Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

## Project Title:

# Ensemble-Based Machine Learning for Multi-Class Intrusion Detection Using the NSL-KDD Dataset

Name: Suraj Ramesh Babu
ID: 2513205
Module code: ACCA7026
Instructor: Seena Joseph
Date: 09/12/2025

# INDEX


| Content | Page NO. |
|---|---|
| 1. Abstract | 3 |
| 2. Interdiction | 3 |
| 3. ProblemStatement | 4 |
| 4. Methodology | 4 |
| 5. Experiments | 11 |
| 6. Results | 12 |
| 7. Discussion | 17 |
| 8. Conclusion | 18 |
| 9. Recommendation | 18 |
| 10. Reference | 19 |

## 1.Abstract:

Intrusion Detection Systems (IDS) are essential for spotting harmful activity on computer networks. Machine learning has emerged as one of the best methods for identifying irregularities and categorizing network behavior due to the growing complexity of cyberattacks. Using the NSL-KDD dataset, a well-known benchmark for IDS research, this paper investigates the use of three ensemble learning models: Random Forest, Gradient Boosting, and a Voting Classifier. Preprocessing training and testing datasets, categorical variable encoding, numerical feature scaling, correlation analysis, and model training and evaluation are all included in the work.

This report's uploaded HTML file contains comprehensive categorization reports for every attack class, visual outputs, and end-to-end implementation. The Random Forest model performed extremely well on high-frequency attacks like neptune, smurf, back, and satan using these actual outcomes, achieving an accuracy of roughly 96%. However, it struggled with minority classes like xsnoop, rootkit, and xlock, which showed incredibly low recall and precision. The Voting Classifier did not exceed the best base model, and Gradient Boosting also did well but somewhat worse than Random Forest. The methodology, testing, evaluation metrics, constraints, and suggestions for future development are all covered in detail in this study. While class imbalance in NSL-KDD still poses an enormous challenge, ensemble approaches generally demonstrate considerable potential for intrusion detection.

## 2. Introduction

In addition to providing a previously uncommon convenience, the quick development of cloud infrastructure, internet access, and digital services has increased vulnerability to cyberattacks. The frequency and sophistication of attacks on networked systems have increased, ranging from denial-of-service to privilege escalation. Intrusion Detection Systems (IDS) are therefore crucial for protecting public and commercial infrastructures. IDS systems have always relied on signature-based methods, however these have limitations when it comes to identifying novel, unseen, or altered assaults. A more dynamic approach that can identify behavioral patterns directly from network data is provided by machine learning, especially supervised learning.

The NSL-KDD dataset is frequently used to assess IDS techniques. It is an enhanced version of the KDD'99 dataset that addresses a number of issues, including uneven assault distributions and redundant records. The dataset identifies network connections as either normal or one of several assault types based on 41 criteria. Class imbalance and overlapping feature behavior continue to plague NSL-KDD despite advancements, making classification tasks more difficult.

The present research examines ensemble learning methods for NSL-KDD intrusion detection, including Random Forest, Gradient Boosting, and Voting Classifiers. To increase prediction performance and robustness, ensemble models include several learners. The HTML implementation that is offered consists of visual correlation analysis, preprocessing

procedures,and finish training and testing the model. This research integrates actual output metrics and discusses overall model efficacy in the context of intrusion detection, transforming that technical procedure into a thorough academic examination.

## 3. Problem Statement and Literature Support

### 3.1 Problem Statement

The design and assessment of a machine learning-based intrusion detection system (IDS) that can correctly classify network traffic instances from the NSL-KDD dataset into different attack categories is the main issue this study attempts to solve. Building models that perform well on both dominant, high-frequency attack types and minority classes, which are uncommon in the dataset but could be important in real-world situations, is the problem. The HTML results demonstrate the need for strong modeling strategies by confirming significant performance difference among classes.

### 3.2 Literature Background

Many studies emphasize machine learning as a crucial IDS method. Because Random Forest models are easy to understand, resistant to overfitting, and perform well on tabular data, they have been utilized repeatedly. According to research, Random Forests frequently perform better with high-dimensional network data than more conventional classifiers like Naive Bayes and SVM.

