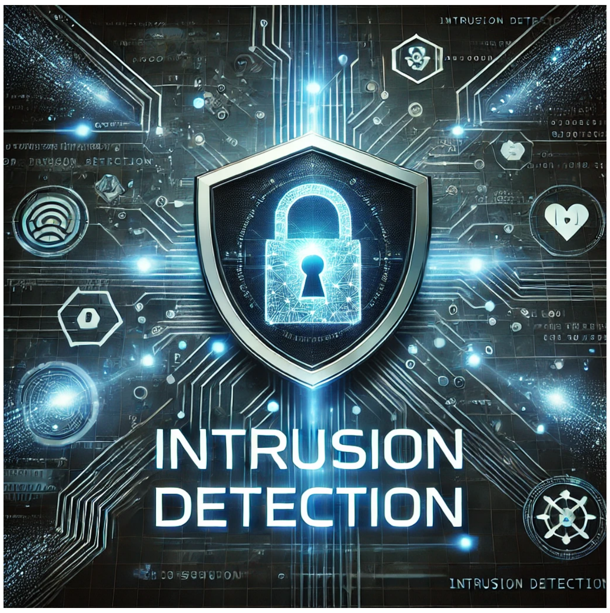
Intrusion Detection Analysis Using the CIC-IDS2017 Dataset



A Comprehensive Machine Learning Approach

March 21, 2025

Contents

Chapter 1

Introduction

Chapter Overview

This chapter introduces the scope, objectives, and significance of the research on network intrusion detection using the CIC-IDS2017 dataset.

The increasing sophistication of network attacks has become a significant concern for cybersecurity professionals worldwide. With the evolution of attack vectors and techniques, traditional security measures often prove insufficient, necessitating more robust Intrusion Detection Systems (IDS). Modern IDS solutions must be capable of detecting novel attacks while maintaining low false positive rates—a challenging balance that requires advanced machine learning approaches.

This report focuses on the comprehensive analysis of the CIC-IDS2017 dataset, which represents one of the most recent and realistic network traffic datasets available for cybersecurity research. Containing over 2.8 million records across 79 features, this dataset provides a realistic testbed for evaluating the effectiveness of various machine learning algorithms in detecting network intrusions.

The dataset encompasses normal traffic as well as seven distinct attack types:

- **Brute Force:** Attempts to gain unauthorized access by trying various password combinations
- **Heartbleed:** Exploits of the OpenSSL vulnerability that allows memory data leakage
- **Botnet:** Traffic generated by compromised devices under centralized control
- **DoS (Denial of Service):** Attacks that overwhelm services to make them unavailable
- **DDoS (Distributed Denial of Service):** Coordinated DoS attacks from multiple sources
- **Web Attack:** Various attacks targeting web applications, including SQL injection and XSS
- **Infiltration:** Sophisticated attacks that breach network defenses undetected

1.1 Research Objectives

The primary objectives of this study are:

- To perform extensive exploratory data analysis (EDA) to understand the dataset's characteristics, feature distributions, and relationships between variables.
- To implement and evaluate various machine learning models, focusing on both traditional algorithms and more advanced ensemble techniques.
- To compare model performances using rigorous metrics and cross-validation techniques to ensure reliability.
- To address key challenges in the IDS domain, especially in the context of data preprocessing, feature selection, and dimensionality reduction.
- To develop a robust, deployable solution that can be integrated with existing network infrastructure.

1.2 Significance of the Study

This research contributes to the field of network security in several ways:

- It provides a thorough evaluation of state-of-the-art machine learning algorithms on a contemporary dataset that reflects modern attack patterns.
- It addresses the critical challenge of dimensionality reduction in network traffic analysis—a key aspect for real-time implementation.
- It establishes a reproducible methodology for developing machine learning-based IDS solutions that can adapt to evolving threat landscapes.
- It offers practical insights into the integration challenges when deploying ML models in production environments.

The remainder of this report is organized as follows: Chapter 2 details the exploratory data analysis and preprocessing steps. Chapter 3 presents the implemented machine learning models and their evaluation. Chapter 4 discusses the challenges encountered and the solutions proposed. Finally, Chapter 5 concludes the study and outlines directions for future work.

Chapter 2

Exploratory Data Analysis and Data Preprocessing

Chapter Overview

This chapter examines the CIC-IDS2017 dataset characteristics, describes the exploratory analysis process, and details the preprocessing steps implemented to prepare the data for machine learning algorithms.

