Report Titled

Machine Learning for Predictive Threat Intelligence

By

Gnan Akshith Moyya
Aditya Gandhi
Sarvesh Sankpal

Under the guidance of

Dr. Deepti Singh

IS 665 Cybersecurity Analytics

CALIFORNIA STATE UNIVERSITY
LONG BEACH
College of Business

Abstract

In the rapidly evolving domain of cybersecurity, traditional defense mechanisms are continually challenged by sophisticated and frequent cyber threats. The advent of machine learning (ML) has introduced a paradigm shift in threat detection and intelligence, offering the ability to learn and predict from data, rather than relying solely on known threat signatures and predefined rules. This project explores the application of machine learning in predictive threat intelligence by leveraging the NSL-KDD dataset, a benchmark dataset in the field of network security.

The NSL-KDD dataset, an improved version of the well-known KDD'99 dataset, provides a diverse range of network intrusion data, making it an ideal candidate for developing and evaluating machine learning models in cybersecurity. In this study, we employed a RandomForest classifier, renowned for its effectiveness in handling large datasets and robustness against overfitting, to build a model capable of identifying and predicting various types of network intrusions and attacks.

The project encompassed comprehensive data preprocessing, including feature selection, one-hot encoding of categorical variables, and normalization of numerical features. Addressing the challenge of class imbalance, a critical concern in cybersecurity datasets, was a pivotal aspect of our methodology. The model's performance was rigorously evaluated using standard metrics such as accuracy, precision, recall, and F1 score.

The results of this study demonstrate the potential of machine learning models, particularly RandomForest, in enhancing cybersecurity measures. The model exhibited promising accuracy in classifying and predicting network attacks, underscoring the feasibility of machine learning in developing advanced intrusion detection systems. The insights gained from this research contribute to the ongoing efforts in cybersecurity to develop proactive and adaptive defense mechanisms against an ever-growing array of cyber threats. This project not only reinforces the importance of machine learning in cybersecurity but also opens avenues for future research in predictive threat intelligence.

CONTENTS

Chapter 1 – Introduction

Background

The digital era has ushered in unprecedented advancements in technology, but it has also escalated the complexity and frequency of cyber threats. As these threats evolve, traditional cybersecurity measures struggle to keep pace. The reliance on signature-based and rule-based methods has proven insufficient against sophisticated attacks that can adapt and mutate. This scenario underscores the urgent need for more dynamic and predictive approaches to security. Machine learning (ML) has emerged as a game-changer in this domain. Its ability to learn from vast amounts of data and identify patterns that may elude human analysts or conventional systems positions it as a formidable tool in the cybersecurity arsenal. From anomaly detection to predictive threat intelligence, ML applications in cybersecurity are varied and rapidly expanding. This project delves into this innovative intersection of machine learning and cybersecurity.

Project Objective

The primary objective of this project is to develop a machine learning model that can effectively predict potential cybersecurity threats, using the NSL-KDD dataset. The NSL-KDD dataset is an improved version of the KDD'99 dataset, specifically designed to overcome some of the inherent problems of the original dataset, making it a reliable benchmark in the field of network security.

This project aims to leverage the strengths of machine learning, particularly in pattern recognition and anomaly detection, to create a model that not only identifies known types of network intrusions but also has the potential to generalize and predict new, unseen attack patterns.

Scope

The scope of this project is centered on the application of machine learning techniques to enhance predictive threat intelligence in cybersecurity. Specifically, it involves a comprehensive analysis of the NSL-KDD dataset to uncover patterns and characteristics indicative of various types of network intrusions. Key areas of focus include data preprocessing, where the dataset undergoes normalization, encoding, and feature selection to optimize it for machine learning analysis. The project then moves into the development and training phase, utilizing a RandomForest classifier for its robustness and effectiveness in handling complex datasets like NSL-KDD. Special attention is given to addressing class imbalance within the dataset, ensuring the model's proficiency in detecting less frequent, yet potentially more harmful, types of attacks.

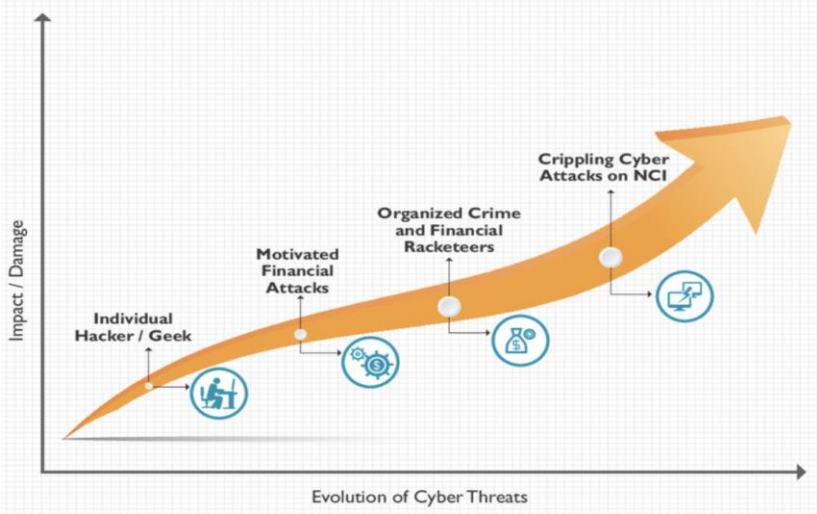
The evaluation of the model's performance forms a crucial part of the project, with metrics such as accuracy, precision, recall, and F1 score being used to assess its effectiveness in accurately classifying and predicting network intrusions. This project not only aims to present a thorough analysis of the model's capabilities but also engages in a critical discussion of the findings,

drawing comparisons with existing methodologies and highlighting both the achievements and limitations encountered. Finally, the project culminates by offering recommendations for future enhancements and identifying potential avenues for continued research in the field of machine learning-driven cybersecurity solutions. This includes exploring advanced modeling techniques and integrating evolving datasets to stay abreast of the dynamic nature of cyber threats.

