A Project Report On

"Network Intrusion Detection System"

Submitted in partial fulfilment of the
requirement for the award of degree of

B.Tech in Computer Engineering
of
Fr. Conceicao Rodrigues College of Engineering
Mumbai

Compiled by:
Bhushan Anand Ladgaonkar

Under the guidance of:

Mr. Arvind Kumar
Manager (Programming)
Database Group, WOB, ONGC, Mumbai

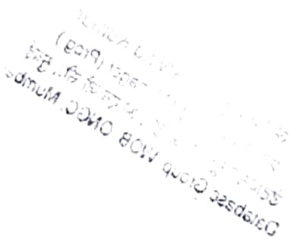
May-June, 2024

Declaration

I hereby declare that the work presented in this project report entitled **"Network Intrusion Detection System"** in partial fulfilment of requirement for the award of degree of **B.Tech in Computer Engineering**, submitted in the department of Computer Engineering , Fr. Conceicao Rodrigues College of Engineering , Mumbai is an authentic record of my work carried out during the Summer Training from 19/05/2025 to 19/06/2025 , under the guidance of **Mr. Arvind Kumar**, Database Group, WOB, ONGC, Mumbai.

Bhushan

Signature of the Student



Bhushan Anand Ladgaonkar

Certificate

This is to certify that **Mr. Bhushan Ladgaonkar**, a student of **B.Tech in Computer Engineering** of Fr. Conceicao Rodrigues College of Engineering, Mumbai has done his Summer Training at Database Group, WOB, ONGC, Mumbai. The project work entitled **"Network Intrusion Detection System"** embodies the original work done by **Mr. Bhushan Ladgaonkar** during his summer training period from 19/05/2025 to 19/06/2025

अरविन्द कुमार / Arvind Kumar
प्रबंधक (प्रोग्रामिंग) / Manager (Prog.)
तेलदोल समूह, अ. द्रो., ओ.एन.जी.सी., मुंबई
Database Group, WOB, ONGC, Mumbai

Signature of Project Guide/Mentor

Mr. Arvind Kumar

Ch. Manager (Programming)

Acknowledgement

The summer training at ONGC is a golden opportunity for learning and self-development. I consider myself very fortunate and honoured to be able to be a part of it and have such experienced and expertprofessionalsto lead me through the completion of this project. It gives me immense pleasure and a sense of satisfaction to have an opportunity to acknowledge and to express gratitude to those who were associated with me during my summer training at Database Group, WOB, ONGC, Mumbai**.**

I express my sincere thanks and gratitude to ONGC authorities for allowing me to undergo the training in this prestigious organization. I would like to thank Database Group, WOB, ONGC, Mumbai for providing me the technical guidance, opportunity and infrastructure to work.

Finally, I would thank my parents for imparting me moral support and motivation during this project.

Table of Contents

1. Abstract

In today's digital age, the increasing frequency and sophistication of cyberattacks necessitate advanced security solutions. Traditional Intrusion Detection Systems (IDS), which rely on static signatures, often fail to detect novel threats, creating significant security vulnerabilities. This project addresses this gap by developing an intelligent Network Intrusion Detection System (NIDS) using machine learning. The system is trained on the comprehensive UNSW-NB15 dataset to accurately classify network traffic as either "Benign" (normal) or "Attack". The project followed a complete data science workflow, including data preprocessing, feature engineering, and a comparative analysis of different models and class imbalance handling techniques. The raw dataset was cleaned by dropping irrelevant columns, removing over 490,000 duplicate records, and handling data types. Feature selection was performed using a RandomForest model to identify the most impactful features, and log transformation was applied to handle skewed data distributions. Four machine learning models—Logistic Regression, RandomForest, XGBoost, and LightGBM—were systematically trained and evaluated under two distinct scenarios: with a balanced training set created using SMOTE (Synthetic Minority Over-sampling Technique), and without SMOTE using class weight adjustments. The comprehensive comparison revealed that while SMOTE significantly boosted the performance of the linear model, the tree-based models achieved superior results without it. The LightGBM (No SMOTE) model proved most effective at minimizing missed attacks (lowest False Negatives), while the RandomForest (No SMOTE) model achieved the best F1-Score and lowest false alarm rate. Ultimately, the RandomForest (With SMOTE) model was identified as the best overall solution, offering an excellent balance between high detection rates and reliability.

