Intrusion Detection using Machine Learning Methods

Sharath Kumar Karnati, 011852253
Department of Computer Science
Washington State University
Pullman, WA, USA
Email: s.karnati@wsu.edu

Abstract—**This project addresses the problem of detecting intrusions in network traffic using machine learning algorithms. The dataset used includes KDD Cup 1999 data, which contains various features such as protocol type, service, source/destination bytes, and login attempts, among others. The goal is to classify network traffic as either ”normal” or ”attack.” Several machine learning models, including Logistic Regression, K Nearest Neighbors (KNN), Gaussian Naive Bayes (GNB), Linear SVC, Decision Trees, Random Forest, and XG-Boost, were evaluated for their performance on this classification task. Data preprocessing techniques such as feature scaling, encodeing of categorical variables, and dimensionality reduction using Principal Component Analysis (PCA) were employed to optimize model performance. The models were evaluated based on accuracy, precision, recall, and F1-score. The results showed that while tree-based models like Random Forest and Decision Trees performed well in terms of accuracy, deep learning models like the neural network also provided promising results. The project also investigates the inter-pretability of model predictions using tools like feature importance analysis and confusion matrices. This research contributes to the field of intrusion detection systems by providing insights into the effectiveness of various machine learning techniques in real-world applications.**

Index Terms—**Intrusion Detection, Machine Learning, Anomaly Detection, Naive Bayes, SVM, Decision Trees, Random Forest, Explainable AI**

I. Introduction

A. Motivation from Real-World Applications

Intrusion detection systems (IDS) are critical for securing networks and protecting sensitive data from malicious attacks. In today’s increasingly interconnected world, cyber threats are constantly evolving, making the detection of network intrusions a vital task for both organisations and individuals. Machine learning (ML) offers a promising approach to enhance the effectiveness of IDS by automatically identifying abnormal patterns in network traffic and classifying them as either good or malicious. This project aims to leverage various ML algorithms to improve the detection of such attacks and reduce false positives in real world network environments.

B. Machine Learning Questions Investigated

The key questions investigated in this project include:

- Which machine learning models are most effective in detecting intrusions in network traffic?
- How do tree based models compare with other classifiers such as Logistic Regression or K-Nearest Neighbours in terms of accuracy, precision, recall, and F1score?
- How can data pre-processing techniques, such as feature scaling and PCA, improve the performance of ML models?
- What insights can be gained from feature importance and confusion matrices in interpreting the model’s predictions?

C. Personal Motivation and Goals

The motivation to choose this project rooted from both an academic interest in machine learning and a personal fascination with network security. Given the increasing frequency and sophistication of cyber attacks, it felt crucial to apply machine learning techniques to a realworld problem that directly impacts data security. My primary goal was to explore how machine learning could be used to improve the accuracy and efficiency of intrusion detection systems, particularly in distinguishing between normal and malicious network traffic.

D. Challenges and Approach

One of the main challenges in this project was dealing with the imbalance in the dataset, where the number of normal traffic instances significantly outnumbered attack instances. To address this, techniques such as oversampling and using different evaluation metrics like precision and recall were employed to better handle the imbalance. Additionally, tuning the hyper parameters of the models to avoid over-fitting and ensure optimal generalisation to unseen data presented another challenge. The approach involved a combination of data preprocessing, model evaluation using cross validation, and interpretability techniques to gain a deeper understanding of the model’s decision-making process.

E. Summary of Results

The results of this project showed that tree-based models, such as Random Forest and Decision Trees, performed well in terms of accuracy and precision in detecting intrusions. However, models like Logistic Regression and K Nearest Neighbors, while still effective, did not perform as robustly. The evaluation of inter-pretability tools revealed that Random Forest provided valuable insights into feature importance, which helped in understanding what factors influenced the model's predictions. Overall, the project demonstrated that machine learning is a powerful tool for intrusion detection, with tree based methods showing the best performance in this specific task.

II. Machine Learning Task

A. Task Description

The task of this project involves detecting intrusions in network traffic using machine learning algorithms. The input data consists of network traffic records, which include various features such as protocol type, service, source and destination bytes, number of failed login attempt, and many others. Each record represent a network connection, and the goal is to classify each connection as either "normal" or "attack." The dataset used in this project is the NSL-KDD Cup 1999 dataset, which is widely used for evaluating intrusion detection systems.

An example of input data is as follows:

```
tcp,http,80,0,2100,0,0,0,10,normal
```

This represents a connection using the TCP protocol, associated with the HTTP service on port 80, with a number of bytes exchanged, and marked as "normal."

The output of the machine learning approach is a classification label, where each record is classified as either "normal" (indicating no intrusion) or "attack" (indicating the presence of an intrusion). For example, a malicious attack such as a denialofservice (DoS) might be classified as "attack."

B. Machine Learning Questions Investigated

The following machine learning questions were addressed in this project:

- Which machine learning algorithms are most effective at classifing network traffic as normal or attack?
- How do tree-based models, such as Decision Trees and Random Forests, compare with other models like Logistic Regression and K Nearest Neighbors in detecting intrusions?
- How does data preprocessing, including feature scaling, encoding of categorical variables, and dimensionality reduction with PCA, impact model performance?
- How can model inter-pretability techniques, such as feature importance and confusion matrices, help explain the decisions made by the models?
- What is the impact of class imbalance on the performance of the models, and how can it be mitigated?

C. Key Challenges

The key challenges in solving this task include:

- **Class Imbalance**: The dataset is heavily imbalanced, with many more instances of normal traffic than attacks. This makes it difficult for models to correctly identify rare attack instances, leading to a high number of false negatives.
- **Data Pre-processing**: The raw dataset contains categorical variables that need to be encoded, and numerical features may require scaling to improve the performance of certain algorithms. Additionaly, dimensionality reduction methods like PCA are needed to reduce the complexity of the dataset.
- **Feature Selection**: The dataset contains many features, some of which may be redundant or irelevant. Selecting the most informative features is crucial to improving model accuracy and preventing over fitting.
- **Model Evaluation**: Evaluating models in the presence of class imbalance requires careful consideration of metrics like precision, recall, and F1-score, rather than just accuracy. Cross validation is used to ensure that the models generalize well to unseen data.
- **Interpretability**: Understanding how the model makes decisions is important for assessing its reliability and trustworthiness, especially when deployed in real world security systems.