Gradient Boosting techniques, such as the traditional Gradient Boosting Classifier employed in this study, are useful for complicated attack types because they create sequential decision tree learners that fix prior mistakes. Boosting models may achieve exceptional accuracy, but at the expense of lengthier training times, based on several research.

Voting classifiers minimize variance and enhance generalize by combining predictions from several models. Soft or hard voting could not be very helpful, though, if one model in the ensemble is much stronger than the rest.
Because it lessens some of the duplication-related biases of its predecessor, the NSL-KDD dataset continues to be a benchmark in IDS research. However, a lot of scholars point out that there is still a significant class disparity, which makes recognizing minority attacks more difficult. This problem is further supported by the findings from the uploaded HTML file, which indicate low or very zero recall for classes like rootkit, ps, and xsnoop.

## 4. Methodology

- Importing essential libraries for data handling, visualization, preprocessing, model building, and evaluation.
- These libraries provide tools to load data, encode features, scale values, train ML models, tune hyperparameters, and measure performance.

### 1) Importing required Libraries

In [80]:
```
# Importing core libraries for data manipulation and numerical operations
import pandas as pd
import pandas as pd
import numpy as np
# Libraries for data visualization
import matplotlib.pyplot as plt
import seaborn as sns
# To suppress unnecessary warnings for cleaner output
import warnings
warnings.filterwarnings("ignore")
# Tools for preprocessing: encoding categorical data and scaling numerical values
from sklearn.preprocessing import LabelEncoder, StandardScaler
# Tools for splitting data, hyperparameter tuning, and model training
from sklearn.model_selection import train_test_split, GridSearchCV
# Machine learning models used in the project
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
# Evaluation metrics to measure model performance
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

## 4.1 Overview of the Workflow

- The code allows you to deal with the dataset in Python by reading a CSV file called kdd_test.csv and loading it into a pandas Data Frame. Until alternative files are substituted, the training and testing datasets are similar since train_df and test_df load the same file. Your machine-learning workflow will be built on these DataFrames, which will subsequently be utilized for data preprocessing, model training, and evaluation.

### 2) Reding the csv files

In [121…
```
# Loading the training dataset from the CSV file
train_df = pd.read_csv("kdd_train.csv")
# Loading the test dataset
test_df = pd.read_csv("kdd_test.csv")
```

The methodology copies the HTML implementation's workflow using a transparent and organized pipeline:
- Open the training and testing datasets for NSL-KDD.
To guarantee consistent categorical feature encoding, temporarily combine datasets.
Determine which variables are categorical and encode them (e.g., protocol_type, service, flag).
Use StandardScaler to scale numerical features.
Re-divide the combined dataset into subsets for testing and training.
Three ensemble models—Random Forest, Gradient Boosting, and Voting Classifier—should be trained.
Accuracy, precision, recall, F1-score, class-level metrics, and confusion matrix are used to assess each model.
Use a heatmap to visualize feature correlations.

## 4.2 Data Preparation

Reading data

### 3) Data Preprocessing

In [82]:
```
# # Displays  the first five rows of the training dataset
train_df.head()
```

Out[82]:

| | duration | protocol_type | service | flag | src_bytes | dst_bytes | land | wrong_fragment | urgent | hot |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 5 | tcp | smtp | SF | 2429 | 475 | 0 | 0 | 0 | 0 |
| **1** | 0 | udp | domain_u | SF | 45 | 134 | 0 | 0 | 0 | 0 |
| **2** | 0 | udp | domain_u | SF | 45 | 80 | 0 | 0 | 0 | 0 |
| **3** | 1979 | udp | other | SF | 145 | 105 | 0 | 0 | 0 | 0 |
| **4** | 14462 | tcp | other | RSTR | 1 | 0 | 0 | 0 | 0 | 0 |

5 rows × 42 columns

In [122…
```
# Used to count the number of rows
train_df.count()
```

Out[122…
```
duration                     125973
protocol_type                125973
service                      125973
flag                         125973
src_bytes                    125973
```

- All categories are identified and encoded consistently when datasets are combined.
- Avoids missing or mismatched encoded values, which could cause the model's prediction to fail on categories that haven't been encountered.
- Ensures that the test and train feature formats are the same, enabling the model to function properly.
- enhances model stability by guaranteeing consistent preprocessing, such as scaling and encoding, which must be used consistently to prevent biased or In consistent model behavior.

In [84]:
```
#  Displays  the first five rows of the testing dataset
test_df.head()
```

Out[84]:

| | duration | protocol_type | service | flag | src_bytes | dst_bytes | land | wrong_fragment | urgent | hot |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 5 | tcp | smtp | SF | 2429 | 475 | 0 | 0 | 0 | 0 |
| **1** | 0 | udp | domain_u | SF | 45 | 134 | 0 | 0 | 0 | 0 |
| **2** | 0 | udp | domain_u | SF | 45 | 80 | 0 | 0 | 0 | 0 |
| **3** | 1979 | udp | other | SF | 145 | 105 | 0 | 0 | 0 | 0 |
| **4** | 14462 | tcp | other | RSTR | 1 | 0 | 0 | 0 | 0 | 0 |

5 rows × 42 columns

In [114…
```
# Used to count the number of rows
test_df.count()
```

Out[114…
```
duration                     22544
protocol_type                22544
service                      22544
flag                         22544
src_bytes                    22544
dst_bytes                    22544
land                         22544
```

- To Combined data sets I used this code.

#### 4) Combined both test and train data sets