2. Problem Statement

As global interconnectivity expands, the volume and complexity of network traffic are increasing at an unprecedented rate. While this digital transformation enables innovation, it also creates a larger and more sophisticated attack surface for malicious actors. Traditional Network Intrusion Detection Systems (IDS), which have long been a cornerstone of cybersecurity, primarily operate on a signature-based detection model. These systems rely on a pre-defined database of known attack patterns and signatures to identify threats.

The fundamental problem with this approach is its inherent inability to detect novel or evolving cyber threats. Modern adversaries frequently employ **zero-day exploits** (attacks that target previously unknown vulnerabilities) and **polymorphic attacks** (malware that constantly changes its code to evade signature detection). A signature-based IDS is effectively blind to these new forms of intrusion until its database is manually updated, a process that can take hours or days, leaving a critical window of vulnerability.

This project directly addresses this security gap by tackling the following challenges:

1. **Inadequacy of Static Rules:** The core challenge is to move beyond static, rule-based detection and develop an intelligent, adaptive system that can learn from data to identify malicious patterns.
2. **Handling a High-Dimensional and Complex Dataset:** The chosen dataset, **UNSW-NB15**, is representative of modern network environments. It is high-dimensional, containing numerous features that describe network flow characteristics. Building a model that can effectively learn from this complexity without overfitting is a significant task.
3. **Addressing Severe Class Imbalance:** Like real-world network traffic, the UNSW-NB15 dataset is severely imbalanced, with a very small proportion of attack records compared to a vast number of normal (benign) traffic records. This poses a major challenge for machine learning models, as they can easily become biased towards the majority class and achieve high accuracy simply by ignoring the rare but critical attack class.

Therefore, the problem is to design and implement a machine learning pipeline capable of accurately classifying network traffic as either **Benign** or **Attack** on the complex and imbalanced UNSW-NB15 dataset, thereby providing a more robust and adaptive solution for modern network security.

3. Scope and Objectives of Project

Scope:

- The scope of this project is to design, implement, and evaluate a machine learning-based Network Intrusion Detection System. The development is conducted exclusively using the **UNSW-NB15 dataset**, a comprehensive collection of real and simulated network traffic that includes modern attack scenarios.
- The primary function of the developed system is **binary classification**, where the goal is to accurately categorize individual network traffic records into one of two classes: **'Benign'** (normal) or **'Attack'**.
- This project encompasses the full data science lifecycle, from data acquisition and cleaning to model training and performance evaluation. However, the scope is confined to the development and testing of the machine learning model itself. It does not include real-time network traffic monitoring, integration with existing security infrastructure like firewalls, or the development of a front-end graphical user interface (GUI).
- The final deliverable is a fully functional and documented Jupyter Notebook that contains the complete data processing pipeline and the trained models, ready for analysis and potential future deployment.

Objectives:

- To clean the **UNSW-NB15** dataset by handling missing data, removing duplicate rows, and converting data types.
- To perform feature engineering by analyzing feature correlations and using a RandomForest model to rank and select the most important features.
- To implement and compare two distinct strategies for handling class imbalance: SMOTE oversampling and class weight adjustments.
- To train and evaluate four machine learning models: **Logistic Regression, RandomForest, XGBoost, and LightGBM.**
- To compare the models using metrics like **Accuracy, Precision, Recall, and F1-Score,** focusing on the trade-offs between minimizing false negatives and false positives.

4. Solution Design

The project was structured as a sequential machine learning pipeline, designed for both effectiveness and computational efficiency on standard hardware.