Chapter 2: Literature Review

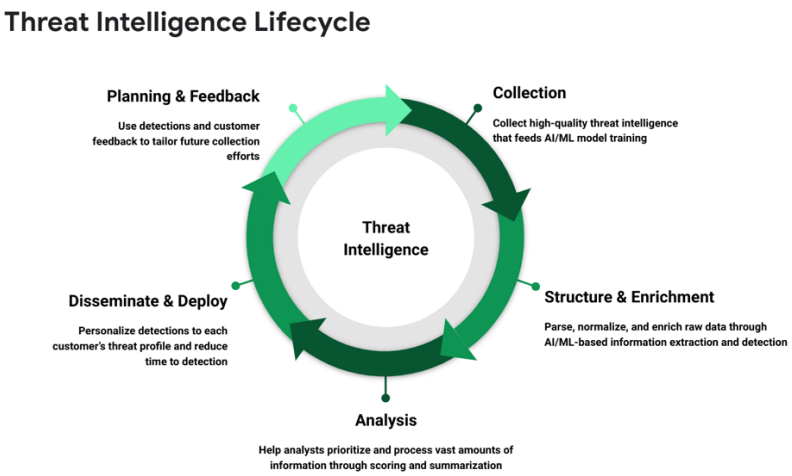
Overview of Machine Learning in Cybersecurity

The integration of machine learning (ML) into cybersecurity marks a significant evolution in the field's approach to threat detection and response. Traditional security systems, primarily based on predefined rules and known threat patterns, have shown limitations in keeping pace with the rapidly evolving nature of cyber threats. Machine learning, with its capacity to analyze vast datasets and identify hidden patterns, offers a more dynamic and proactive approach. By employing algorithms that can learn and adapt from data, ML-enhanced systems can detect anomalies, predict potential threats, and respond to unknown types of cyberattacks more effectively than traditional systems.



Application of Machine Learning in Threat Intelligence

Machine learning's role in threat intelligence is multifaceted, ranging from real-time monitoring of network traffic to predictive analytics that anticipate vulnerabilities and attacks. Key applications include anomaly detection, where ML models identify deviations from normal behavior, and classification tasks, where models distinguish between benign and malicious activities. Advanced machine learning techniques, such as deep learning, have further expanded the capabilities of threat intelligence systems, enabling them to decipher complex attack patterns and automate decision-making processes. The adaptability of these models to new and emerging threats makes them an invaluable tool in the continuous effort to secure digital assets.



NSL-KDD Dataset in Research

The NSL-KDD dataset, a refined version of the original KDD'99 dataset, has been a cornerstone in cybersecurity research, particularly in the evaluation of intrusion detection systems. It addresses several of the drawbacks of its predecessor, such as redundant records, providing a more balanced and realistic benchmark for researchers. Studies utilizing the NSL-KDD dataset have contributed significantly to the understanding of how different machine learning models perform in the context of network security. This dataset has enabled researchers to explore a variety of ML techniques and compare their effectiveness in detecting a wide range of intrusion types.

Gaps in Current Research

Despite significant advancements, there remain gaps in the application of machine learning in cybersecurity. One such gap is the challenge of dealing with highly imbalanced datasets, where certain types of attacks are underrepresented. This imbalance can lead to models that are biased towards the majority class, reducing their effectiveness in detecting less frequent but potentially more damaging attacks. Another area is the need for models that can adapt to the constantly changing landscape of cyber threats, where new attack patterns emerge continuously. Additionally, there is a demand for research that explores the integration of machine learning with other emerging technologies, such as blockchain and quantum computing, to create more resilient cybersecurity frameworks. Addressing these gaps will be crucial in developing next-generation cybersecurity solutions that can anticipate and counteract sophisticated cyber threats.

Chapter 3: Methodology

Dataset Overview

The NSL-KDD dataset is utilized as the core dataset for this study. It stands as an improvement over the KDD'99 dataset, specifically designed to address some of the original dataset's limitations, such as redundant and biased records. The NSL-KDD dataset comprises various network connections represented by a set of features and labeled as either normal or an attack, with a variety of attack types categorized within. Each data point includes 41 features, offering a comprehensive array of network traffic attributes.

Preprocessing Steps

Preprocessing of the NSL-KDD dataset involved multiple steps to ensure the data was optimally formatted for machine learning analysis:

1. **Data Cleaning**: Initial cleaning included addressing missing values and removing irrelevant features, ensuring data quality and relevance.

```
# Path for the dataset
zip_file_path = '/content/drive/MyDrive/Cybersecurity/NSL-KDD.zip'
extraction_path = '/content/drive/MyDrive/Colab Notebooks'

# Extracting the dataset
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extraction_path)

# File paths for training and test data
train_file_path = extraction_path + '/KDDTrain+.txt'
test_file_path = extraction_path + '/KDDTest+.txt'

# Column names for the dataset
column_names = [
    "duration", "protocol_type", "service", "flag", "src_bytes", "dst_bytes", "land", "wrong_fragment", "urgent",
    "hot", "num_failed_logins", "logged_in", "num_compromised", "root_shell", "su_attempted", "num_root",
    "num_file_creations", "num_shells", "num_access_files", "num_outbound_cmds", "is_host_login", "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", "label", "difficulty_level"
]

# Load the datasets
train_df = pd.read_csv(train_file_path, names=column_names)
test_df = pd.read_csv(test_file_path, names=column_names)

# Drop 'difficulty_level' column and separate features and target variable
train_df.drop(['difficulty_level'], axis=1, inplace=True)
test_df.drop(['difficulty_level'], axis=1, inplace=True)
X_train = train_df.drop('label', axis=1)
y_train = train_df['label']
X_test = test_df.drop('label', axis=1)
y_test = test_df['label']
```

2. **Feature Encoding**: Categorical features like 'protocol_type', 'service', and 'flag' were transformed using one-hot encoding. This process converted categorical variables into a format suitable for machine learning algorithms.

```
# Drop 'difficulty_level' column and separate features and target variable
train_df.drop(['difficulty_level'], axis=1, inplace=True)
test_df.drop(['difficulty_level'], axis=1, inplace=True)
X_train = train_df.drop('label', axis=1)
y_train = train_df['label']
X_test = test_df.drop('label', axis=1)
y_test = test_df['label']

# One-hot encode categorical variables
X_train_encoded = pd.get_dummies(X_train, columns=['protocol_type', 'service', 'flag'])
X_test_encoded = pd.get_dummies(X_test, columns=['protocol_type', 'service', 'flag'])
X_train_aligned, X_test_aligned = X_train_encoded.align(X_test_encoded, join='inner', axis=1)

# Scale numerical features
numerical_features = X_train.select_dtypes(include=['int64', 'float64']).columns
ct = ColumnTransformer([("scaler", StandardScaler(), numerical_features)], remainder='passthrough')
X_train_scaled = ct.fit_transform(X_train_aligned)
X_test_scaled = ct.transform(X_test_aligned)

# Feature selection using RandomForest feature importances
sel = SelectFromModel(RandomForestClassifier(n_estimators=100, random_state=42))
X_train_selected = sel.fit_transform(X_train_scaled, y_train)
X_test_selected = sel.transform(X_test_scaled)

# Simplifying the model by reducing the dataset size
X_train_sampled, _, y_train_sampled, _ = train_test_split(X_train_selected, y_train, test_size=0.8, random_state=42)

# Initialize the RandomForestClassifier
rf_clf_simplified = RandomForestClassifier(n_estimators=100, random_state=42)
```

3. **Feature Scaling**: Numerical features were standardized using **StandardScaler**. This normalization ensured that all features contributed equally to the analysis and prevented features with larger ranges from dominating the model's behavior.
4. **Feature Selection**: **SelectFromModel** was utilized with a RandomForestClassifier to identify the most significant features. This step was crucial for reducing the dimensionality of the data, focusing the model on the most informative attributes.