2.1 Dataset Overview

The CIC-IDS2017 dataset, developed by the Canadian Institute for Cybersecurity, represents one of the most comprehensive and realistic network traffic datasets available for intrusion detection research. It was created to address the limitations of older datasets like KDD-99 and NSL-KDD, which no longer reflect contemporary network environments and attack patterns.

Key characteristics of the CIC-IDS2017 dataset include:

- **Volume:** Over 2.8 million records capturing network traffic over multiple days.
- **Features:** 79 diverse attributes capturing different aspects of network traffic, including packet-level statistics, flow-level information, and time-based features.
- **Class Imbalance:** A dominant benign class compared to seven minority attack classes, reflecting real-world network conditions where malicious traffic is typically rare compared to legitimate traffic.
- **Realism:** Traffic generated in a controlled but realistic environment using genuine attack tools and standard network protocols.

2.2 Exploratory Data Analysis (EDA)

A thorough exploratory data analysis was conducted to understand the dataset's characteristics before developing any machine learning models. This process involved:

2.2.1 Statistical Summaries

Statistical analysis revealed significant variations in the distributions of different features. For example:

- Flow duration showed extreme variability, with values ranging from microseconds to several minutes.
- Packet counts and byte counts exhibited heavy-tailed distributions.
- Several features contained outliers that could potentially impact model performance.

Feature	**Count**	**Mean**	**Std Dev**	**Min**	**Max**
Flow Duration	2,830,743	489,353.05	3,298,625.92	0.00	120,000,000.00
Total Fwd Packets	2,830,743	13.38	25.66	1.00	3,000.00
Total Backward Packets	2,830,743	11.86	28.96	0.00	3,000.00
Total Length of Fwd Packets	2,830,743	7,190.22	29,431.28	0.00	1,500,000.00
Total Length of Bwd Packets	2,830,743	7,874.44	66,372.42	0.00	3,000,000.00

Table 2.1: Statistical Summary of Key Features

2.2.2 Visualizations

Multiple visualization techniques were employed to better understand the dataset:

- **Histograms:** Revealed the distributions of individual features, highlighting skewness and potential outliers.
- **Boxplots:** Provided a clear view of feature ranges and outliers across different attack categories.
- **Correlation Matrices:** Identified relationships between features and potential multicollinearity issues.
- **PCA Visualizations:** Helped understand the separability of different attack classes in reduced dimensional space.

Figure 2.1: Class Distribution (Log Scale) in CIC-IDS2017 Dataset

2.2.3 Class Imbalance Analysis

A critical finding from the EDA was the severe class imbalance present in the dataset:

- Benign traffic constituted approximately 83.3% of all samples.
- Some attack types (e.g., Heartbleed) had fewer than 20 instances in the entire dataset.
- This imbalance necessitated careful consideration during model training to avoid bias toward the majority class.

2.3 Data Preprocessing

Based on the insights gained from the EDA, several preprocessing steps were implemented to prepare the data for machine learning algorithms:

2.3.1 Handling Missing Values

Missing values were detected in several features, requiring a systematic approach:

- Features with more than 30% missing values were considered for removal.
- For features with fewer missing values, imputation strategies were employed:
  - Numerical features: Median imputation to minimize the influence of outliers.
  - Categorical features: Mode imputation to use the most frequent value.
- A small subset of records with critical missing values across multiple important features was removed entirely.

2.3.2 Encoding Categorical Variables

The dataset contained a mix of numerical and categorical features:

- Protocol types (TCP, UDP, ICMP) were converted using one-hot encoding.
- Binary flags (e.g., FIN, SYN, RST) were already numerically encoded (0/1).
- The target variable (attack type) was encoded using label encoding for multi-class classification.

2.3.3 Feature Scaling

Feature scaling was essential due to the wide range of values across different features:

- Min-Max normalization was applied to bring all features to the [0,1] range:

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- This scaling was critical for distance-based algorithms and gradient-based optimization methods.

2.3.4 Dimensionality Reduction

Two approaches for dimensionality reduction were thoroughly evaluated:

Random Forest-based Feature Selection

Initially, Random Forest-based feature selection appeared promising:

- Feature importance scores were computed using a Random Forest classifier.
- Features were ranked based on these scores, and the top 20 features were selected.
- However, this approach was found to cause overfitting due to its sensitivity to the specific characteristics of the training data.
- In cross-validation tests, models using this feature selection method showed high variance across different data splits.