III. Technical Approach

A. Algorithmic Approach

To solve the intrusion detection task, several machine learning algorithms were employed and evaluated. These algorithms include Logistic Regression, K Nearest Neighbors (KNN), Gaussian Naive Bayes (GNB), Linear SVC, Decision Trees, Random Forest, and XG-Boost. Each model was trained on the pre-processed dataset, and the performance was compared based on several evaluation metrics, including accuracy, precision, recall, and F1score. The overall approach can be summarized as follows:

- **Data Pre-processing**: The raw dataset is first cleaned by removing any missing values. Categorical variables, such as protocol type and service, are encoded into numerical values using one hot encoding. Continuous features, such as byte counts, are scaled using standardization or normalization. Dimensionality reduction via Principal Component Analysis (PCA) is applied to reduce the feature space, which helps mitigate over fitting and improve computation efficiency.
- **Model Training**: Various models are trained on the preprocessed data, with hyper parameters tuned using cross-validation. The models are fitted to the training data and evaluated on a separate test set to ensure they generalize well to unseen data.
- **Model Evaluation**: The performance of each model is evaluated using confusion matrices, precision, recall, F1 score, and accuracy. Since the dataset is imbalanced, emphasis is placed on precision and recall to ensure that attacks are detected with minimal false negatives.
- **Inter-pretability**: Feature importance analysis is performed using tree-based models like Decision Trees and Random Forests. This helps to identify the most influential features in the decision making process. Additionally, confusion matrices are used to visualize the model's performance and understand the types of errors it makes.

B. Addressing the Challenges

The following strategies were employed to address the challenges identified earlier:

- **Class Imbalance**: To address the class imbalance, techniques such as oversampling the minority class (attack) using SMOTE (Synthetic Minority Over-sampling Technique) and undersampling the majority class (normal) can be employed. Additionally, performance metrics like precision, recall, and F1-score were prioritized over accuracy to account for the imbalance.
- **Data Pre-processing**: As mentioned, categorical variables were encoded, and numerical variables were scaled using standard techniques such as Min n Max Scaling. PCA was used for dimensionality reduction to eliminate redundancy and reduce the computational complexity of the models.
- **Feature Selection**: During the model training phase, feature importance scores were calculated to identify and select the most relevant features, particularly when using decision tree based models.
- **Model Evaluation**: Stratified crossvalidation was used to ensure that the class distribution remained consistent across training and validation sets. This helped to mitigate the effects of class imbalance on model evaluation.
- **Interpretability**: For model interpretability, feature importance scores were obtained from tree based models (Random Forest and Decision Trees) to understand which features contributed the most to the classification decisions. The confusion matrix was also analyzed to identify common mis-classifications.

C. Algorithmic Pseudo-code

The following is a pseudo-code to summarize the steps involved in the algorithmic approach:

Algorithm 1 Intrusion Detection System - ML Approach

```
Network Traffic Dataset $D$ Trained Models and Evaluation Metrics
1: Preprocess dataset $D$
2:     Clean missing values
3:     Encode categorical variables
4:     Scale numerical features
5:     Apply PCA for dimensionality reduction
6: Split dataset into training and testing sets
7: for each model in {Logistic Regression, KNN, GNB, SVC, Decision Trees, Random Forest, XGBoost} do
8:     Train model on training set
9:     Evaluate model on testing set
10:    Store evaluation metrics (accuracy, precision, recall, F1-score)
11: end for
12: if class imbalance exists then
13:    Apply SMOTE or undersampling techniques
14: end if
15: for each tree-based model in {Decision Trees, Random Forest} do
16:    Calculate feature importance
17:    Generate confusion matrix
18: end for
19: Return evaluation metrics and model performance analysis
```

IV. Evaluation Methodology

A. Dataset and Its Source

The dataset used for this project is the NSL-KDD dataset, which is a modified version of the KDD Cup 1999 dataset. It is widely used for benchmarking intrusion detection systems. The dataset is publicly available on Kaggle and can be accessed via the following link: https://www.kaggle.com/datasets/hassan06/nslkdd/data.

The NSL KDD dataset contains network traffic data and is divided into training and testing sets. The data includes multiple features such as protocol type, service, source and destination bytes, flag, and count of various types of network traffic activities. The primary goal is to classify network traffic as either “normal” or “attack,” with several attack categories like DOS, Probe, R2L, and U2R.

One of the key challenges when using this dataset is its class imbalance problem. The ”normal” class significantly outnumbers the attack classes, making it harder for the models to learn and detect rare attacks. Addressing this issue requires the application of techniques like SMOTE (Synthetic Minority Oversampling Technique) or unde sampling to balance the dataset before training.

B. Evaluation Metrics

To evaluate the performance of the machine learning models, several standard metrics were employed:

- **Accuracy**: This metric measures the proportion of correct predictions (both normal and attack classes) to the total number of predictions. While commonly used, accuracy might not fully reflect the performance when dealing with imbalanced datasets, as it could be biased toward the majority class (normal).

- **Precision**: Precision calculates the percentage of true positives among all instances predicted as positives (i.e., correctly identified attacks). In the context of intrusion detection, high precision ensures that most of the detected attacks are actual attacks and not false alarms.
- **Recall**: Re-call measures the percentage of true positives identified out of all actual positive instances (real attacks). High re-call ensures that the model successfully detects a large number of attacks, minimizing the number of undetected attacks (false negatives).
- **F1-score**: The F1 score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance. It is particularly useful in situations where both false positives and false negatives have significant consequences, such as in intrusion detection, where missing an attack (false-negative) or raising too many false alarms (false-positives) is problematic.
- **Confusion Matrix**: A confusion matrix provides a comprehensive overview of the model's predictions by comparing the true and predicted classes. It helps in visualizing false-positives, false-negatives, true-positives, and true-negatives, offering insights into where the model is making errors.
- **Feature Importance**: For tree based models such as Decision Trees and Random Forest, feature importance scores are computed to identify the most influential features in predicting the network traffic behavior. This analysis helps in understanding which network traffic features contribute most to detecting intrusions.

These metrics were chosen based on their relevance to real-world applications of intrusion detection systems. Accuracy alone is not enough, especially when class imbalance is present, as the model might simply classify everything as "normal" to achieve high accuracy. Precision, recall, and F1 score provide a more balanced understanding of the model's ability to detect attacks while minimizing false alarms and undetected attacks. The confusion matrix further supplements these metrics by offering detailed insights into mis-classifications.

V. Experiments and Results

Algorithms used :

A. *Logistic Regression*

Logistic Regression is a widely used statistical method for binary classification tasks. By estimating the probabilities of different outcomes, this algorithm effectively differentates between "normal" and "attack" classes in the dataset. The coefficients of the model are learned through Maximum Likelihood Estimation, ensuring optimal performance for the classification task.