1.Data Loading & Checkpointing:
The four raw UNSW-NB15 CSV files were initially merged into a single DataFrame. The columns were correctly named by reading the NUSW-NB15_features.csv file. To optimize subsequent runs, this merged DataFrame was saved as a single checkpoint file (merged_nids_dataset.csv).

2. Data Cleaning & Feature Engineering (Pre-Split):
This phase focused on cleaning the data and reducing its complexity before any memory-intensive operations.

- **Column Dropping:** Irrelevant columns (attack_cat, srcip, dstip) and features with a high percentage of missing values (ct_flw_http_mthd, is_ftp_login) were dropped.
- **Duplicate Removal:** Over 490,000 duplicate rows were identified and removed to ensure data quality.
- **Data Type Conversion:** Columns like sport, dsport, and ct_ftp_cmd were converted from text to integer types.
- **Cardinality Reduction:** To prevent memory errors during encoding, the service column's cardinality was limited by grouping its least frequent categories into a single "Other_Service" category.
- **Log Transformation:** To handle skewed distributions, a log1p transformation was applied to key numerical features.

4. **Feature Selection:**
   To further optimize performance and reduce dimensionality, feature selection was performed. A RandomForest model was trained on a 20% stratified sample of the data. Based on the calculated feature importances' , the **top 66 features** were selected for the final models.

4. Final Preprocessing (Post-Split):

- **Train-Test Split:** The cleaned and feature-selected dataset was split into training (70%) and testing (30%) sets using stratified sampling to maintain the original class distribution.
- **One-Hot Encoding:** Categorical features (`proto`, `state`, `service`) in the training and testing sets were converted into a numerical format using `OneHotEncoder`, which produced a memory-efficient sparse matrix.
- **Standard Scaling:** All numerical features were standardized using `StandardScaler`, which was fit on the training data and applied to both sets.

5. Model Training and Evaluation:

Four models (**Logistic Regression, RandomForest, XGBoost, LightGBM**) were trained and evaluated under two scenarios:

- **Scenario 1 (Without SMOTE):** Models were trained on the processed, imbalanced training data. Imbalance was managed using the `scale_pos_weight` parameter where applicable.
- **Scenario 2 (With SMOTE):** The processed training data was first balanced using the SMOTE algorithm. The models were then trained on this oversampled dataset.

  All models were evaluated on the same unseen test set to ensure a fair and direct comparison of their performance.

5. Implementation Technologies and Platforms

- **Language & Environment:**
  This project was implemented using **Python 3.11** within the **Jupyter Lab** environment. The entire workflow was managed using the **Anaconda** distribution, which provided a robust and isolated environment for managing the project's dependencies.
- **Core Libraries:**
  - **Pandas & NumPy:** These libraries were fundamental for all data handling tasks, including efficient loading from CSVs, data cleaning, transformation, and manipulation of the large-scale network traffic dataset.
  - **Scikit-learn:** This served as the primary machine learning framework. It was used for a wide range of tasks, including:
    - **Preprocessing:** StandardScaler for feature scaling, OneHotEncoder and ColumnTransformer for handling categorical data.
    - **Model Training:** LogisticRegression and RandomForestClassifier.
    - **Data Splitting:** train_test_split for creating stratified train and test sets.
    - **Evaluation:** A suite of metrics including classification_report, confusion_matrix, roc_auc_score, and others.
  - **Imbalanced-learn:** This specialized library was crucial for addressing the class imbalance using the **SMOTE (Synthetic Minority Over-sampling Technique)** algorithm.
  - **XGBoost & LightGBM:** These high-performance, gradient-boosting libraries were chosen for their state-of-the-art performance, speed, and memory efficiency, making them ideal for training on millions of records.
  - **Joblib:** Employed for serializing and saving the final trained models and preprocessing pipelines to disk, ensuring they can be reloaded for future use without retraining.