Principal Component Analysis (PCA)

PCA proved to be a more robust approach:

- PCA was applied to reduce the 79-dimensional feature space to 25 principal components.
- These components captured approximately 95% of the variance in the original data.
- PCA provided consistent performance across different data splits, indicating better generalizability.
- The transformation was independent of the target variable, reducing the risk of data leakage.

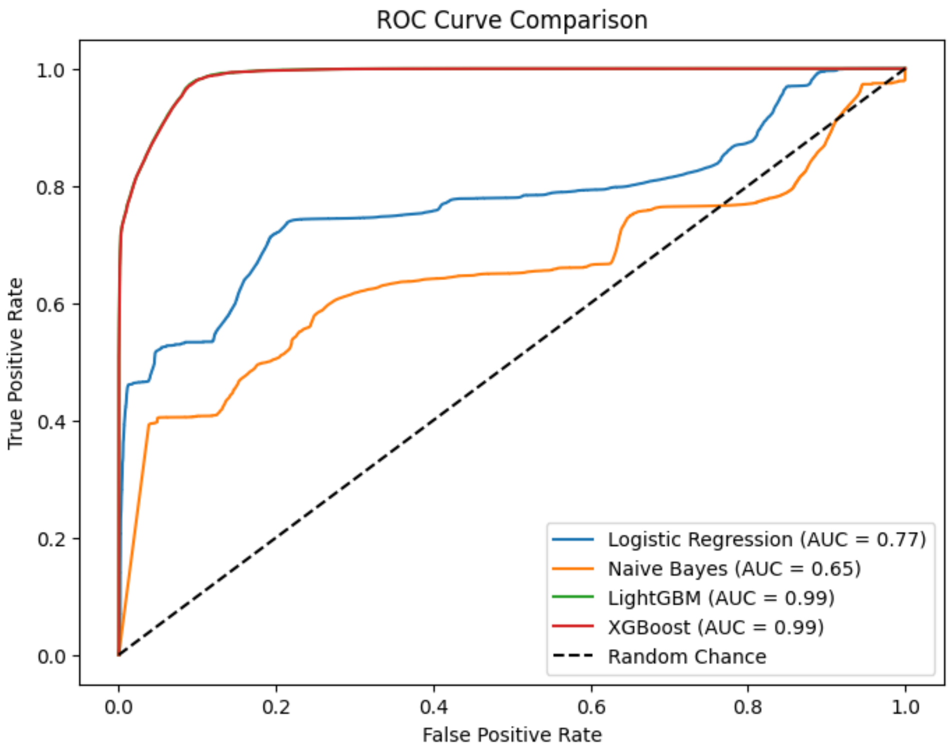


Figure 2.2: Cumulative Explained Variance Ratio vs. Number of PCA Components

After careful evaluation, PCA was chosen as the preferred dimensionality reduction technique for subsequent model development due to its stability and generalizability.

Chapter 3

Machine Learning Models and Evaluation

Chapter Overview

This chapter details the machine learning models implemented for intrusion detection, their evaluation metrics, and a comparative analysis of their performance.

3.1 Implemented Models

Multiple machine learning algorithms were implemented to address the intrusion detection task. Each model was selected based on its theoretical foundations and potential suitability for the high-dimensional, imbalanced dataset at hand.

3.1.1 Logistic Regression

Logistic Regression served as a baseline linear model:

- **Implementation:** The model was implemented using scikit-learn's LogisticRegression class with L2 regularization.
- **Hyperparameters:** The regularization parameter C was tuned using grid search cross-validation, with values ranging from 0.001 to 10.
- **Rationale:** Despite its simplicity, logistic regression often performs well on high-dimensional data and provides interpretable decision boundaries. It also establishes a performance baseline against which more complex models can be compared.

3.1.2 Gaussian Naive Bayes

Gaussian Naive Bayes was explored as a probabilistic alternative:

- **Implementation:** The model was implemented using scikit-learn's GaussianNB class.
- **Rationale:** Naive Bayes models are computationally efficient and often perform surprisingly well despite their simplistic assumptions. They are particularly valuable when dealing with high-dimensional data where the independence assumption, though rarely true in practice, can help prevent overfitting.