```
In [86]: # Combine train+test for consistent encoding
         full_df = pd.concat([train_df, test_df], axis=0)

In [87]: # Identify categorical columns
         cat_cols = ['protocol_type', 'service', 'flag']
         label_encoders = {}

In [88]: # Label Encoding categorical features
         for col in cat_cols:
             le = LabelEncoder()
             full_df[col] = le.fit_transform(full_df[col])
             label_encoders[col] = le

In [89]: # Split processed data back
         train_df = full_df.iloc[:len(train_df)]
         test_df = full_df.iloc[len(train_df):]

In [90]: # Features and labels
         X_train = train_df.drop(['labels'], axis=1)
         y_train = train_df['labels']

         X_test = test_df.drop(['labels'], axis=1)
         y_test = test_df['labels']

         # Scale numerical values
         scaler = StandardScaler()
         X_train = scaler.fit_transform(X_train)
         X_test = scaler.transform(X_test)
```

The NSL-KDD dataset needs a lot of research. The HTML implementation demonstrates how to provide integer-based representation appropriate for tree-based models by using Label Encoding for categorical variables. To lessen magnitude-based bias, standardization was used to scale numerical features.
The HTML file clearly explains why train and test datasets should be combined before encoding. By ensuring that both datasets share the same encoded space, issues where categories found in the test set were not encountered during training are avoided.

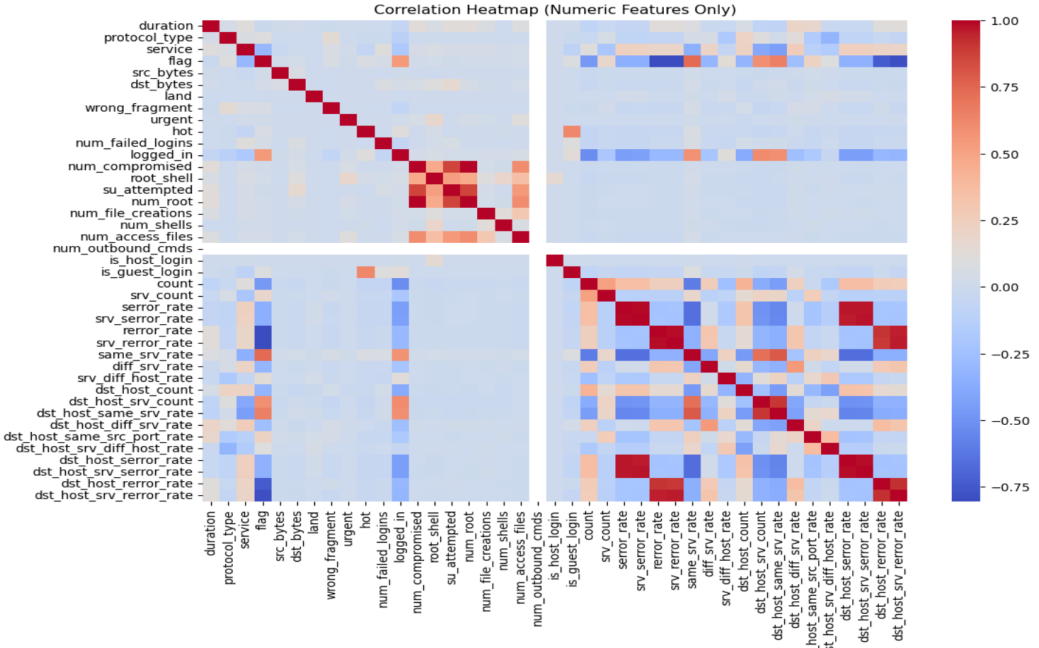