- **Visualization:**
    - **Matplotlib & Seaborn:** Used extensively to create all visualizations, including feature importance bar plots, confusion matrices, and comparative ROC curves.

6. Deployment and Testing of the Software

Testing:
The robustness and generalization capability of the developed models were rigorously tested. The cleaned and feature-selected dataset, containing 2,048,193 records, was split into a **70% training set (1,433,735 records)** and a **30% testing set (614,458 records)**.

The testing set was held out and remained completely unseen by the models during the entire training process (including fitting of scalers, encoders, and the SMOTE algorithm). All final performance metrics, such as Precision, Recall, and F1-Score, were calculated exclusively on this unseen test set. This methodology ensures a realistic and unbiased evaluation of how well each model would perform on new, real-world data.

Deployment:
While real-time deployment was outside the scope of this project, the entire pipeline was designed with future deployment in mind. The final, optimized pipeline can be saved into serialized files using libraries like joblib or pickle.

The key components to be saved for a deployment pipeline would include:

- **The Final Trained Model:** (e.g., the RandomForest (With SMOTE) model object).
- **The Preprocessor Object:** The fitted ColumnTransformer that contains the learned StandardScaler and OneHotEncoder.
- **The List of Selected Features:** The list of the 66 feature names that the model expects as input.

These saved files would allow the complete preprocessing and prediction workflow to be loaded into a separate Python script or an API, enabling real-time classification of new network traffic data without needing to retrain.

7. Output Snapshots

Snapshot 1: Initial Data Shape:

```
--- Starting Data Loading ---
Loading data from existing checkpoint: .\merged_nids_dataset.csv
Data loaded successfully from checkpoint.

Final DataFrame shape after loading/merging: (2540047, 49)

--- Data Loading Complete ---
```

Figure 1: Initial shape of the merged UNSW-NB15 dataset.

Snapshot 2: Data Cleaning Summary:

```
--- Starting Data Cleaning ---

Limited 'service' column to top 10 categories + 'Other_Service'.
New unique service categories: 11

Dropped 5 highly correlated features.

Applied log transformation to key numerical features.

Shape after all cleaning: (2048193, 39)

--- Data Cleaning Complete ---
```

Figure 2: Summary of the data cleaning process, including duplicate removal and feature reduction.

Snapshot 3: Feature Importance Plot:

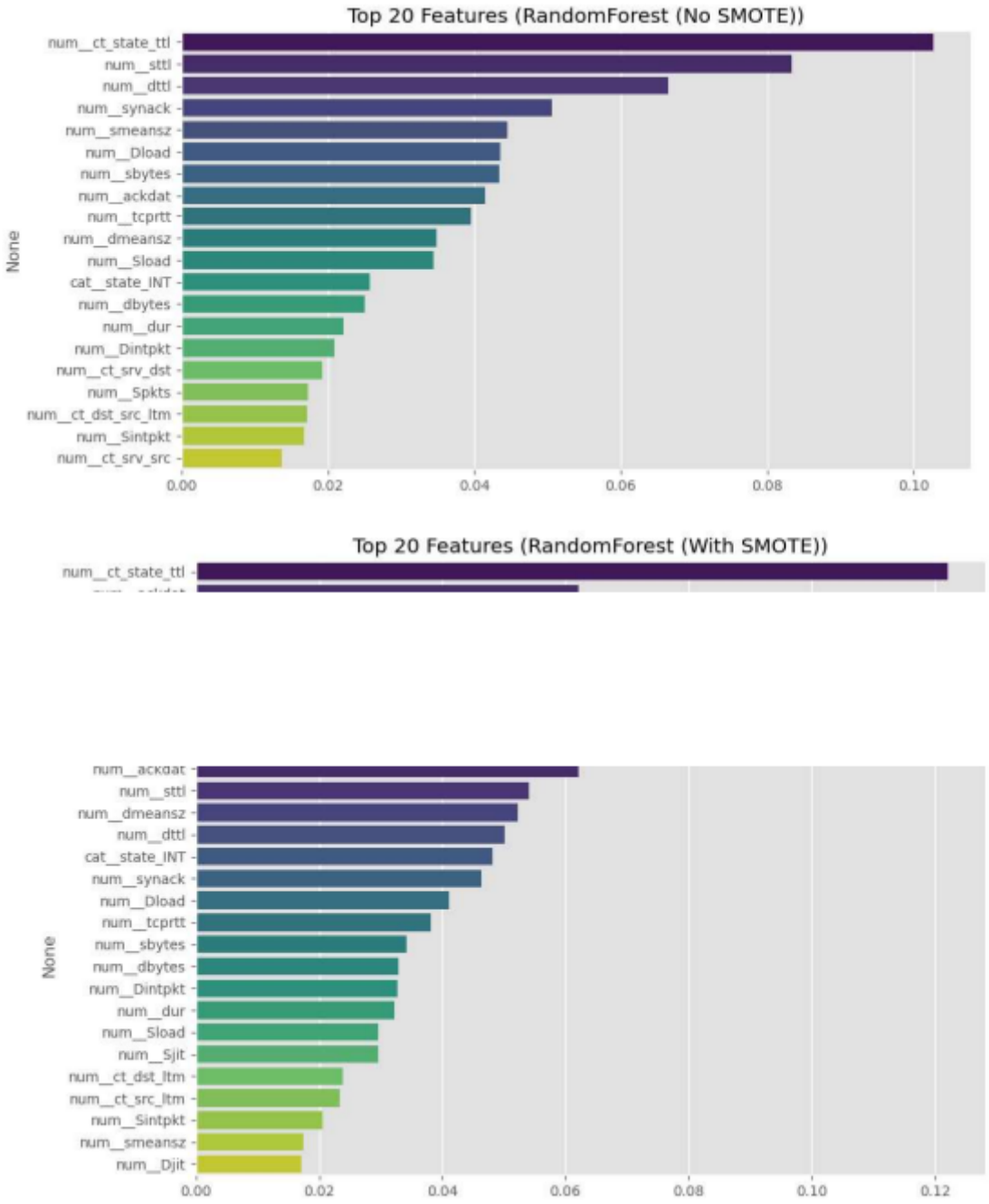


Figure 3: Top 20 most important features as determined by the RandomForest model.

Snapshot 4: Model Performance Comparison Table:

```
--- Analysis of Model Performance: Category Winners ---


--- Conclusion: Best Performing Models by Category ---

🚀 Fastest Training Time:              LightGBM (No SMOTE) (16.36 seconds)
------------------------------------------------------------
🎯 Highest Accuracy:                   RandomForest (No SMOTE) (0.9937)
📈 Highest Precision (Macro Avg):      RandomForest (No SMOTE) (0.9683)
📡 Highest Recall (Macro Avg):         XGBoost (No SMOTE) (0.9923)
⚖️ Best F1-Score (Balance):            RandomForest (No SMOTE) (0.9617)
📊 Best ROC AUC Score:                 XGBoost (No SMOTE) (0.9993)
------------------------------------------------------------
🚨 Lowest False Positives (FP): RandomForest (No SMOTE) (Count: 1538)
🛡️ Lowest False Negatives (FN): LightGBM (No SMOTE) (Count: 26)
------------------------------------------------------------

--- Overall Recommendation for NIDS ---

⏳ The recommended 'Best Overall' model from this analysis, balancing detection and
reliability, is **RandomForest (With SMOTE)**.

--- Final Analysis Complete ---
```

Figure 4: Comparative performance metrics for all four models, with and without SMOTE.