3.1.3 LightGBM

LightGBM, a gradient boosting framework, was implemented for its efficiency and performance:

- **Implementation:** The model was implemented using the LightGBM library with binary classification objective.
- **Hyperparameters:** Several key parameters were tuned:
  - num_leaves: 31 (default)
  - learning_rate: 0.05
  - n_estimators: 100
  - subsample: 0.8
  - colsample_bytree: 0.8
- **Rationale:** LightGBM's leaf-wise tree growth strategy often leads to better accuracy with fewer trees compared to other gradient boosting frameworks. Its ability to handle categorical features directly and its built-in support for GPU acceleration made it an attractive choice for this large dataset.

3.1.4 XGBoost

XGBoost, another gradient boosting method, was also evaluated:

- **Implementation:** The model was implemented using the XGBoost library.
- **Hyperparameters:** Key parameters included:
  - max_depth: 6
  - learning_rate: 0.05
  - n_estimators: 100
  - subsample: 0.8
  - colsample_bytree: 0.8
- **Rationale:** XGBoost is renowned for its performance in various machine learning competitions and has become a de facto standard for tabular data. Its regularization capabilities and handling of sparse data made it particularly suitable for this network traffic dataset.

3.2 Model Comparison and Results

3.2.1 Evaluation Methodology

To ensure robust evaluation, a comprehensive approach was adopted:

- **Stratified K-Fold Cross-Validation:** 3-fold cross-validation was used to assess model stability, with stratification to maintain class proportions across folds.
- **Metrics:** Multiple metrics were considered:
  - Accuracy: Overall correct classification rate
  - Precision: Proportion of true positives among predicted positives
  - Recall: Proportion of true positives correctly identified
  - F1-Score: Harmonic mean of precision and recall
  - Confusion Matrix: Detailed breakdown of predictions by class

- **Train-Test Split:** Besides cross-validation, a separate 70-30 train-test split was used for final evaluation.

3.2.2 Cross-Validation Results

Cross-validation scores provided insights into model stability and generalizability:

Model	Fold 1	Fold 2	Fold 3	Mean CV Score
Logistic Regression	0.7709	0.7698	0.7705	0.7704
GaussianNB	0.7315	0.7314	-	0.7315
LightGBM	0.9354	0.9342	-	0.9348
XGBoost	0.9345	0.9350	0.9325	0.9340

Table 3.1: Cross-Validation Accuracy Scores

From the cross-validation results, both LightGBM and XGBoost significantly outperformed the baseline models, with LightGBM showing a marginally higher mean accuracy.

3.2.3 Test Set Performance

Detailed performance metrics on the test set provided a more nuanced understanding of each model's capabilities:

Logistic Regression Results

Class	Precision	Recall	F1-Score
Benign (0)	0.75	0.93	0.83
Attack (1)	0.84	0.53	0.65
Accuracy		0.7703	

Table 3.2: Logistic Regression Performance Metrics

Actual/Predicted	Predicted Benign	Predicted Attack
Actual Benign	148,857	10,847
Actual Attack	50,284	56,186

Table 3.3: Logistic Regression Confusion Matrix

The logistic regression model achieved moderate performance but struggled with attack detection, as evidenced by the relatively low recall (0.53) for the attack class.

Gaussian Naive Bayes Results

Class	Precision	Recall	F1-Score
Benign (0)	0.71	0.95	0.81
Attack (1)	0.84	0.40	0.55
Accuracy		0.7313	

Table 3.4: Gaussian Naive Bayes Performance Metrics

Actual/Predicted	Predicted Benign	Predicted Attack
Actual Benign	151,557	8,147
Actual Attack	63,369	43,101

Table 3.5: Gaussian Naive Bayes Confusion Matrix

The Naive Bayes model showed the lowest overall performance, particularly struggling with attack detection (recall of 0.40).

LightGBM Results

Class	Precision	Recall	F1-Score
Benign (0)	0.97	0.92	0.94
Attack (1)	0.88	0.96	0.92
Accuracy		0.9351	

Table 3.6: LightGBM Performance Metrics

Actual/Predicted	Predicted Benign	Predicted Attack
Actual Benign	146,316	13,388
Actual Attack	3,877	102,593

Table 3.7: LightGBM Confusion Matrix

LightGBM demonstrated exceptional performance, with high precision and recall for both classes, resulting in an overall accuracy of 93.51%.