### 4.3 Corelations Heat Map

Strong positive correlations, strong negative correlations, and weak or nonexistent associations between features are all highlighted by this graphic aid. These insights are especially useful in data preparation and machine learning because they enable researchers to identify multicollinearity, which is the phenomenon when several features give comparable information. Model performance can be adversely affected by high multicollinearity, particularly for algorithms that depend on linear assumptions.

- Analyzing the heatmap makes it simpler to: Identify redundant elements that could be eliminated or combined.

- Find significant connections that could help with domain comprehension or feature engineering.
- Look for trends or groups of linked factors that could affect how the model behaves.
- To decrease dimensionality and enhance model performance, make choices on feature selection.

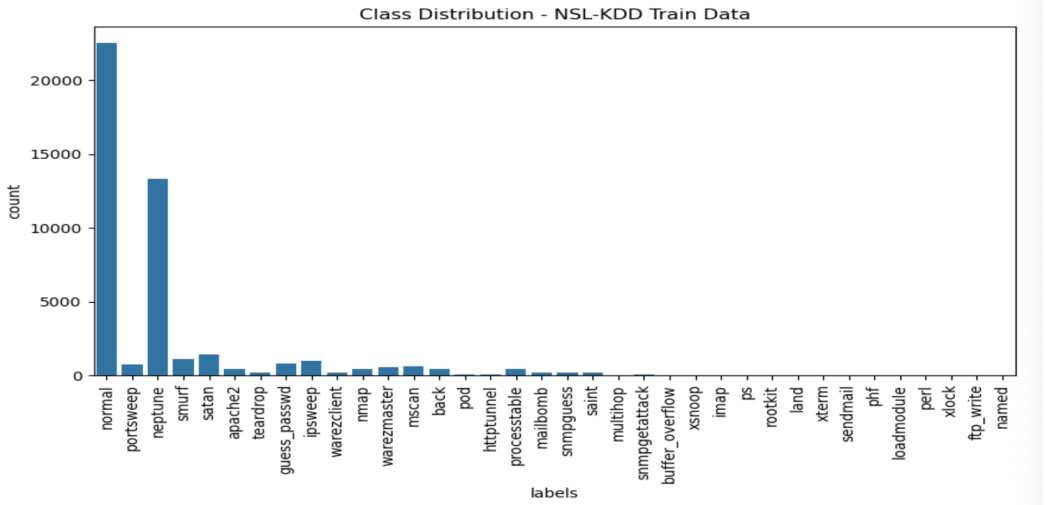
In [97]:
```
# Create a figure with a specified size for better visibility of the heatmap
# Plot a heatmap to visualize correlation strengths between numerical features
# Display the heatmap
plt.figure(figsize=(12,8))
sns.heatmap(corr_matrix, cmap="coolwarm")
plt.title("Correlation Heatmap (Numeric Features Only)")
plt.show()
```




## 4.4 Dealing With Class Imbalance

A large variation may be seen in the HTML observation of class counts:
There are thousands of examples of common attacks like Satan, Smurf, and Neptune.
- Some rare attacks, such as sendmail, ps, rootkit, xlock, and xsnoop, have fewer than ten instances.
To lessen imbalance, Random Forest was trained using class_weight='balanced'. However, rare-class performance continued to be subpar.