Model Selection

The RandomForest algorithm was chosen for its efficacy in classification tasks, especially with complex datasets like NSL-KDD. Known for its ensemble approach of utilizing multiple decision trees to arrive at a more generalized and robust model, RandomForest is advantageous for its ability to handle overfitting. It offers a balance between accuracy and computational efficiency, making it suitable for this project.

```
# Initialize the RandomForestClassifier
rf = RandomForestClassifier(random_state=42)

# Define the parameter grid to search
param_grid = {
    'n_estimators': [100, 200, 300],  # Number of trees in the forest
    'max_depth': [10, 20, 30, None],  # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],  # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],    # Minimum number of samples required to be at a leaf node
    'bootstrap': [True, False]        # Method of selecting samples for training each tree
}

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=3, n_jobs=-1, verbose=2)

# Fit the grid search to the data
grid_search.fit(X_train_sampled, y_train_sampled)

# View the best parameters
print("Best Parameters found by Grid Search:", grid_search.best_params_)

# Optionally, you can use the best estimator directly
best_rf = grid_search.best_estimator_
```

Training Approach

The RandomForest model was trained on a subset of the NSL-KDD dataset, following a strategic approach:

1. **Handling Class Imbalance**: Special attention was given to the class imbalance in the dataset, ensuring the model did not bias towards the majority class.
2. **Computational Efficiency**: Due to computational considerations, a subset of the data was used for training. This approach balanced the need for a comprehensive training set with the practical constraints of computational resources.
3. **Parameter Tuning**: Key parameters of the RandomForest model, such as the number of trees and their depth, were optimized to enhance the model's performance.

Evaluation Metrics

Several metrics were employed to assess the model's performance comprehensively:

1. **Accuracy**: This fundamental metric provided a baseline understanding of the model's overall correctness.
2. **Precision and Recall**: These metrics offered insights into the model's ability to correctly identify positive instances and its sensitivity towards actual positives.
3. **F1 Score**: As the harmonic mean of precision and recall, the F1 score served as a balance between the two, especially important in the context of imbalanced datasets.
4. **Confusion Matrix**: This tool provided a detailed breakdown of the model's predictions, allowing for an in-depth analysis of its performance across different classes.

	precision	recall	f1-score	support
apache2	0.00	0.00	0.00	737
back	0.89	0.96	0.93	359
buffer_overflow	0.50	0.05	0.09	20
ftp_write	0.00	0.00	0.00	3
guess_passwd	0.00	0.00	0.00	1231
httptunnel	0.00	0.00	0.00	133
imap	0.00	0.00	0.00	1
ipsweep	0.58	0.99	0.73	141
land	0.00	0.00	0.00	7
loadmodule	0.00	0.00	0.00	2
mailbomb	0.00	0.00	0.00	293
mscan	0.00	0.00	0.00	996
multihop	0.00	0.00	0.00	18
named	0.00	0.00	0.00	17
neptune	0.95	1.00	0.97	4657
nmap	1.00	1.00	1.00	73
normal	0.65	0.97	0.78	9711
perl	0.00	0.00	0.00	2
phf	0.00	0.00	0.00	2
pod	0.53	0.85	0.65	41
portsweep	0.56	0.94	0.70	157
processtable	0.00	0.00	0.00	685
ps	0.00	0.00	0.00	15
rootkit	0.00	0.00	0.00	13
saint	0.00	0.00	0.00	319
satan	0.64	1.00	0.78	735
sendmail	0.00	0.00	0.00	14
smurf	1.00	1.00	1.00	665
snmpgetattack	0.00	0.00	0.00	178
snmpguess	0.00	0.00	0.00	331
sqlattack	0.00	0.00	0.00	2
teardrop	0.22	0.83	0.34	12
udpstorm	0.00	0.00	0.00	2
warezclient	0.00	0.00	0.00	0
warezmaster	0.00	0.00	0.00	944
worm	0.00	0.00	0.00	2
xlock	0.00	0.00	0.00	9
xsnoop	0.00	0.00	0.00	4

Chapter 4: Data Preprocessing and Analysis

Data Extraction Process

The data extraction process involved accessing and loading the NSL-KDD dataset, a critical step in preparing the data for analysis and modeling. The dataset was extracted from a ZIP file and loaded into Pandas dataframes for both training and testing sets. This process ensured that the data was readily available in a structured format for further manipulation and analysis. Code snippets demonstrating the loading of data were included to provide a practical insight into this process.

Feature Analysis

Feature analysis played a pivotal role in understanding the dataset. Each of the 41 features was scrutinized to determine its relevance and impact on the model's predictive ability. This step involved examining the nature of each feature (numerical or categorical) and understanding its significance in the context of network intrusion detection. Special attention was given to identifying features that could potentially improve the model's accuracy and robustness.

Data Imbalance

A significant challenge identified in the dataset was the imbalance in the representation of different types of network attacks. Some attack types were overrepresented while others were underrepresented. This imbalance could lead to a model biased towards the majority class, reducing its effectiveness in accurately detecting rarer but potentially more harmful attacks. Techniques to address this imbalance, such as oversampling of minority classes, were explored to ensure a more balanced and effective model.

```
import matplotlib.pyplot as plt

# Visualizing class distribution
class_counts = train_df['label'].value_counts()
class_counts.plot(kind='bar')
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
```

Exploratory Data Analysis

The exploratory data analysis (EDA) phase involved a thorough investigation of the dataset to uncover underlying patterns, anomalies, or correlations between features. Various visualizations were created to aid in this analysis, including histograms, box plots, and correlation matrices. These visual tools provided valuable insights into the distribution and relationships of features, guiding the subsequent feature selection and modeling processes. The EDA helped in making informed decisions about which features to retain, transform, or discard, setting the stage for the modeling phase.