Snapshot 5: Confusion Matrices:

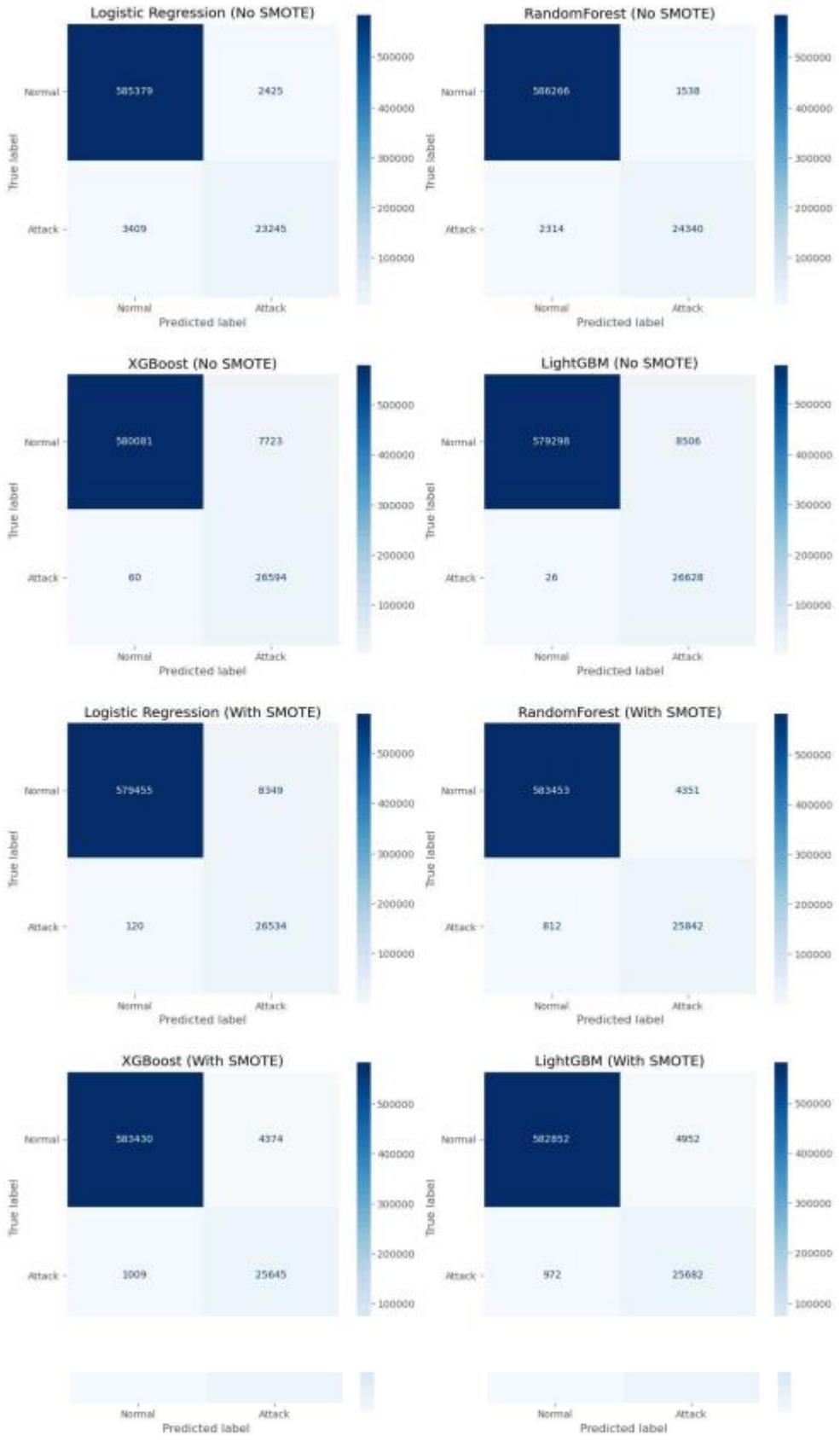


Figure 5: Confusion matrices for all models, visualizing True/False Positives and Negatives.