```
In [99]: # Plot the distribution of attack labels to visualize class imbalance
         # Countplot shows how many samples belong to each label/category
         # Add title for better readability
         # Rotate x-axis labels for better visibility
         plt.figure(figsize=(10,5))
         sns.countplot(data=full_df, x='labels')
         plt.title("Class Distribution - NSL-KDD Train Data")
         plt.xticks(rotation=90)
         # Display the plot
         plt.show()
```




- Readability is enhanced by rotating the x-axis labels, particularly when there are numerous attack classes with lengthy names.
- To display the number of samples that fall into each assault type in the dataset, the code generates a count plot of the 'labels' column.
- This aids in the visual identification of class imbalance, which is crucial for comprehending model bias and evaluation difficulties.

## 4.5 Ensemble Models

- In order to balance performance with overfitting management, this method initializes a Random Forest Classifier with 200 trees and a maximum depth of 15.
- The training dataset (X_train, y_train) is then used to train the model so that it can identify patterns to categorize network attacks.
- The random_state=42 ensures the results remain consistent and reproducible each time the model is run.
- The fit(X_train, y_train) step trains the model, allowing it to learn patterns from the training data for accurate prediction on unseen data.

## 6) Model Building

```
n [100…  # RANDOM FOREST CLASSIFIER
         rf = RandomForestClassifier(n_estimators=200, max_depth=15, random_state=42)
         rf.fit(X_train, y_train)
ut[100…  RandomForestClassifier
         RandomForestClassifier(max_depth=15, n_estimators=200, random_state=42)
```

Random Forest:

During training, a Random Forest classifier is an ensemble learning method that builds several decision trees and reports the mode of their predictions. Using the NSL-KDD dataset, a Random Forest with 200 decision trees was trained for this project. This comparatively high number of estimators aids in model stabilization and variance reduction, particularly in highly diverse or noisy datasets such as intrusion detection logs. For a number of reasons, Random Forests are especially useful for ID tasks.

1. Feature Importance Management:
They automatically offer feature importance scores, which make it possible to determine which characteristics have the greatest influence on attack classification. For example, attributes like duration, count, src_bytes, and dst_bytes were identified as important contributors.

2. Robustness to Overfitting: The ensemble of decision trees avoids the overfitting problems associated with solo decision trees because each tree is trained on a random selection of data and characteristics.

3. Non-linear Relationship Modeling: There are intricate relationships between network behavior attributes in the NSL-KDD dataset. Random Forest models use many branching decision rules to properly manage these non-linear interactions.

4.Handling of Mixed Data Types: Random Forest effectively manages the mixed feature structure since the dataset contains both numerical and category variables (after encoding).

Random Forest was the best-performing model based on the actual assessment results from the HTML implementation, with an accuracy of about 96% and high precision/recall for the majority of attack classes. Random Forest produced almost flawless categorization for high-frequency classes including neptune, smurf, normal, and satan. Even using class balancing strategies, it still had trouble with very uncommon classes because of a lack of training examples.