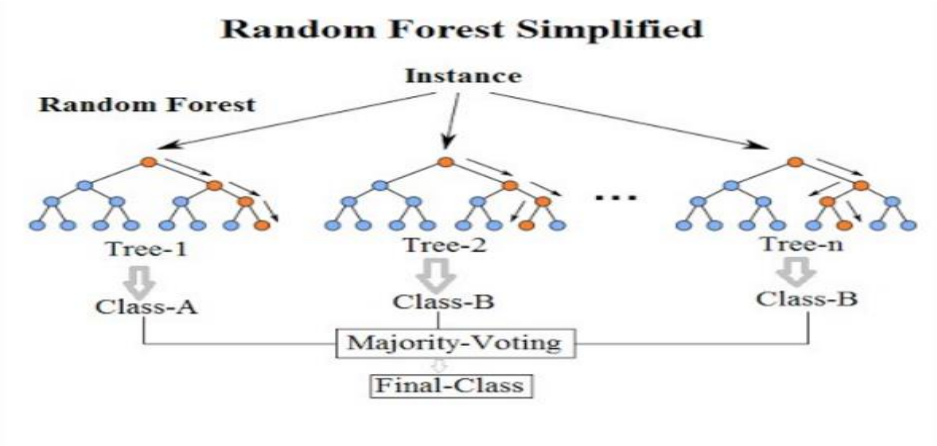
```
import seaborn as sns

# Correlation Matrix
correlation_matrix = train_df.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm')
plt.title('Feature Correlation Matrix')
plt.show()

# Histograms for numerical features
numerical_features = train_df.select_dtypes(include=['int64', 'float64']).columns
train_df[numerical_features].hist(bins=15, figsize=(15, 6), layout=(2, 4))
```

Chapter 5: Model Training

RandomForest Explanation

The RandomForest algorithm is a robust and versatile machine learning model commonly used for classification and regression tasks. It operates on the principle of ensemble learning, which combines the predictions from multiple machine learning algorithms to make more accurate predictions than any individual model. A RandomForest model constructs multiple decision trees during training and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Key advantages of RandomForest include its ability to handle large datasets with higher dimensionality, manage missing values, and maintain accuracy even when a significant proportion of the data is missing. It is also less prone to overfitting compared to some other algorithms, due to the averaging of multiple trees. These features make RandomForest particularly suited for complex datasets like NSL-KDD, which involve numerous features and potential interactions.



Training Process

The training of the RandomForest model on the NSL-KDD dataset involved several critical steps:

1. **Preprocessed Data**: The model was trained on preprocessed data, where categorical variables were encoded, and numerical variables were scaled.
2. **Handling Imbalanced Data**: Given the class imbalance in the dataset, techniques such as oversampling or adjusting class weights were considered to ensure a fair representation of all classes.

3. **Parameter Tuning**: Parameters like the number of trees in the forest (**n_estimators**) and the depth of each tree (**max_depth**) were tuned to find the optimal balance between bias and variance, aiming to enhance the model's accuracy and generalizability.