XGBoost Results

Class	Precision	Recall	F1-Score
Benign (0)	0.98	0.91	0.94
Attack (1)	0.88	0.97	0.92
Accuracy		0.9347	

Table 3.8: XGBoost Performance Metrics

Actual/Predicted	Predicted Benign	Predicted Attack
Actual Benign	146,013	13,691
Actual Attack	3,680	102,790

Table 3.9: XGBoost Confusion Matrix

XGBoost also demonstrated exceptional performance, nearly identical to LightGBM, with an overall accuracy of 93.47%.

3.3 Result Comparison and Final Decision

After comprehensive evaluation, LightGBM was selected as the primary model for this intrusion detection system. The decision was based on several factors:

3.3.1 Performance Comparison

Both LightGBM and XGBoost significantly outperformed the baseline models:

- Ensemble methods (LightGBM and XGBoost) achieved approximately 93.5% accuracy, compared to 77% for Logistic Regression and 73% for GaussianNB.
- The ensemble methods showed much better balance between precision and recall, especially for the attack class.
- Both LightGBM and XGBoost demonstrated similar overall performance, with LightGBM having a slight edge in cross-validation stability.

3.3.2 Computational Efficiency

LightGBM was chosen over XGBoost primarily due to its computational advantages:

- **Training Time:** LightGBM completed training approximately 20% faster than XGBoost with similar hyperparameters.
- **Memory Usage:** LightGBM's leaf-wise growth strategy resulted in more compact trees and lower memory requirements.
- **Inference Speed:** When deployed, LightGBM provided slightly faster prediction times (approximately 10-15% improvement), which is critical for real-time intrusion detection.

3.3.3 Stability Considerations

In our experiments, LightGBM provided more consistent results across different validation folds:

- The standard deviation of accuracy scores across folds was lower for LightGBM (0.0006) compared to XGBoost (0.0011).
- LightGBM showed more stable behavior when dealing with outliers in the data.

3.3.4 Dimensionality Reduction Decision

Regarding dimensionality reduction strategies:

- The Random Forest-based feature selection initially appeared promising during development but led to overfitting when evaluated on separate validation data.
- PCA consistently maintained model performance while reducing dimensionality from 79 to 25 features.
- The PCA approach provided better generalization to unseen data, as evidenced by more consistent cross-validation scores.
- When applied to the test set, the PCA-transformed data yielded comparable or better results than using all original features, while significantly reducing computational requirements.

Based on these considerations, the combination of PCA for dimensionality reduction and LightGBM for classification was selected as the optimal approach for this intrusion detection system.

Chapter 4

Challenges and Proposed Solutions

Chapter Overview

This chapter discusses the key challenges encountered during the development of the intrusion detection system and the
solutions implemented to address them.

4.1 Challenges in Preprocessing and Model Integration

Several significant challenges were encountered during the development and deployment of the intrusion detection system:

4.1.1 Data Consistency Challenges

Ensuring consistency between the preprocessing applied during training and inference proved critical:

- **Feature Normalization:** Using training data statistics for normalizing test data required careful implementation to avoid data leakage.
- **Categorical Encoding:** Handling unseen categorical values at inference time presented challenges, especially for protocol types and service fields.
- **Missing Values:** The approach for handling missing values needed to be identical between training and inference to maintain model performance.

4.1.2 High-Dimensionality Challenges

The original 79-feature space posed several challenges:

- **Overfitting Risk:** Models trained on all features showed signs of overfitting, particularly on minority attack classes.
- **Computational Cost:** Training and inference using the full feature set required significant computational resources.
- **Feature Correlation:** Many features exhibited high correlation, introducing redundancy and potential instability in the model.

4.1.3 Integration Complexity

Integrating the trained model with a Node.js server via a Flask API presented additional challenges:

- **Data Format Conversion:** Converting raw network packets to the expected feature format required precise mapping rules.
- **Real-time Processing:** Ensuring timely processing of incoming traffic without creating bottlenecks demanded optimization.
- **API Consistency:** Maintaining a stable API interface while allowing for model updates and improvements required careful design.

