3 and Class 6) perform near-perfectly with AUC values around 0.99.



C. Feature Importance



1. The feature importance chart shows which features contribute most to the model's decisions.
2. PC2, PC3, and PC1 are the most important features, contributing significantly more than others.
3. After these top features, importance drops off sharply for features like PC17, with many features contributing very little.

D. Analysis & Comparison :

Model	Accuracy	F1-score
Linear Regression	79.9%	0.78
Decision Trees	99.77%	0.83
Random Forest	99.83%	0.82

For unsupervised learning we implemented a Gaussian Mixture Model (GMM), and tried to fit the dataset with 2 clusters, one ideally corresponding to the Benign samples, and the other cluster ideally representing all the classes corresponding to the attacks. Comparing the results with and without SMOTE, we could clearly see a high increase in the test accuracy, which showcased the positive impact of SMOTE and encouraged us to integrate it in our pipeline for usage by all models.

For the supervised learning models, we performed multi-class classification, where the Decision Tree model gave promising results, outperforming Linear Regression in terms of both Accuracy and F1-score, encouraging us to experiment with Random Forests as well. Random Forest, after careful finetuning, outperformed all other models as expected. In terms of training time, however, Random Forests took the most time since we kept n_estimators = 300.

Gantt Chart

1	**Project Proposal**					
1.1	Introduction and Background	Sahil	9/30/24	10/4/24	4	100%
1.2	Problem Definition	Harshit	9/30/24	10/4/24	4	100%
1.3	Methods	Aumkar	9/30/24	10/4/24	4	100%
1.4	Potential Datasets	Harshit	9/30/24	10/4/24	4	100%
1.5	Potential Results and Discussion	Archit	9/30/24	10/4/24	4	100%
1.6	Video Creation and Recording	Aumkar	9/30/24	10/4/24	4	100%
1.7	GitHub Page	Sahil	9/30/24	10/4/24	4	100%
1.8	Peer Review	All	9/30/24	10/4/24	4	100%
2	**Midpoint Report**					
2.1	Method 1 (M1) - Unsupervised Algorithm - Design and Selection	Sahil	10/7/24	10/20/24	14	100%
2.2	M1 Data Cleaning	Harshit	10/7/24	10/20/24	14	100%
2.3	M1 Data Visualization	Aumkar	10/7/24	10/20/24	14	100%
2.4	M1 Feature Reduction	Harshit	10/7/24	10/20/24	14	100%
2.5	M1 Implementation & Coding	Archit	10/7/24	10/20/24	14	100%
2.6	M1 Results and Evaluation	Aumkar	10/7/24	10/20/24	14	100%
2.7	Method 2 (M2) - Supervised Algorithm - Design and Selection	Sahil	10/21/24	11/4/24	14	100%
2.8	M2 Data Cleaning	Archit	10/21/24	11/4/24	14	100%
2.9	M2 Data Visualization	Harshit	10/21/24	11/4/24	14	100%
2.1	M2 Feature Reduction	Aumkar	10/21/24	11/4/24	14	100%
2.11	M2 Implementation & Coding	Harshit	10/21/24	11/4/24	14	100%
2.12	M2 Results and Evaluation	Sahil	10/21/24	11/4/24	14	100%
2.13	Midpoint Report Preparation	All	11/6/24	11/7/24	1	100%
2.14	Peer Review	All	11/8/24	11/8/24	1	100%
3	**Final Project Presentation**					
3.1	Method 3 (M3) - Supervised Algorithm - Design and Selection	Archit	11/11/24	11/15/24	5	100%
3.2	M3 Data Cleaning	Sahil	11/11/24	11/15/24	5	100%
3.3	M3 Data Visualization	Harshit	11/11/24	11/15/24	5	100%
3.4	M3 Feature Reduction	Aumkar	11/11/24	11/15/24	5	100%
3.5	M3 Implementation & Coding	Harshit	11/11/24	11/15/24	5	100%
3.6	M3 Results and Evaluation	Archit	11/11/24	11/15/24	5	100%
3.7	Implementation of application	Sahil	11/16/24	11/25/24	10	100%
3.8	Final Presentation Preparation	All	11/25/24	12/1/24	6	100%
3.9	Peer Review	All	12/3/24	12/3/24	1	100%

Contribution Table

Team Member	**Proposal Contributions**
Sahil Samantary	Data pre-processing, Decision Tree algorithm implementation, brainstorming
Harshit Gupta	Random Forest Model, GMM model, graph generation, project report, ideation
Archit Vikas Shinde	Logistic regression algorithm, project report, dataset and GitHub repo management
Aumkar Makarand Gadekar	Data visualizations, website, exploratory data analysis

References

[1] Intrusion detection evaluation dataset (CIC-IDS2017)
[2] Yulianto, Arif, Parman Sukarno, and Novian Anggis Suwastika. "Improving adaboost-based intrusion detection system (IDS) performance on CIC IDS 2017 dataset." Journal of Physics: Conference Series. Vol. 1192. IOP Publishing, 2019.
[3] Towards Model Generalization for Intrusion Detection: Unsupervised Machine Learning Techniques
[4] Talukder, M.A., Islam, M.M., Uddin, M.A. *et al.* Machine learning-based network intrusion detection for big and imbalanced data using oversampling, stacking feature embedding and feature extraction. *J Big Data* **11**, 33 (2024). https://doi.org/10.1186/s40537-024-00886-w

[a]ToDo: Add GMM results: with and without SMOTE