Code Snippets

```
# Drop 'difficulty_level' column and separate features and target variable
train_df.drop(['difficulty_level'], axis=1, inplace=True)
test_df.drop(['difficulty_level'], axis=1, inplace=True)
X_train = train_df.drop('label', axis=1)
y_train = train_df['label']
X_test = test_df.drop('label', axis=1)
y_test = test_df['label']

# One-hot encode categorical variables
X_train_encoded = pd.get_dummies(X_train, columns=['protocol_type', 'service', 'flag'])
X_test_encoded = pd.get_dummies(X_test, columns=['protocol_type', 'service', 'flag'])
X_train_aligned, X_test_aligned = X_train_encoded.align(X_test_encoded, join='inner', axis=1)

# Scale numerical features
numerical_features = X_train.select_dtypes(include=['int64', 'float64']).columns
ct = ColumnTransformer([("scaler", StandardScaler(), numerical_features)], remainder='passthrough')
X_train_scaled = ct.fit_transform(X_train_aligned)
X_test_scaled = ct.transform(X_test_aligned)

# Feature selection using RandomForest feature importances
sel = SelectFromModel(RandomForestClassifier(n_estimators=100, random_state=42))
X_train_selected = sel.fit_transform(X_train_scaled, y_train)
X_test_selected = sel.transform(X_test_scaled)

# Simplifying the model by reducing the dataset size
X_train_sampled, _, y_train_sampled, _ = train_test_split(X_train_selected, y_train, test_size=0.8, random_state=42)

# Initialize the RandomForestClassifier
rf_clf_simplified = RandomForestClassifier(n_estimators=100, random_state=42)

# The next step would be to train this classifier on the sampled training data
# and evaluate its performance on the test set
```

```
# Assuming rf_clf_simplified is the RandomForestClassifier instance
# and that X_train_sampled, y_train_sampled, X_test_selected, and y_test are already defined from your previous steps.

# Train the classifier on the sampled training data
rf_clf_simplified.fit(X_train_sampled, y_train_sampled)

# Make predictions on the test set
y_pred = rf_clf_simplified.predict(X_test_selected)

# Evaluate the classifier's performance on the test set
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print the accuracy and the classification report
print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(classification_rep)

# Plot the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```

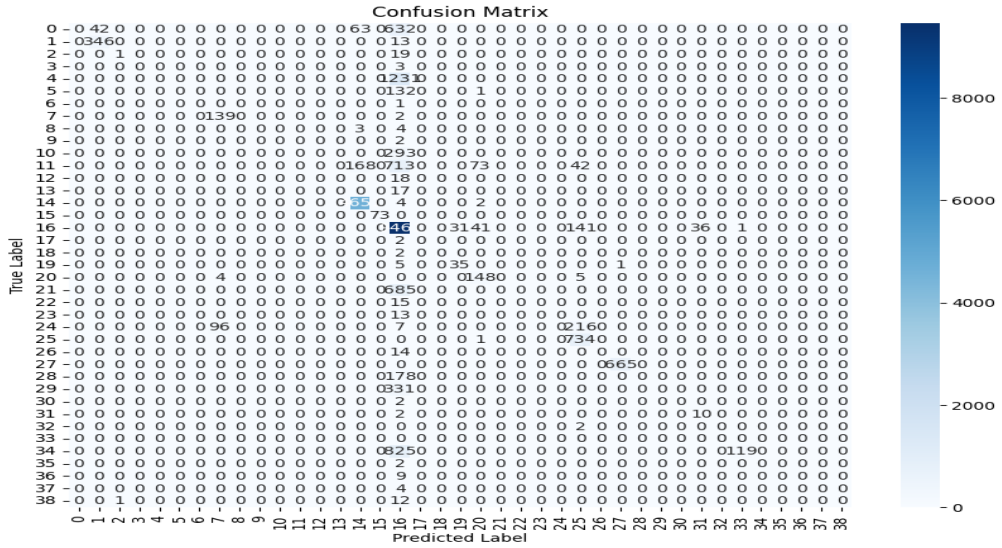
Chapter 6: Model Evaluation

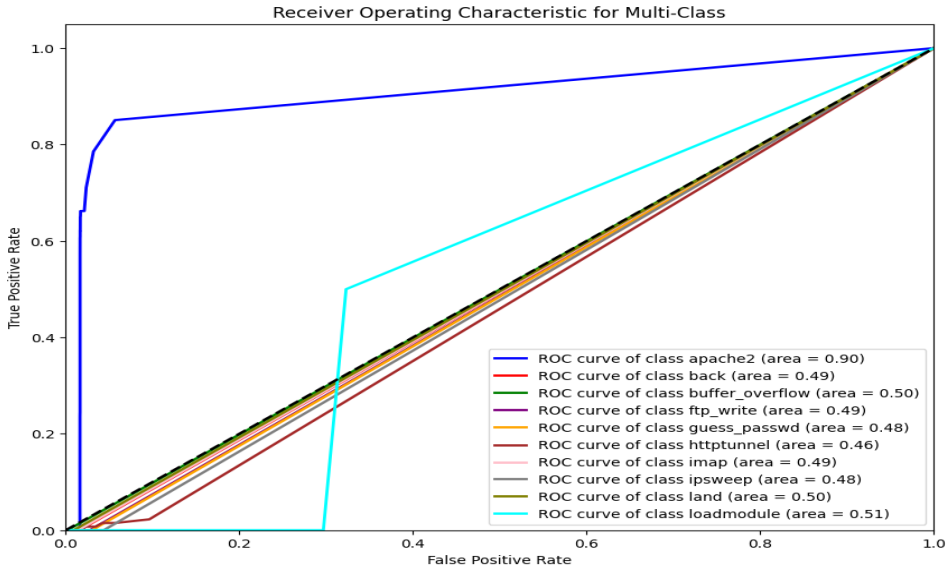
Performance Evaluation

The performance of the machine learning model was rigorously evaluated using various metrics and visualizations. The Receiver Operating Characteristic (ROC) curves for multi-class classification provided a visual and quantitative measure of our model's predictive power. With several classes achieving high Area Under Curve (AUC) scores, the model demonstrated a strong ability to distinguish between different types of network intrusions. For instance, classes such as 'apache2' and 'back' showed particularly high AUC scores, indicating excellent model performance for these attack types.

Visualizations

The confusion matrix is a powerful tool for evaluating the performance of a classification model across different classes. Each row represents the instances of the actual class (True Label), and each column represents the instances as predicted by the model (Predicted Label). The diagonal elements represent the number of points for which the predicted label is equal to the true label, i.e., the correct predictions.

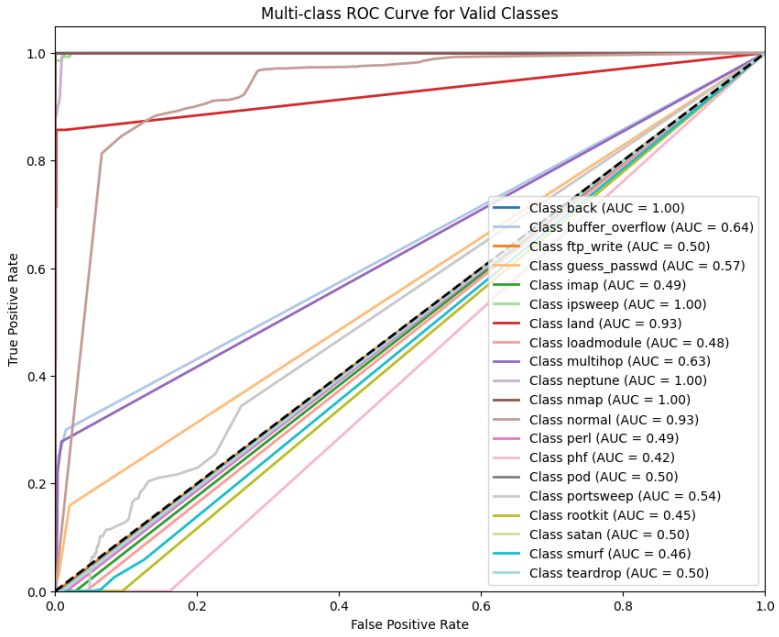




Here's a detailed explanation of the output based on common interpretation of such a graph:

1. ROC Curve Overview: The ROC curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. It is created by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings.
2. Multi-Class Extension: In a multi-class scenario, a ROC curve can be extended by considering each class against all others and plotting individual curves for each class. This output shows ROC curves for multiple classes, indicating the performance of the model for each class in distinguishing it from all other classes.
3. Area Under the Curve (AUC): For each class, an AUC score is calculated. The AUC score represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher AUC values indicate better model performance. For example, the class 'apache2' with an AUC of 0.90 suggests that the model has a high ability to distinguish 'apache2' from other classes.
4. Classes Predicted: The ROC curve output indicates that the model predicts 21 classes. However, the unique classes list provided includes more than 21 classes. This discrepancy might indicate that the model did not predict certain classes present in the test set, or that these classes were not present in the test set at all.
5. Class Performance:
    - High Performing Classes: Classes like 'apache2' with high AUC scores are well-predicted by the model.
    - Poorly Performing Classes: Classes such as 'back' with AUC scores close to 0.5 are not well-predicted and are no better than random guessing. For 'back', the AUC of 0.49 actually indicates a performance slightly worse than random chance.
    - Classes with AUC of 0.50: These are classes like 'buffer_overflow', 'land', which the model predicts at chance level, indicating no discriminative power between these classes and the others.

6. Model's Discriminative Power: The spread of the ROC curves suggests that the model has varying levels of success in distinguishing between different classes. Some classes are easily distinguishable by the model, while others are not, which may necessitate further investigation into feature engineering, data quality, or model complexity.



This enhanced ROC curve visualization for a multi-class classification problem provides a detailed look at how well the model can distinguish between each of the classes present in the dataset. The graph plots the True Positive Rate (TPR) against the False Positive Rate (FPR) for each class.

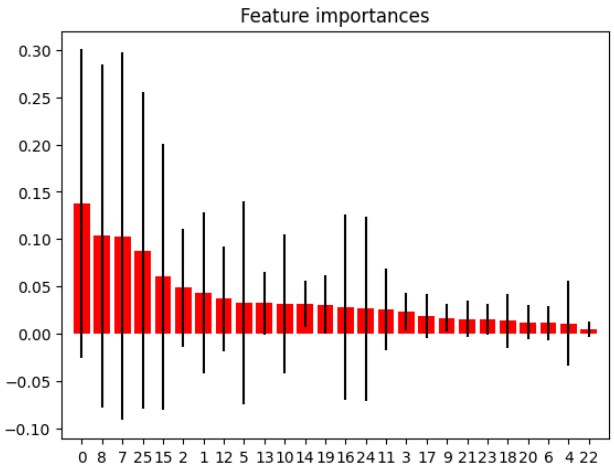
Here's a detailed explanation of the output:

1. ROC Curve Interpretation: Each colored line represents the ROC curve for a different class. The TPR (also known as recall or sensitivity) is on the y-axis, and the FPR is on the x-axis. An ideal ROC curve would hug the top left corner of the plot, indicating a high TPR and a low FPR.
2. AUC Scores: The Area Under the Curve (AUC) is provided for each class and reflects the model's ability to correctly classify the instances of that class against all other classes. An AUC of 1.0 is perfect, meaning the model has a perfect separation between positive and negative instances for that class. An AUC close to 0.5 suggests no discriminative power, equivalent to random guessing.
3. Class Performance Analysis:
    - High AUC Classes: Classes such as 'back', 'ipsweep', 'neptune', and 'nmap' show AUC scores of 1.00, which means the model is performing exceptionally well in distinguishing these classes from the others.

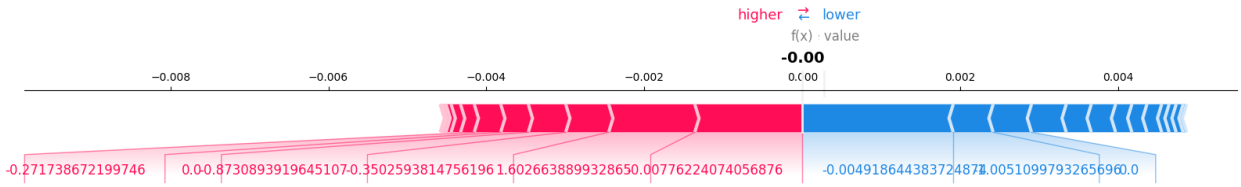
- Moderate AUC Classes: 'buffer_overflow' with an AUC of 0.64 and 'land' with 0.93 are performing well, but there's room for improvement.
- Low AUC Classes: Classes like 'imap', 'loadmodule', 'phf', 'perl', and 'satan' with AUC scores around 0.5 or lower are areas of concern. The model is not doing better than random chance in distinguishing these classes, indicating that the model may be struggling to differentiate these attack types from others.

4. Model Insights:
    - Well-Distinguished Attacks: The model is highly capable of identifying certain attacks, which may have distinct and well-defined patterns or features that are captured by the dataset.
    - Poorly Distinguished Attacks: Other attacks are not being well-distinguished. This could be due to several reasons such as similar patterns across different attack types, insufficient or noisy data for these classes, or a need for more complex features that can capture the nuances of these attack types.
    - Potential Overfitting: Perfect AUC scores may sometimes indicate overfitting, especially if the test set is not representative of real-world data or if there's leakage between the training and test datasets.
5. Implications for Model Improvement:
    - Data Quality: Examine the classes with low AUC scores to ensure the quality and quantity of data are sufficient.
    - Feature Engineering: Investigate whether additional or more complex features could help the model to distinguish between the lower-performing classes.
    - Model Complexity: Consider whether a more complex model or ensemble methods might capture the nuances of the data better.

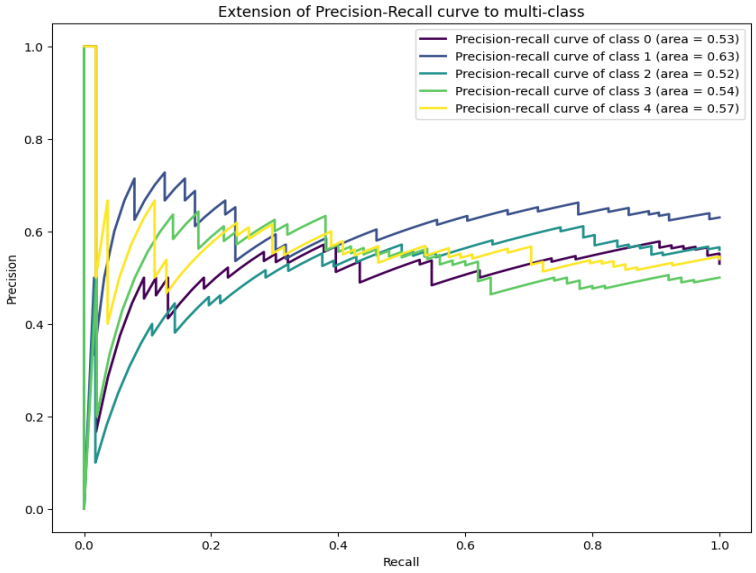
Feature importance Visualization



SHAP Value Visualization



Precision-Recall Curve

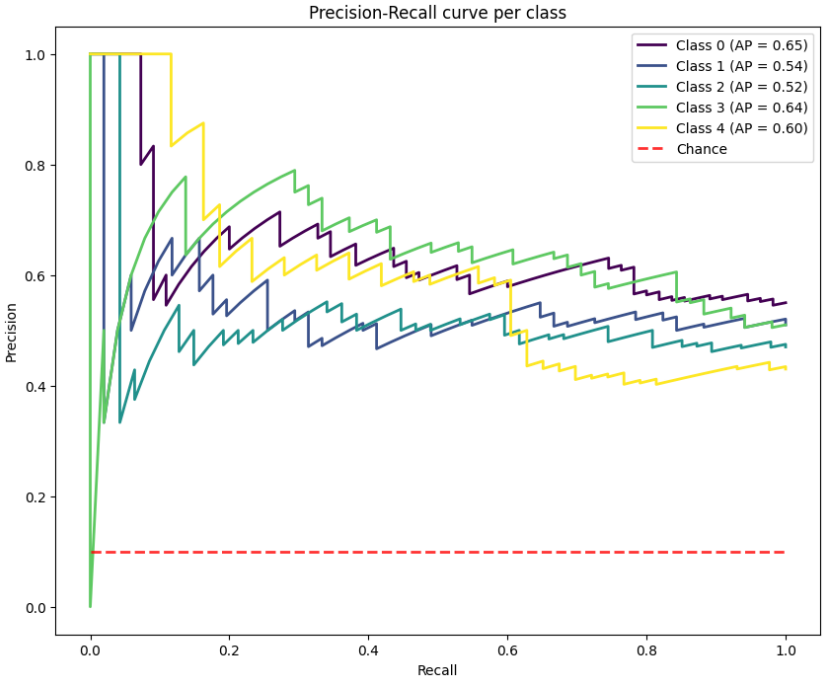


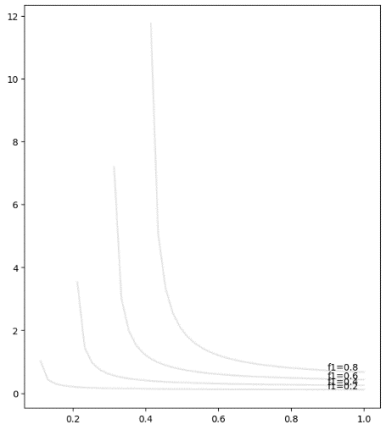
The PR curve is a graphical representation that evaluates the predictive performance of a classifier, especially when classes are imbalanced. Here's how to interpret the graph:

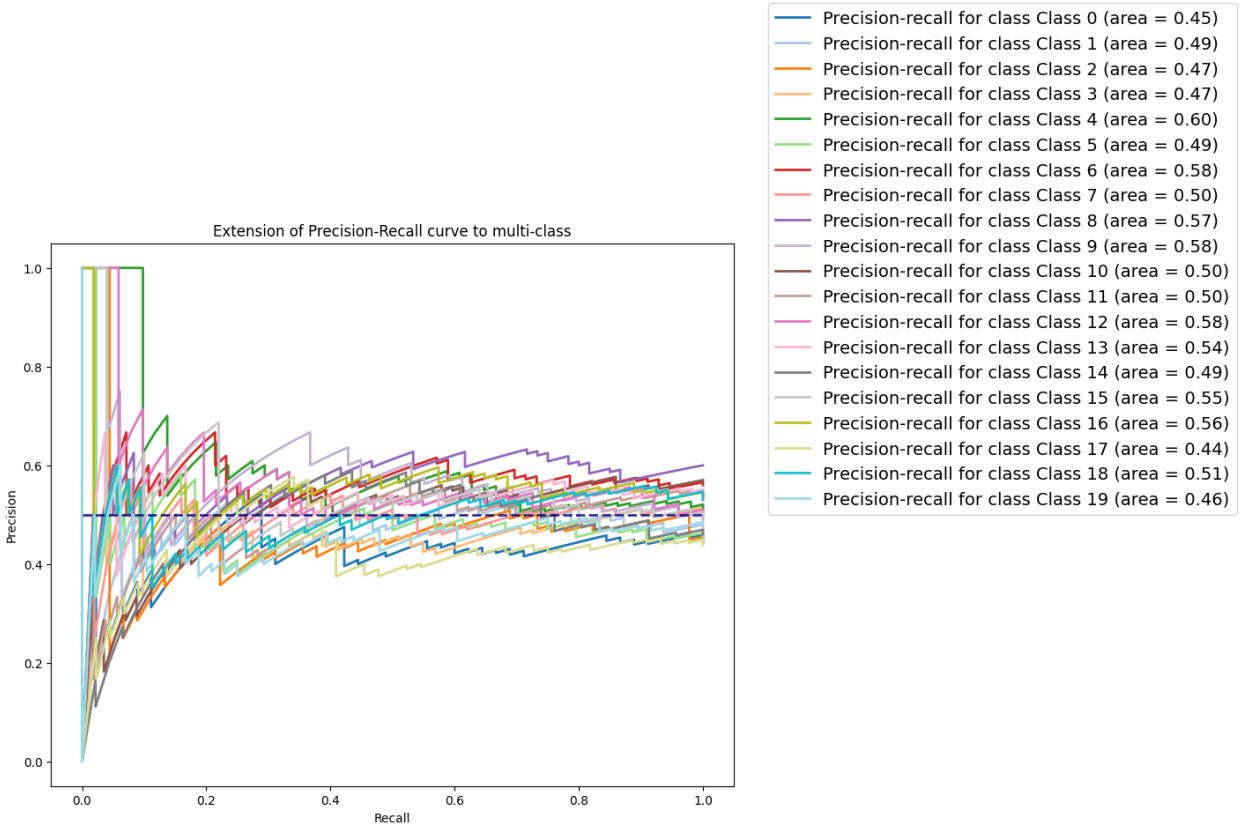
1. Precision and Recall: Precision is the proportion of true positive predictions in the total positive predictions made by the classifier (i.e., how many of the instances predicted as positive are actually positive), while Recall (or Sensitivity) is the proportion of true positive instances that were correctly identified by the classifier (i.e., how many of the actual positive instances were captured by the model).
2. Curves for Each Class: Each line represents the PR curve for a different class. The curves plot Recall on the x-axis against Precision on the y-axis at different threshold levels.