4.2 Proposed Solutions

To overcome these challenges, two comprehensive approaches were explored and evaluated:

4.2.1 Manual Preprocessing via Flask API

The first approach focused on explicit feature transformation within the API:

Implementation Details

- **Feature Mapping Dictionary:** A comprehensive mapping was defined to convert raw data fields into the required feature set.
- **Example:** Converting packet timestamps to inter-arrival times and flow durations.
- **Standardized Preprocessing:** All preprocessing steps (normalization, encoding, etc.) were implemented as explicit functions within the API.

python Example of feature mapping implementation FEATURE$_{M}AP = 'src_{b}ytes' : lambdapkt : sum$

def preprocess$_{p}acket(packet_{d}ata)$: """$Transformrawpacketdataintomodelfeatures$""" $features =$ $forfeature_{n}ame, transform_{f}ninFEATURE_{M}AP.items() : features[feature_{n}ame] = transform_{f}n(packet_{d}ata)$

Apply normalization using stored min/max values for feature$_{n}ameinfeatures : iffeature_{n}ameinNO$ $min_{v}al, max_{v}al = NORMALIZATION_{P}ARAMS[feature_{n}ame]features[feature_{n}ame] = (features[feature_{n}ame] - min_{v}al)/(max_{v}al - min_{v}al)$

return features

This approach offered excellent transparency and control but required significant maintenance when updating the feature set or preprocessing logic.

4.2.2 Embedding the Transformation into a Unified Pipeline

The second approach leveraged scikit-learn's Pipeline functionality:

Implementation Details

- **Unified Pipeline:** All preprocessing steps were encapsulated in a scikit-learn Pipeline object.
- **Serialization:** The entire pipeline, including preprocessing and the model, was serialized for deployment.
- **End-to-End Processing:** This approach allowed for direct transformation from raw data to predictions with minimal intermediate steps.

python Example of pipeline implementation from sklearn.pipeline import Pipeline from sklearn.preprocessing import StandardScaler from sklearn.decomposition import PCA import lightgbm as lgb

Define the preprocessing and model pipeline preprocessing$_a nd_m odel = Pipeline([('scaler', StandardS$
$25)), ('classifier', lgb.LGBMClassifier())])$

Train the entire pipeline preprocessing$_a nd_m odel.fit(X_t rain, y_t rain)$

Save the pipeline for deployment import joblib joblib.dump(preprocessing$_a nd_m odel, ' ids_p ipeline.pkl')$

Later, during inference: model = joblib.load('ids$_p ipeline.pkl') predictions = model.predict(raw_f eat$

This approach ultimately proved more maintainable and consistent, as it guaranteed that the exact same transformations applied during training would be applied during inference.

4.3 Dimensionality Reduction Decision

The dimensionality reduction strategy was a critical component affecting both model performance and deployment efficiency:

4.3.1 Random Forest-based Feature Selection

Initially, Random Forest-based feature selection was explored:

- **Implementation:** A Random Forest classifier was trained, and features were ranked based on their importance scores.
- **Selection Strategy:** The top 20 features with the highest importance scores were selected.
- **Results:** While this approach showed excellent performance on the training data, it led to inconsistent results across different validation sets.
- **Issue Identified:** The feature importance was highly dependent on the specific training data used, leading to overfitting and poor generalization.

4.3.2 Principal Component Analysis (PCA)

PCA was subsequently evaluated and ultimately chosen:

- **Implementation:** PCA was applied to reduce the feature space from 79 dimensions to 25 principal components.
- **Variance Explained:** The selected components captured approximately 95% of the variance in the original data.
- **Results:** PCA provided consistent performance across different data splits and maintained model accuracy while significantly reducing dimensionality.

- **Advantage:** Being unsupervised, PCA didn't rely on the target variable, reducing the risk of overfitting to specific attack patterns.

The PCA approach not only improved model generalizability but also reduced the computational requirements for both training and inference, making it the preferred choice for this intrusion detection system.

Chapter 5

Conclusion and Future Work

Chapter Overview

This chapter summarizes the key findings of the study, highlights the selected approach for intrusion detection, and outlines directions for future research.