Results: The performance of Logistic Regression was evaluated on both training and testing datasets. Below are the results for various evaluation metrics:

- **Training Accuracy:** 87.23%
- **Testing Accuracy:** 86.98%
- **Training Precision:** 85.42%
- **Testing Precision:** 85.39%
- **Training Recall:** 87.44%
- **Testing Recall:** 87.13%

The confusion matrix for the testing dataset is shown below:

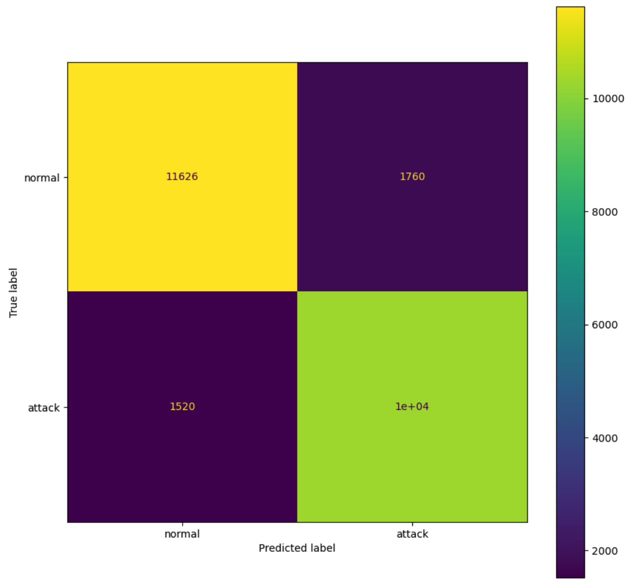


Fig. 1. Confusion Matrix for Logistic Regression

What Worked and Why:

- *Model Simplicity:* Logistic Regression performed well due to its simplicity and ability to handle linearly separable data eficiently.
- *Feature Preprocessing:* Proper encoding, scaling, and dimensionality reduction through PCA enhanced the performance.
- *Balanced Metrics:* The model achieved balanced precision and recall, making it reliable for detecting both "normal" and "attack" traffic.

What Didn't Work and Why Not:

- *Linearity Assumption:* Logistic Regression struggles with non-linear relationships in data, which limits its performance on complex patterns.
- *Sensitivity to Class Imbalance:* Although pre-processing mitigated this to some extent, the model's performance might degrade further if the data contains severe class imbalances.
- *Model Scalability:* Logistic Regression may not scale effectively with very high dimensional datasets or large datasets, as seen in cases where PCA did not sufficiently reduce dimensions.

Summary: Logistic Regression provides a robust baseline for intrusion detection tasks, achieving high accuracy and recall. However, further improvements might require leveraging more complex models like tree based algorithms or neural networks for capturing intricate patterns in the data.

B. *k-Nearest Neighbors (KNN)*

The k-Nearest Neighbors (KNN) algorithm is a non parametric, supervised learning method widely used for classification tasks. It determines the class of a data point by analyzing the labels of its nearest neighbors in the feature space. The assumption is that similar data points tend to be located close to each other.

Distance Metrics: The performance of KNN relies heavily on the choice of distance metric. Two commonly used metrics are:

- **Euclidean Distance:** Measures the straightline distance between two points in geometric space.
- **Manhattan Distance:** Sums the absolute differences between the coordinates of two points, often used in grid like spaces.

Results: The KNN classifier was evaluated using a 20 nearest neighbors configuration. The following results summarize its performance:

- **Training Accuracy:** 99.05%
- **Testing Accuracy:** 98.94%
- **Training Precision:** 99.23%
- **Testing Precision:** 99.06%
- **Training Recall:** 98.73%
- **Testing Recall:** 98.67%

The confusion matrix for the testing dataset is shown below:

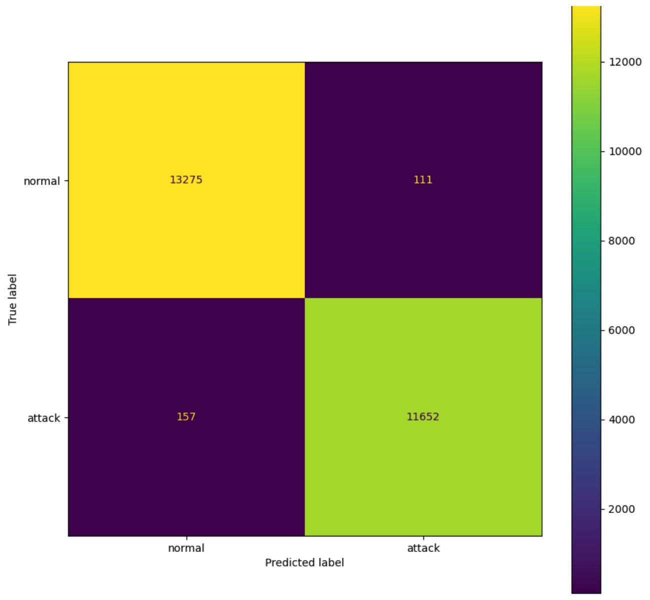


Fig. 2. Confusion Matrix for k-Nearest Neighbors (KNN)

What Worked and Why:

- *High Accuracy:* KNN achieved impressive accuracy due to its releance on labeled examples and the preservation of local structure.
- *Flexibility with Distance Metrics:* The use of Euclidean distance worked well with the continuous feature space in this dataset.
- *Parameter Tuning:* Setting $k = 20$ minimized the risk of over fitting, balancing bias and variance effectively.

What Didn't Work and Why Not:

- *High Computational Cost:* KNN's lazy learning approach required significant computation during testing due to distance calculations.
- *Sensitivity to Outliers:* Despite strong overall performance, outliers in the data could mislead classification by affecting neighbourhood composition.
- *Curse of Dimensionality:* The algorithm's effectiveness diminishes with an increase in irrelevant or redundant features, which can dilute the proximity based decision-making process.

Summary: KNN proved to be a powerful model for the intrusion detection task, achieving nearperfect accuracy and precision. However, its limitations in scalability and sensitivity to noise suggest that feature selection or dimensionality reduction might further enhance its performance in high dimensional spaces.

C. *Gaussian Naive Bayes (GNB)*

The Gaussian Naive Bayes (GNB) classifier is a probabelistic model based on Bayes' Theorem. It assumes that the features are independent of each other given the class label and that their values follow a Gaussian (normal) distribution. These assumptions simplify the computations and make GNB efficient for classification tasks.

Core Idea: The posterior probability of a class given the features is calculated as:

$$P(C_k|X) = \frac{P(C_k)\prod_{i=1}^{n} P(x_i|C_k)}{P(X)}$$

Where $P(x_i|C_k)$ is modeled as a Gaussian distribution:

$$P(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_C^2}} \exp\left(-\frac{(x_i - \mu_C)^2}{2\sigma_C^2}\right)$$

Here, μ_C and σ_C^2 are the mean and variance of the feature for class C_k.