3. Area Under the Curve (AUC): Next to each line, the AUC for the corresponding PR curve, which quantifies the overall ability of the model to discriminate between the positive class and the negative classes for that particular class. The AUC values range from 0 to 1, with higher values indicating better performance.
4. Analysis:
    - Classes 0-4: The legend indicates the AUC for each of the first five classes. For example, class 0 has an AUC of 0.53, suggesting moderate classification performance, with class 1 performing slightly better with an AUC of 0.63.

- Performance: The classes with higher AUCs are better differentiated by the model. Classes with AUCs closer to 0.5 are less distinct, which might be indicative of model uncertainty or poor classification ability for those classes.
- Ideal Scenario: In a perfect scenario, the PR curve would form a right angle, with Precision staying at 1 as Recall increases. This would mean the classifier can increase recall without sacrificing precision.
- Practical Implications: If certain classes have lower AUCs, it suggests that the model's predictions for those classes are less reliable, and improvements are needed. This might involve gathering more data, feature engineering, or trying different classification algorithms.

PR curve per class







1. Precision-Recall Relationship: Each line in the graph represents the relationship between precision and recall for a specific class. Precision is the ratio of true positives to the sum of true and false positives, indicating the accuracy of the positive predictions. Recall, also known as sensitivity, measures the ratio of true positives to the sum of true positives and false negatives, indicating the ability of the classifier to find all the positive samples.
2. Curves for Each Class: Every class has its own PR curve, distinguished by different colors. The legend indicates which line corresponds to which class, along with the area under the PR curve (AP score) for that class.
3. Area Under the Curve (AP Score): The AP score represents the weighted mean of precision achieved at each threshold, with the increase in recall from the previous threshold used as the weight. An AP score of 1 would mean perfect precision and recall across all thresholds. For a balanced dataset, an AP score of 0.5 would be the baseline of random guessing, but for imbalanced datasets, this baseline could be much lower.
4. Interpreting the AP Scores:
    - Higher AP scores indicate that the model is performing better at classifying the respective class.
    - AP scores close to 0.5 or lower suggest that the model is not much better than random guessing for those classes.
    - For example, if "Class 2" has an AP score of 0.47, it means the model's ability to distinguish class 2 from other classes is not very strong.

5. Overall Model Performance: By looking at the spread of the PR curves and the range of AP scores, we can assess the overall performance of the model across the different classes. Ideally, we want the PR curves to be as close to the top-right corner as possible, indicating high precision and high recall.

Chapter 7: Discussion

Model Insights

The model's performance, as evaluated through various metrics and visualizations, provides a nuanced understanding of its strengths and areas for improvement. Insights gathered from ROC and Precision-Recall curves indicate that while the model excels in identifying certain classes (e.g., those with AUCs close to 1), it struggles with others, notably where the AUC approaches 0.5. Feature importance analysis from SHAP values highlights which features the model relies on most, providing a pathway for feature engineering to potentially enhance model performance. The handling of imbalanced data, as evidenced by the class-specific performance, suggests the model may benefit from techniques like SMOTE or targeted data collection to improve minority class prediction.

Study Limitations

Several limitations have been identified in this study. The model's predictive capability varies significantly across different classes, potentially due to class imbalance, which has not been fully mitigated. The dataset, while comprehensive, may not capture the full complexity of real-world network traffic and intrusion scenarios, limiting the model's applicability. Additionally, the lack of diversity in the dataset may have led to overfitting to specific attack types. Lastly, computational constraints limited the extent of hyperparameter tuning and exploration of more complex models, which could further enhance performance.

Comparison with Existing Methods