Snapshot 6: ROC Curves:

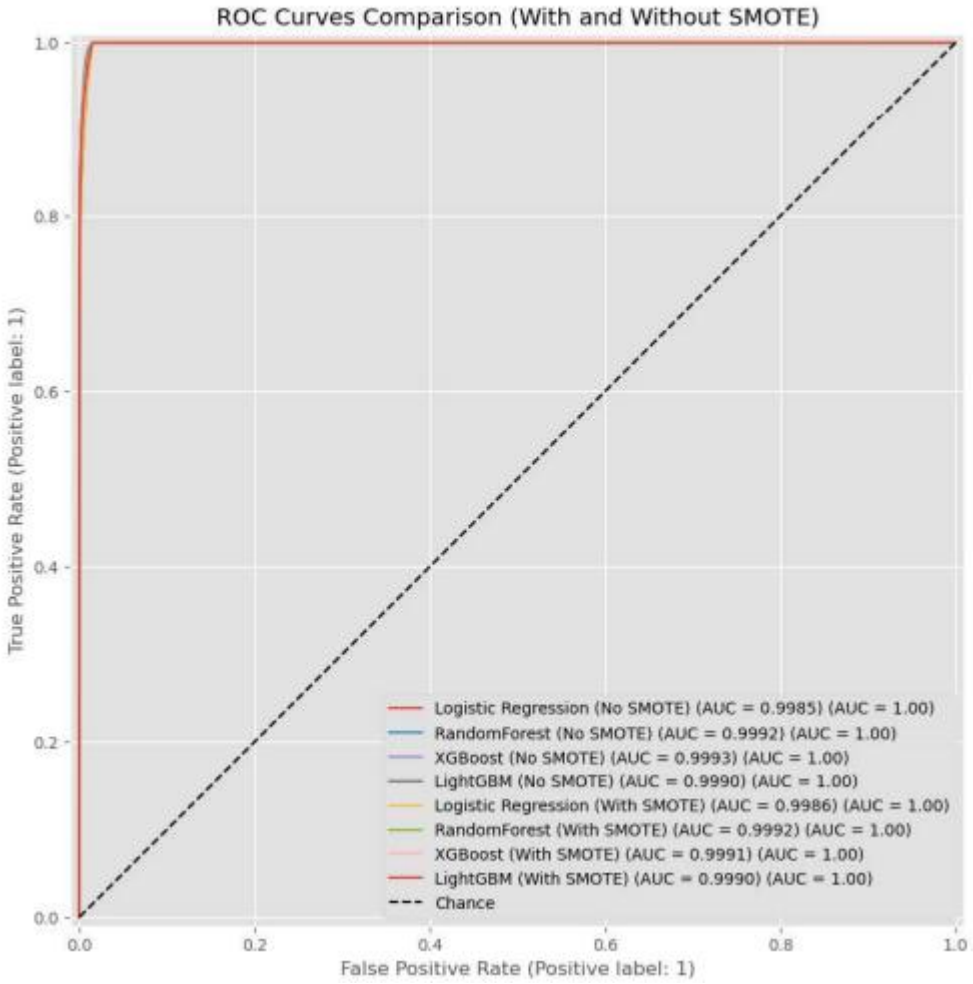


Figure 6: ROC curves comparing the class separation ability of all models.

8. Future Scope

This project has successfully established a robust baseline for a binary classification NIDS. However, its capabilities can be significantly expanded and enhanced in several key areas. The following outlines potential directions for future work that would build upon the foundation laid by this project.

1. Multiclass Attack Classification:
The most immediate and impactful next step is to extend the system from a binary classifier to a **multiclass classifier**.

- **Objective:** Instead of just detecting an "Attack," the system would identify the specific *type* of attack (e.g., DoS, Exploits, Fuzzers, Reconnaissance, etc.).
- **Implementation:** This would involve using the `attack_cat` column from the original dataset as the target variable. The `NaN` values in this column, which correspond to benign traffic, would be filled with a "Normal" label, creating a target with multiple distinct classes. The same high-performing models used in this project (RandomForest, XGBoost, LightGBM) are inherently capable of multiclass classification and could be retrained on this new target variable. This would provide security analysts with much more specific and actionable intelligence about the nature of a detected threat.