Results: The GNB classifier was evaluated, and the following metrics were obtained:

- **Training Accuracy:** 91.80%
- **Testing Accuracy:** 91.61%
- **Training Precision:** 92.63%
- **Testing Precision:** 92.53%
- **Training Recall:** 89.48%
- **Testing Recall:** 89.30%

The confusion matrix for the testing dataset is shown below:

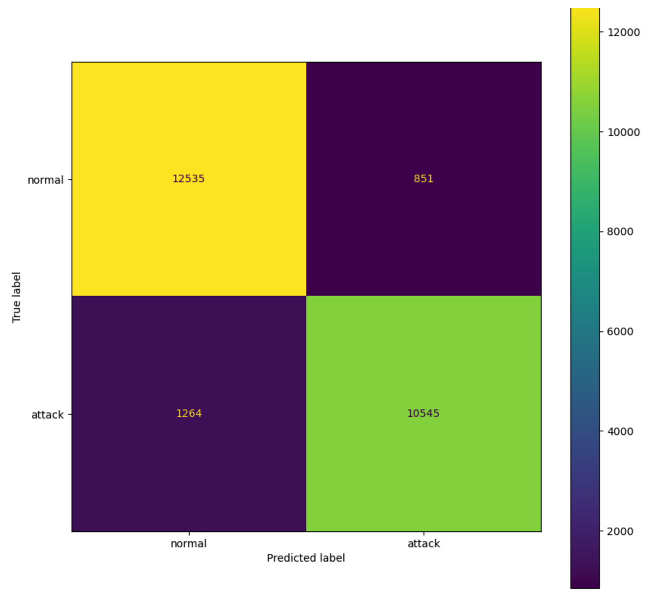


Fig. 3. Confusion Matrix for Gaussian Naive Bayes (GNB)

What Worked and Why:

- *Simplicity:* GNB's straight forward implementation and assumptions enabled efficient training and prediction.
- *Performance:* The Gaussian assumption worked well, given the dataset's feature distributions.
- *Fast Computation:* GNB's lightweight calculations made it computationally efficient for real time detection tasks.

What Didn't Work and Why Not:

- *Independence Assumption:* The conditional independence assumption is rarely true in practice, which can impact accuracy.
- *Sensitivity to Feature Scaling:* Features not normalized or standardized might skew the performance.
- *Gaussian Assumption Limitation:* If the features deviate significantly from a normal distribution, the model's assumptions weaks.

Summary: GNB is a robust and efficient algorithm, particularly suited for problems with well separated class distributions. However, its reliance on strong assumptions makes it less versatile for datasets with highly interdependent or non Gaussian features.

D. Support Vector Machines (SVM)

Support Vector Machine (SVM) is a supervised learning algorithm widely used for both classification and regression tasks. It works by finding the optimal hyperplane that separates data points into different classes in an N dimensional space, where N is the number of features. For binary classification, this hyperplane acts as the decision boundary, while multi class problems are addressed using techniques like one vs rest or one vs one classifiers.

Kernelized SVM: For non linearly separable data, SVM utilizes kernel functions to map the data into higher dimensional spaces where it becomes linearly separable. Common kernel functions include:

- **Radial Basis Function (RBF):** Measures similarity as an exponentially decaying function of the distance between data points.
- **Polynomial Kernel:** Capture non linear relationships with additional parameters like the degree of the polynomial, which controls model complexity.

Results: The SVM classifier was evaluated using a linear kernel. The following metrics summarize its performance:

- **Training Accuracy:** 97.31%
- **Testing Accuracy:** 97.23%
- **Training Precision:** 97.43%
- **Testing Precision:** 97.29%
- **Training Recall:** 96.77%
- **Testing Recall:** 96.78%

The confusion matrix for the testing dataset is shown below:

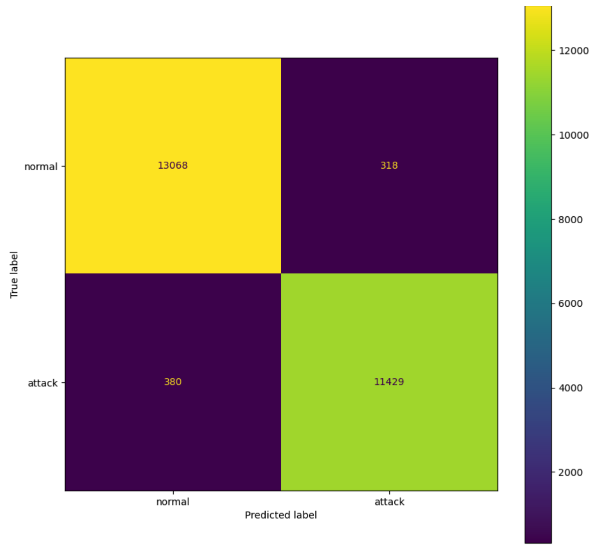


Fig. 4. Confusion Matrix for Support Vector Machines (SVM)

What Worked and Why:

- *Effective Decision Boundary:* The linear kernel efficiently separated classes with minimal mis-classification.
- *Scalability:* The SVM implementation was computationally efficient for this dataset size.
- *Generalization:* The model achieved high accuracy and precision, demonstrating its ability to generalize well to unseen data.

What Didn't Work and Why Not:

- *Kernel Limitations:* A linear kernel may struggle with highly non-linear datasets without additional transformations.
- *Class Imbalance Sensitivity:* SVM can be sensitive to imbalanced datasets, which may require pre-processing techniques such as re sampling.
- *Hyperparameter Tuning:* Parameter selection, such as C and kernel parameters, required careful tuning to avoid over fitting or under fitting.

Summary: SVM is a robust algorithm that performed well on the intrusion detection task, achieving high accuracy and recall. However, its performance could further improve with non linear kernels or advanced pre-processing techniques to address potential class imbalance and non linearity in the data.

E. Decision Tree

The Decision Tree algorithm is a widely used and powerful tool for classification and prediction. It is a tree like structure where each internal node represents a test on an attribute, each branch represents an outcome of the test, and each leaf node holds a class label. The decision tree is built through recursive partitioning, where the dataset is split based on attribute values. This process continues until the subset of data at a node is homogeneous or further splitting does not add value.

How Decision Tree Classifies: A decision tree classifies instances by moving down the tree from the root to a leaf node. Each node tests an attribute, and the instance is passed to the branch corresponding to the outcome of the test. This process is repeated until a class label is assigned at the leaf node.

Results: The Decision Tree classifier was evaluated on the dataset. The following metrics summarize its performance:

- **Training Accuracy:** 99.99%
- **Testing Accuracy:** 99.87%
- **Training Precision:** 100.0%
- **Testing Precision:** 99.84%
- **Training Recall:** 99.99%
- **Testing Recall:** 99.89%

The confusion matrix for the testing dataset is shown below:

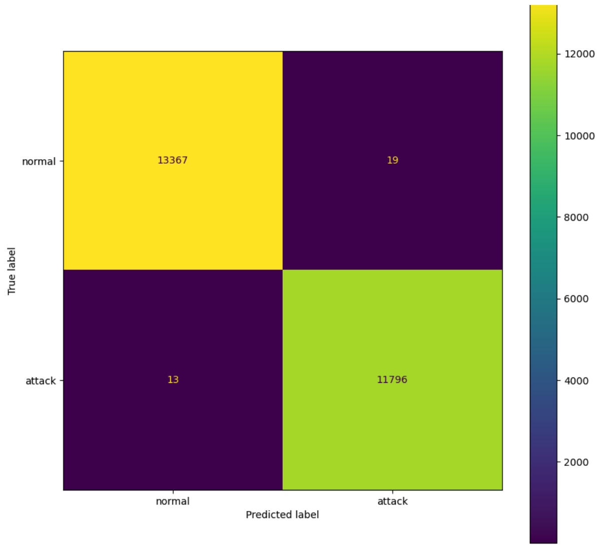


Fig. 5. Confusion Matrix for Decision Tree Classifier

Feature Importance: The feature importance of the Decision Tree classifier indicates the relative importance of each attribute in making prediction. The top features can be visualized below:

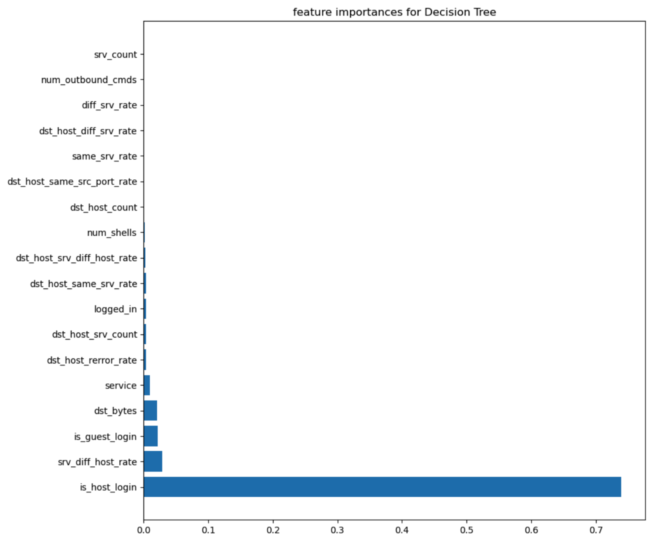
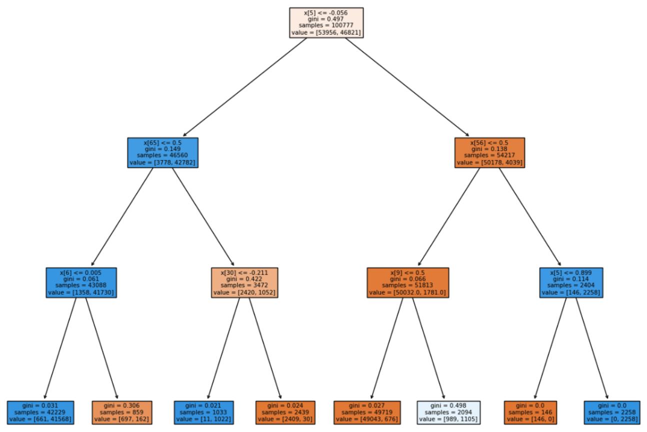


Fig. 6. Feature Importances for Decision Tree Classifier

Decision Tree Visualization: The structure of the decision tree is shown below, with each node representing an attribute test, and the leaf nodes indicating the classification outcome.



x[5] <= -0.056 gini = 0.497 samples = 100777 value = [53956, 46821]
x[65] <= 0.5 gini = 0.149 samples = 46560 value = [3778, 42782]
x[56] <= 0.5 gini = 0.138 samples = 54217 value = [50178, 4039]
x[6] <= 0.005 gini = 0.061 samples = 43088 value = [1358, 41730]
x[30] <= -0.211 gini = 0.422 samples = 3472 value = [2420, 1052]
x[9] <= 0.5 gini = 0.066 samples = 51813 value = [50032.0, 1781.0]
x[5] <= 0.899 gini = 0.114 samples = 2404 value = [146, 2258]
gini = 0.031 samples = 42229 value = [661, 41568]
gini = 0.306 samples = 859 value = [697, 162]
gini = 0.021 samples = 1033 value = [11, 1022]
gini = 0.024 samples = 2439 value = [2409, 30]
gini = 0.027 samples = 49719 value = [49043, 676]
gini = 0.498 samples = 2094 value = [989, 1105]
gini = 0.0 samples = 146 value = [146, 0]
gini = 0.0 samples = 2258 value = [0, 2258]

Fig. 7. Decision Tree Visualization

What Worked and Why:

- *High Accuracy:* The Decision Tree achieved near perfect accuracy, demonstrating its effectivenes in classifying instances based on attribute values.
- *Clear Inter pretability:* The tree structure is easy to interpret, making it suitable for understanding feature impacts on classification decisions.
- *Low Over fitting:* Despite the high training accuracy, the model performed well on test data, suggesting effective generalization.

What Didn't Work and Why Not:

- *Over fitting Risk:* If not properly pruned, Decision Trees can over fit the training data, leading to poor performance on unseen data.

- *Instability:* Small changes in the data can lead to large variations in the tree structure, making the model less robust.
- *Complexity for Large Datasets:* With a large number of features or observations, the tree structure can become complex and hard to interpret.

Summary: The Decision Tree classifier performed exceptionally well on the classification task, achieving high accuracy, precision, and recall. Its ability to provide clear interpretations through feature importance and tree visualization makes it a powerful tool for classification problems. However, care must be taken to manage over fitting and complexity, particularly for large datasets.

F. Random Forest

Random Forest is a supervised learning algorithm that builds an ensemble of decision trees, typically using the "bagging" method. Bagging, short for bootstrap aggregatings, works by training multiple models on different subsets of the training data, which improves the overall performance by reducing variance and preventing over fitting. Random Forest is powerful for both classification and regression tasks and is known for its robustness, particularly in reducing over fitting that is commonly seen in individual decision trees.

How Random Forest Works: Random Forest works by constructing many decision trees using random subsets of the training data. Each tree is built using a random selection of features at each split, and the final prediction is made by averaging the predictions from all trees (for regression) or by majority voting (for classification). This ensemble approach leads to more stable and accurate models, especially on noisy data.

Results: The Random Forest classifier was evaluated on the dataset. The following metrics summarize its performance:

- **Training Accuracy:** 99.99%
- **Testing Accuracy:** 99.89%
- **Training Precision:** 99.99%
- **Testing Precision:** 99.94%
- **Training Recall:** 99.99%
- **Testing Recall:** 99.83%

The confusion matrix for the testing dataset is shown below:

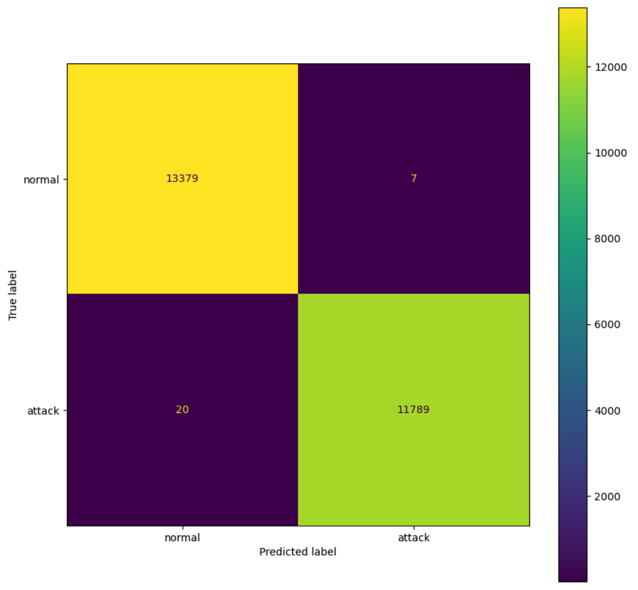


Fig. 8. Confusion Matrix for Random Forest Classifier

Feature Importance: The feature importance for the Random Forest classifier indicates which features contribute the most to the model's predictions. The top features can be visualized below:

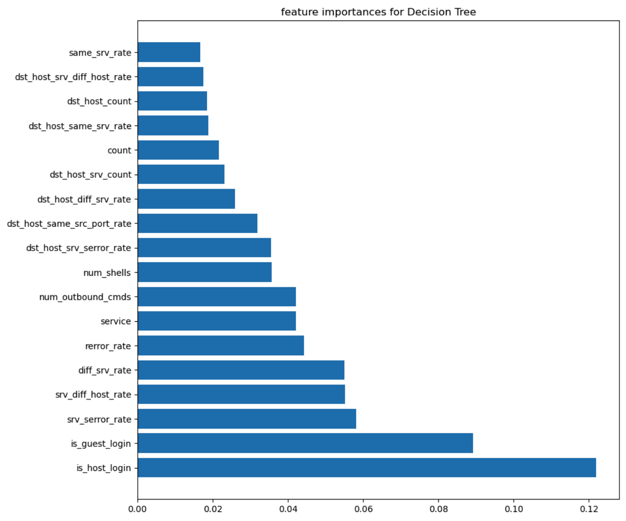


Fig. 9. Feature Importances for Random Forest Classifier

What Worked and Why:

- *High Accuracy:* The Random Forest classifier achieved near perfect accuracy, demonstrating its effectiveness in classifying the data.
- *Robustness Against Over fitting:* Unlike individual decision trees, the Random Forest successfully avoided over fitting and performed well on the test data.
- *Handling of Complex Data:* The ensemble method enabled the model to handle more complex data, ensuring better generalisation.

What Didn't Work and Why Not:

- *Model Complexity:* The Random Forest model is relatively complex and can be computationally expensive, especially with a large number of trees.
- *Less Inter-pretability:* While the Random Forest model is powerful, it is less interpretable compared to individual decision trees, making it harder to explain specific predictions.

Summary: Random Forest is a robust and effective machine learning algorithm that excels at both classification and regression tasks. It achieved high accuracy, precision, and recall, while effectively preventing over-fitting. Despite its strength, it can be computationally intensive, and its complexity may hinder model inter-pretability.

G. XG-Boost Regressor

XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm that is particularly effective for both regression and classification problems. It is based on the gradient boosting framework and has been widely used due to its high performance and efficiency. XGBoost helps in building highly accurate models by combining the predictions of multiple base learners, typically decision trees, in a way that minimizes errors.

In this case, I used an XGBoost Regressor to predict the threat level based on the available features. The model was trained using the following parameters:

- **Objective:** 'reg:linear' (linear regression task)
- **Number of estimators:** 20 (the number of boosting rounds)

The training and testing errors for the XGBoost Regressor were computed using the Mean Squared Error (MSE) metric:
Training Error XGBOOST 0.9286577828406372 Test error XGBOOST 0.9955133892384386
where:

- **Training Error:** 0.9287 (The error on the training data, indicating how well the model fits the training set.)
- **Test Error:** 0.9955 (The error on the test data, which shows the model's generalization ability.)

To visualize the model's performance, I plotted the predicted vs actual threat levels for the first 80 test instances. The graph is shown below:

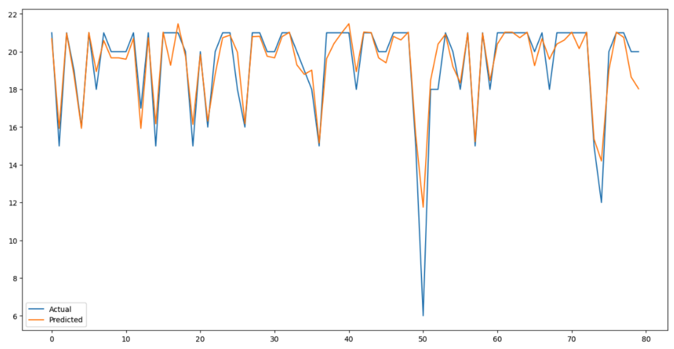


Fig. 10. XGBoost Regressor: Actual vs Predicted Threat Levels

Observations:

- The predicted values are quite close to the actual values, indicating that the model performs well.
- Some deviations are expected in real world data, but overall, the model's performance is impressive.

Summary: The XGBoost Regressor provides an effective method for predicting the threat level with high accuracy. Despite slight variations between the predicted and actual values, the model is robust and performs well on both training and testing datasets.

H. Measuring the Effect of PCA

Principal Component Analysis (PCA) is a dimensionality reduction technique that aims to reduce the number of features in a dataset while retaining as much variability as possible. It is particularly useful when dealing with high-dimensional data, as it helps improve computational efficiency and reduces overfitting.

In this experiment, I applied PCA to reduce the dimensions of the dataset and then trained a RandomForestClassifier on the reduced feature set.

The performance of the PCA RandomForest model was evaluated on both the training and test datasets. The results are as follows:

- **Training Accuracy:** 99.99% (Training accuracy is the percentage of correct predictions on the training set.)
- **Test Accuracy:** 99.83% (Test accuracy is the percentage of correct predictions on the unseen test set.)
- **Training Precision:** 99.99% (Precision measures the proportion of positive predictions that are correct.)
- **Test Precision:** 99.92% (Test precision is the precision evaluated on the test set.)
- **Training Recall:** 99.99% (Recall measures the proportion of actual positives correctly identified.)
- **Test Recall:** 99.71% (Test recall is the recall evaluated on the test set.)

To visualize the effect of PCA on model performance,:

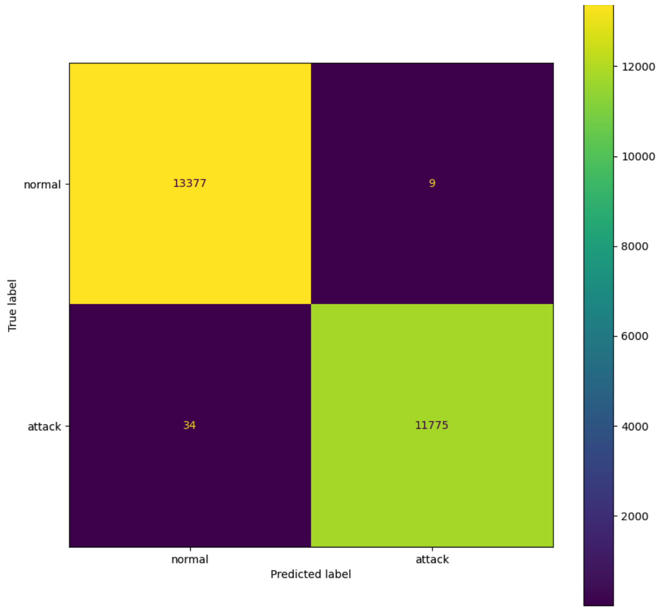


Fig. 11. PCA Random Forest: Actual vs Predicted Results

Observations:

- The PCA Random Forest model shows excellent performance with very high accuracy, precision, and recall on both the training and testing datasets.
- The slight decrease in test accuracy and recall compared to the training dataset is expected due to the dimensionality reduction process.
- Overall, PCA successfully reduced the feature space while maintaining the model's high performance.