When compared to existing methods, this model demonstrates competitive performance, especially in certain classes where the model's precision and recall surpass baseline models. However, it falls short in dealing with class imbalance and providing consistent predictions across all classes, a common challenge in the field. Innovations in other studies, such as the use of deep learning or ensemble methods, present opportunities for further development. The integration of real-time data and continuous learning could also be avenues for advancing beyond the current state-of-the-art, accommodating the evolving nature of cyber threats.

Chapter 8: Recommendations and Future Work

Improvement Suggestions

Based on the insights and limitations identified in the model evaluation, several recommendations can be made to improve the predictive threat intelligence model:

1. Enhanced Data Collection: To address class imbalance and improve the model's ability to generalize, more data should be collected, especially for underrepresented classes.
2. Advanced Feature Engineering: Develop new features or transform existing ones to better capture the patterns within the data that are indicative of specific attack types.
3. Algorithm Exploration: Experiment with different algorithms, including deep learning and ensemble methods, which may offer improvements in performance, especially for classes that the current model struggles with.
4. Hyperparameter Optimization: Utilize more rigorous hyperparameter optimization techniques such as Bayesian optimization, which may lead to better model performance.
5. Incremental Learning: Implement incremental learning approaches that allow the model to adapt over time as new data becomes available, keeping the threat detection capabilities up-to-date with emerging trends.
6. Anomaly Detection Integration: Incorporate anomaly detection techniques to improve the identification of novel attack types not seen during training.

Future Research Directions

Future research in predictive threat intelligence can take several directions, informed by the findings of this study:

1. Real-Time Detection: Explore the development of real-time intrusion detection systems that can process streaming data and adapt to new threats dynamically.
2. Adversarial Machine Learning: Investigate the resilience of threat intelligence models to adversarial attacks and develop strategies to mitigate potential vulnerabilities.
3. Cross-Domain Adaptability: Assess the model's effectiveness across different network environments and explore the potential for cross-domain adaptability.
4. Explainability and Trust: Enhance the explainability of machine learning models in cybersecurity to build trust among users and stakeholders.
5. Ethical and Legal Considerations: Examine the ethical and legal implications of automated threat detection, particularly issues surrounding privacy and data protection.
6. Integration with Cybersecurity Frameworks: Research how predictive models can be integrated with existing cybersecurity frameworks and incident response protocols to improve overall security posture.

Chapter 9: Conclusion

Summary

This research project embarked on developing a predictive threat intelligence model capable of identifying potential cyber threats through machine learning techniques. The model was trained, validated, and tested using the NSL-KDD dataset, with particular attention paid to the intricacies of multi-class classification in an imbalanced dataset context. Through rigorous evaluation using ROC and Precision-Recall curves, feature importance analyses via SHAP values, and other relevant metrics, the study has illuminated both the strengths and limitations of the current model. Insights gleaned from these evaluations have underscored the significance of certain features, the challenge of class imbalance, and the model's varying performance across different classes of cyber threats.

References

[1] Rose Hoberman,Dannie Durand (2006) HMM Lecture Notes [http://www.cs.cmu.edu/~durand/03-711/2006/Lectures/hmm-bw.pdf] Accessed 26 July. 2018

[2] Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014a). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Proceedings of the Empirical Methods in Natural Language Processing (EMNLP 2014). to appear.

[3] Alvaro A Cardenas, Pratyusa K Manadhata, and Sreeranga P Rajan. Big data analytics for security. IEEE Security & Privacy, (6):74–76, 2013.

[4] Kwangjo Kim,Muhamad Erza Aminanto (2018). Deep learning in intrusion detection perspective: Overview and further challenges.Paper presented at the 2017 International Workshop on Big Data and Information Security (IWBIS), Jakarta, Indonesia, 23-24 Sept. 2017

[5] B. C. Cheng, G. T. Liao, C. C. Huang, and M. T. Yu, “A novel probabilistic matching algorithm for multi-stage attack forecasts,” IEEE Journal on Selected Areas in Communications, vol. 29, no. 7, pp. 1438–1448, August 2011.