2. Real-Time Detection and Integration:
To transition from an offline model to a practical security tool, the system could be integrated into a live network environment.

- **Objective:** To classify network traffic in real-time or near-real-time.
- **Implementation:** The saved preprocessing pipeline and trained model could be deployed as a service. This service would integrate with network monitoring tools like **Scapy**, **Zeek (formerly Bro)**, or directly with packet capture libraries like **libpcap**. A script would be developed to capture network packets, extract the required 66 features, preprocess them using the saved scaler and encoder, and feed them to the model for a prediction.

3. Deep Learning Models for Sequential Data:
Network traffic has a temporal nature, where the sequence of packets can be as important as the features of a single flow.

- **Objective:** To capture time-based patterns and sequential dependencies in traffic that might indicate an attack.
- **Implementation:** Explore deep learning models like **Long Short-Term Memory (LSTM)** or **Gated Recurrent Unit (GRU)** networks. These models

are specifically designed to learn from sequences and could potentially identify sophisticated, low-and-slow attacks that traditional models might miss. This would require restructuring the data into time-series sequences of network flows.

4. Development of a User Interface (UI/UX) and Dashboard:
To make the system accessible to security analysts and network administrators, a graphical user interface is essential.

- **Objective:** To provide a visual dashboard for monitoring network traffic, viewing alerts, and analyzing model performance.
- **Implementation:** A web-based dashboard could be built using Python frameworks like **Streamlit** or **Flask/Dash**. This UI could feature:
    - Real-time graphs showing the volume of benign vs. attack traffic.
    - An alert log that details detected intrusions, including source/destination info and predicted attack type (if multiclass is implemented).
    - Interactive visualizations allowing analysts to filter traffic by protocol, port, or time period.
    - A section for model monitoring, tracking metrics like precision and recall over time to detect model drift.

5. Adversarial Machine Learning and Model Robustness:
As attackers become aware of ML-based detection, they will attempt to craft malicious traffic that evades the model.

- **Objective:** To make the model more resilient to adversarial attacks.
- **Implementation:** Research and implement techniques from the field of adversarial machine learning. This could involve generating adversarial examples to "stress-test" the model and then retraining it on this augmented data to improve its robustness against evasion techniques.

6. Automated Retraining and Model Updating:
Network patterns and attack methods are constantly evolving. A static model will eventually become outdated.

- **Objective:** To create a system that can periodically retrain itself on new data.
- **Implementation:** Develop an MLOps (Machine Learning Operations) pipeline that automates the process of collecting new labeled data, triggering a retraining of the model, evaluating the new model against the old one, and deploying the improved model if it meets performance criteria.

9. Conclusion

This project successfully demonstrated the development of a high-performance Network Intrusion Detection System using machine learning on the UNSW-NB15 dataset. Through a systematic process of data cleaning, feature engineering, and comparative model evaluation, a robust solution was achieved.

The exploration of different techniques to handle the severe class imbalance proved critical. While the models trained without SMOTE showed strong performance, particularly **LightGBM (No SMOTE)** which achieved the lowest number of False Negatives (26), applying **SMOTE** generally led to models with a better balance between detection and reliability.

The comparative analysis of four distinct algorithms revealed that tree-based ensemble models like RandomForest, XGBoost, and LightGBM significantly outperformed the baseline Logistic Regression model. The final evaluation identified the **RandomForest model trained with SMOTE** as the best overall performer, providing an excellent balance of high detection rates (Recall) and reliability (Precision), making it a strong candidate for practical deployment.

In conclusion, this project provided deep, hands-on experience in the end-to-end machine learning lifecycle—from handling large, raw data to cleaning, preprocessing, feature selection, training multiple advanced models, and performing a detailed comparative analysis. It underscored the importance of a structured data pipeline, the critical need to address class imbalance, and the trade-offs involved in selecting a final model based on specific security priorities.