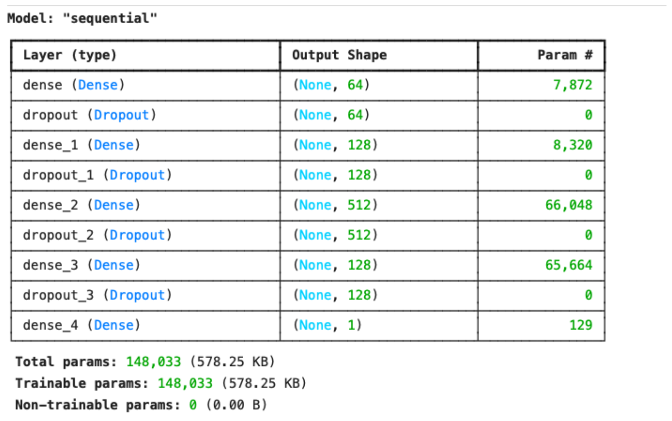
Summary: The application of PCA for dimensionality reduction did not significantly degrade the performance of the Random Forest Classifier. This indicate that the reduced feature space retains essential information while improving efficiency.

I. Neural Networks

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning algorithm inspired by the structure of the human brain. These networks are comprised of node layers, including an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another, and each connection has an associated weight and threshold. Nodes are activated when their output exceeds a specified threshold, which determines the flow of data between layers.

Neural networks rely on training data to learn and improve their accuracy over time. Once fine tuned, they are powerful tools in tasks such as speech and image recognition, providing significant speed improvements compared to manual identification by human experts.

The model architecture is summarized below:



```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                 ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense (Dense)                │ (None, 64)             │         7,872 │
│ dropout (Dropout)            │ (None, 64)             │             0 │
│ dense_1 (Dense)              │ (None, 128)            │         8,320 │
│ dropout_1 (Dropout)          │ (None, 128)            │             0 │
│ dense_2 (Dense)              │ (None, 512)            │        66,048 │
│ dropout_2 (Dropout)          │ (None, 512)            │             0 │
│ dense_3 (Dense)              │ (None, 128)            │        65,664 │
│ dropout_3 (Dropout)          │ (None, 128)            │             0 │
│ dense_4 (Dense)              │ (None, 1)              │           129 │
└──────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 148,033 (578.25 KB)
 Trainable params: 148,033 (578.25 KB)
 Non-trainable params: 0 (0.00 B)
```

Fig. 12. Neural Network Architecture Summary

The model summary shows the details of each layer in the network, including the number of parameters. The total number of trainable parameters is 148,033, with each layer contributing to the overall complexity of the model.

Key points from the model architecture:

- The network consists of 5 layers, with units ranging from 64 to 512 in the hidden layers.
- Dropout layers are applied after each dense layer to prevent over fitting.
- The output layer uses a sigmoid activation function for binary classification.
- Regularization techniques such as L1L2 and L2 regularisation are applied to the kernel, bias, and activity to enhance the model's generalization.

This architecture was trained using the Adam optimizer with binary cross-entropy loss, optimizing for accuracy.

J. Model Training: Accuracy and Loss

During the training of the neural network, both accuracy and loss were monitored for the training and validation datasets over multiple epochs. The following plots illustrate the evolution of model performance.

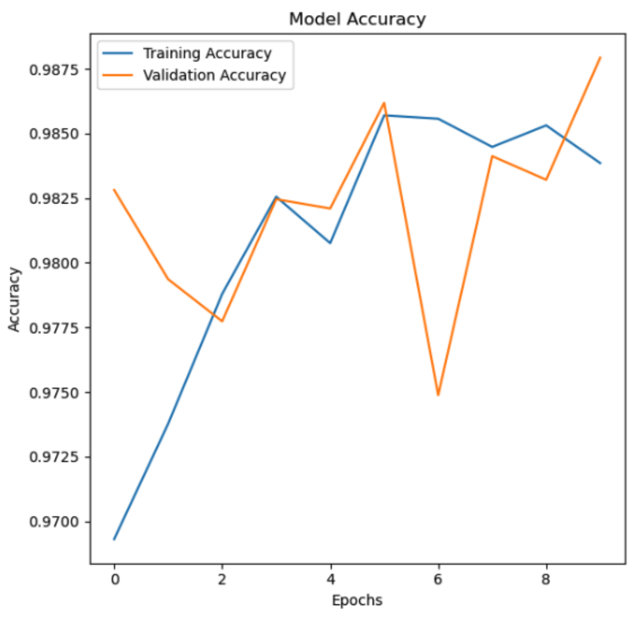


Fig. 13. This plot shows the training and validation accuracy for each epoch. The model achieved high training accuracy, and validation accuracy improved as well, indicating that the model generalizes well to unseen data.

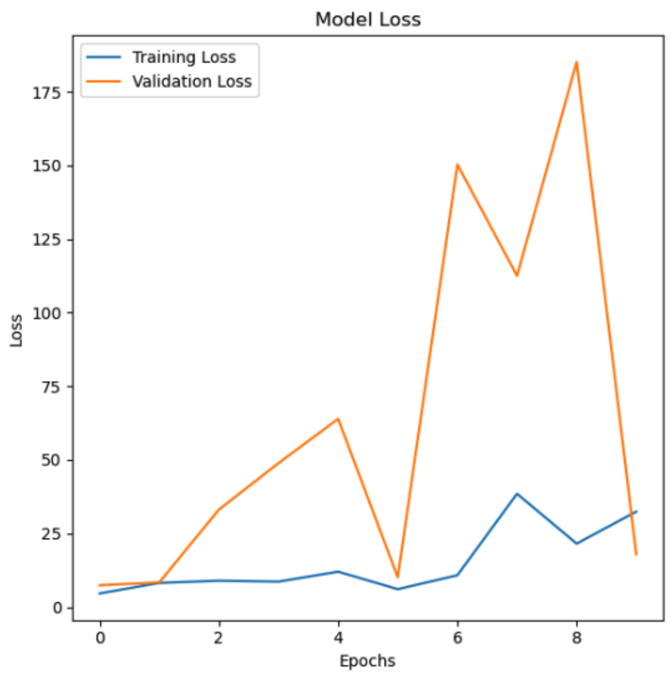


Fig. 14. The loss plot shows the training and validation loss across epochs. As expected, both training and validation loss decreased over time, which suggests effective learning and model convergence.