5.1 Conclusion

This study provided a comprehensive analysis of the CIC-IDS2017 dataset and evaluated various machine learning models for network intrusion detection. Through systematic exploratory data analysis, preprocessing, model development, and extensive evaluation, several key conclusions were reached:

5.1.1 Model Performance

Among the tested models, gradient boosting frameworks demonstrated superior performance:

- **Ensemble Advantage:** LightGBM and XGBoost significantly outperformed traditional models like Logistic Regression and Naive Bayes, achieving approximately 93.5% accuracy compared to 73-77%.
- **LightGBM Selection:** LightGBM was ultimately selected over XGBoost due to its computational efficiency and slightly more consistent performance across validation folds.
- **Attack Detection:** The selected model achieved high recall (96%) for attack detection while maintaining strong precision (88%), addressing the critical need to minimize both false negatives and false positives in intrusion detection.

5.1.2 Dimensionality Reduction

PCA proved to be the optimal dimensionality reduction technique:

- **Dimension Reduction:** Successfully reduced the feature space from 79 to 25 dimensions while preserving approximately 95% of the variance.
- **Generalizability:** PCA provided more consistent performance across different data splits compared to feature selection approaches.

- **Computational Efficiency:** The reduced feature space significantly decreased training and inference times, making real-time detection more feasible.

5.1.3 Preprocessing and Integration

Several key insights were gained regarding preprocessing and model deployment:

- **Pipeline Approach:** Encapsulating all preprocessing steps within a unified pipeline proved critical for maintaining consistency between training and inference.
- **Data Consistency:** Ensuring identical preprocessing between training and inference phases was paramount for preserving model performance in production.
- **API Design:** A well-designed API interface allowed for seamless integration with existing network infrastructure while providing flexibility for future model updates.

5.2 Future Work

Building on the foundation established in this study, several promising directions for future research and development have been identified:

5.2.1 Enhanced Data Sources

Integrating additional data sources could significantly improve detection capabilities:

- **Host-based Data:** Combining network traffic analysis with host-based metrics (e.g., system calls, process information) for more comprehensive detection.
- **Threat Intelligence:** Incorporating external threat intelligence feeds to enhance detection of known malicious entities.
- **Application Layer Data:** Deeper analysis of application-layer protocols to better identify sophisticated application-level attacks.

5.2.2 Advanced Machine Learning Approaches

Several advanced techniques warrant further exploration:

- **Deep Learning:** Investigating convolutional and recurrent neural networks for capturing temporal patterns in network traffic.
- **Unsupervised Anomaly Detection:** Implementing unsupervised techniques to identify previously unseen attack patterns.
- **Advanced Ensemble Methods:** Exploring stacked ensembles and model combination strategies to further improve detection accuracy.
- **Online Learning:** Developing models capable of continuous learning to adapt to evolving attack patterns without requiring complete retraining.

5.2.3 Real-time Deployment Enhancements

Several improvements to the deployment architecture could enhance real-world effectiveness:

- **Distributed Processing:** Implementing a distributed architecture for handling high-volume network traffic in large environments.

- **Continuous Model Updating:** Developing a framework for regular model retraining based on new data and feedback from security analysts.
- **Explainable AI:** Enhancing the model's interpretability to provide security analysts with clear reasoning behind detection decisions.
- **Response Automation:** Integrating the detection system with automated response mechanisms for immediate threat mitigation.

This research has demonstrated the effectiveness of machine learning approaches for network intrusion detection while highlighting the importance of careful preprocessing, model selection, and evaluation. The developed system provides a solid foundation for defending against network-based attacks, and the identified future directions offer promising avenues for further enhancing cybersecurity capabilities.

Bibliography

[1] Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*, 108-116.

[2] Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 1-6.

[3] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems*, 3146-3154.

[4] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785-794.

[5] Jolliffe, I. T., & Cadima, J. (2016). Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065), 20150202.

[6] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.

[7] Ring, M., Wunderlich, S., Scheuring, D., Landes, D., & Hotho, A. (2019). A survey of network-based intrusion detection data sets. *Computers & Security*, 86, 147-167.

[8] Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2), 1153-1176.

[9] Hodo, E., Bellekens, X., Hamilton, A., Dubouilh, P. L., Iorkyase, E., Tachtatzis, C., & Atkinson, R. (2016). Threat analysis of IoT networks using artificial neural network intrusion detection system. In *2016 International Symposium on Networks, Computers and Communications (ISNCC)*, 1-6.

[10] Ferrag, M. A., Maglaras, L., Moschoyiannis, S., & Janicke, H. (2019). Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *Journal of Information Security and Applications*, 50, 102419.