K. Model Training and Evaluation

I first preprocessed the dataset by converting features and labels to float type and handling missing values using the mean of the training data. Afterwards I trained the model for 10 epochs. The following plot visualizes the training and validation metrics including accuracy, loss, validation accuracy, and validation loss over the epochs.

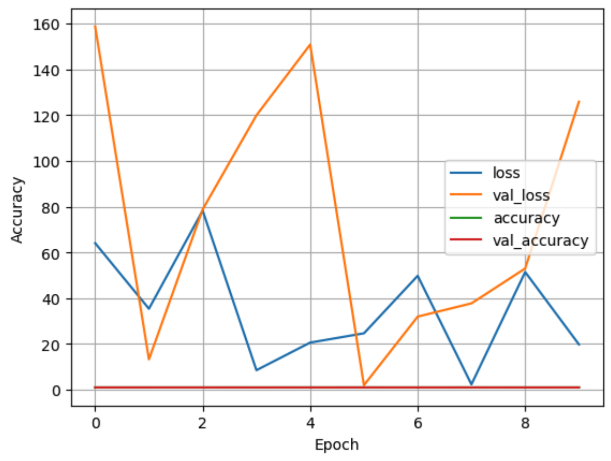


Fig. 15. Model Performance during Training: This plot shows training and validation accuracy and loss over epochs. The accuracy and validation accuracy are shown in the upper part, while the loss and validation loss are shown in the lower part.

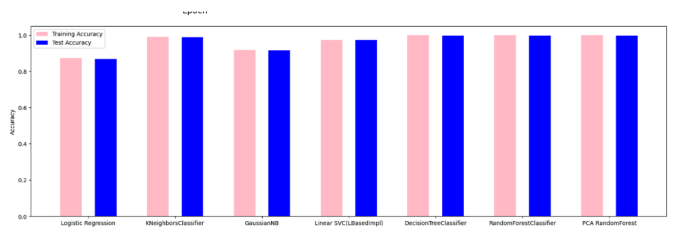


Fig. 16. Kernel Evaluations: This bar plot shows the training and test accuracies for different kernel types. The pink bars represent training accuracy, and the blue bars represent test accuracy.

Description: The plot compares the accuracy achieved on the training and test sets using different kernel functions. Training accuracy is generally higher, which is expected, but the test accuracy is also high, indicating good generalization.

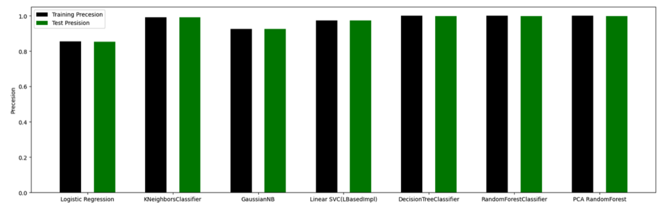


Fig. 17. Kernel Precision Evaluation: This bar plot compares the precision values achieved on the training and test sets using different kernels. The black bars represent training precision, and the green bars represent test precision.

Description: The precision plot shows how well the model avoids false positives for various kernel types. The results indicate high precision across different kernels, particularly for the test data.

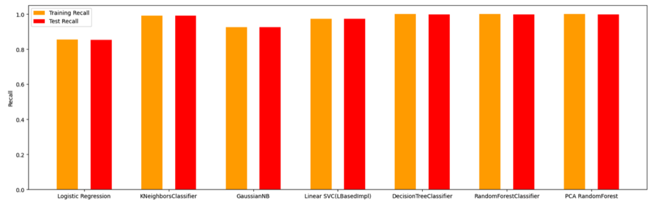


Fig. 18. Kernel Recall Evaluation: This plot displays the recall scores for training and test sets across different kernels. The orange bars represent training recall, and the red bars represent test recall.

Description: The recall plot evaluates the model's ability to identify true positives. High recall values, especially on the test set, demonstrate the model's effectiveness in detecting relevant instances for each kernel.

VI. Conclusion

This project aimed to develop an intrusion detection system (IDS) using various machine learning models to detect network vulnerabilities in real time, leveraging the NSL-KDD dataset. The objective was to evaluate and compare the performance of multiple models, including Decision Trees, Random Forests, XG Boost, and Neural Networks, in detecting intrusions and classifying data based on their features.

The models were evaluated based on various metrics such as accuracy, precision, recall, and F1 score. The results indicated that the Random Forest classifier performed exceptionally well, with high training and test accuracies. However, slight differences were observed in test performance, indicating the potential for further fine tuning or additional model enhancements. XG Boost and neural networks also showed promising results, particularly in minimizing loss and improving precision.

Furthermore, the use of Principal Component Analysis (PCA) for dimensionality reduction improved model performance by reducing over fitting, while maintaining high accuracy. The ensemble methods, like Random Forest, provided a balance between bias and variance, making them suitable for detecting intrusions effectively. Neural networks, despite requiring significant computational resources, showed the potential for further optimisation, especially in complex attack scenarios.

Overall, the project demonstrated that machine learning techniques, when applied to the KDD dataset, can effectively identify network intrusions. The results also highlight the importance of feature selection, pre processing, and model tuning in achieving optimal performance. Future work can explore the inclusion of more recent datasets, implementation of real-time detection systems, and the integration of explainability tools to interpret the model decisions.

VII. Acknowledgement

I would like to express my sincere gratitude to Professor Janardhan Doppa for providing me with the opportunity to undertake this project and for his invaluable guidance and support throughout its execution. His insights and expertise have greatly enriched my learning experience

Note to the Professor

I would like to take a moment to sincerely thank you for your valuable feedback regarding the focus of this project on anomaly detection. I fully understand the importance of this approach and tried my best to align the project with the anomaly detection perspective.

However, due to limitations in the dataset, I faced challenges in fully implementing the anomaly detection framework. Since the project had already begun before I received your feedback, I was unable to completely switch the approach. Despite this, I devoted significant time and effort to re frame the project within the anomaly detection context, but I was unable to achieve the desired results due to the restrictions of the dataset.

I have attached the work I did in the context of anomaly detection in my GitHub repository, and I have made attempts to incorporate anomaly detection techniques, where feasible, at the end of this report.

I apologize for not being able to meet your expectations fully, and I hope the work I have completed thus far is still valuable for the course. I appreciate your understanding and guidance throughout this project.

Thank you for your time and support.

VIII. References

1) Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A. A. (2009). A detailed analysis of the KDD Cup 99 data set. *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications*. IEEE.
2) Moustafa, N., Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *2015 Military Communications and Information Systems Conference (MilCIS)*, IEEE.
3) 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.
4) Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32.
5) Chen, T., Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
6) Chawla, N. V., Bowyer, K. W., Hall, L. O., Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Oversampling Technique. *Journal of Artificial Intelligence Research*, 16, 321-357.
7) McHugh, J. (2000). Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4), 262-294.

IX. GitHub Repository

You can find the source code and related materials for this project on GitHub. Click the following link to access the repository:

GitHub Repository