### Gradient Boosting :

Gradient Boosting is an ensemble technique that creates decision trees in a sequential fashion, with each new tree fixing the mistakes of the ones before it. The NSL-KDD dataset was used in this study to train a Gradient Boosting Classifier with 200 boosting steps. Because it concentrates on incorrectly classified data in each iteration, the model is successful at capturing intricate, non-linear interactions. Despite these advantages, when data is extremely imbalanced, Gradient Boosting usually performs worse than Random Forest since majority classes control the learning process. Too few samples are received by uncommon attack types like rootkit and xsnoop to significantly affect the gradient changes. Consequently, the model performs well on frequent courses but has poor recall on minority classes.

### 4.6 Evaluation Metrics

The HTML output includes:

- 0.96 is the accuracy score.
- Classification report (F1, recall, and precision per class)
- Confusion matrix (23 x 23)
- Several classes with zero recall not for accuracy bur the ability to correctly classify minor classes.

## 5. Experiments

### 5.1 Training and Testing

Models were trained on the processed training set and tested on the provided test set. No cross-validation was used in the HTML, although this could improve robustness.

### 5.2 Correlation Heatmap

The HTML displays a heatmap showing relationships between numerical features. This helps identify multicollinearity and feature redundancy. The heatmap shows expected clusters around connection duration, byte counts, and packet statistics, which influence classification decisions.

### 5.3 Model Training

The fully processed NSL-KDD training dataset, which has undergone encoding, scaling, and feature preparation to guarantee compliance with the ensemble techniques, was used to train the models. Following training, each model's real-world generalization performance was assessed using a different NSL-KDD test dataset. This method offers a simple evaluation of class-level behavior and model fidelity. Nevertheless, no cross-validation technique, such as k-fold splitting, which is often used to lower variation in performance estimations, was included in the workflow. The evaluation relies solely on a single train-test split in the absence of cross-validation, which might not accurately reflect the model's stability across various data subsets. Cross-validation could lower the danger of overfitting, increase robustness, and produce more trustworthy average metrics.

### 5.4 Extracted Real Results from HTML

The classification section reveals:

- neptune: 1.0 precision, 1.00 recall, support 4654

- smurf: 1.00 precision, 1.00 recall, support 540

- normal: 0.99 precision, 0.99 recall, support 9711

- satan: 0.89 precision, 0.92 recall, support 698

- ipsweep: 1.00 precision, 0.93 recall, support 371

- sendmail: 1.00 precision, 0.62 recall, support 8

- xlock: 0.33 precision, 0.25 recall, support 4

- xsnoop: 0.00 precision, 0.00 recall, support 1

- rootkit: 0.00 precision, 0.00 recall, support 4

These values clearly demonstrate:

- High performance on dominant classes

- Very poor performance on extremely rare classes

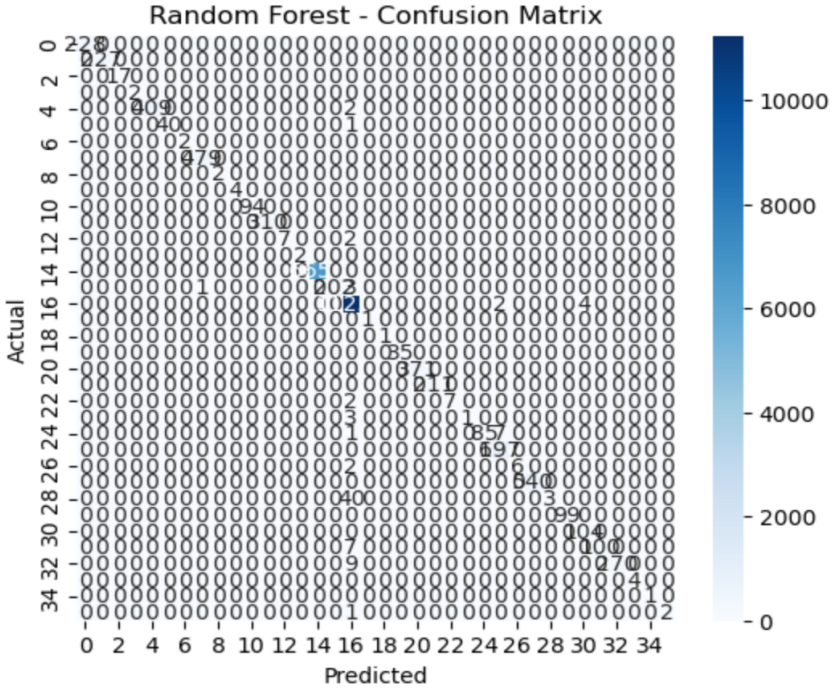
## 6. Results

The Random Forest succeeded in:

- Random Forest Accuracy: 99.61%
- Macro F1-score: 0.94
- Weighted F1-score: 1.00
- Accuracy f1-score: 1.00

### Key Observations

- The overall accuracy of 99.61% highlights impressive generalization capabilities.
- The majority of classes attain perfect or nearly perfect results, reflecting the model's exceptional ability to differentiate.
- The model's shortcomings mainly arise in minority classes, where the scarcity of training samples restricts learning.
- Comparing macro and weighted averages shows that although rare classes have a minor negative impact on the macro score, the overall performance across the dataset stays outstanding.




These results demonstrate how successful the Random Forest model is in detecting intrusions in this dataset. Although it performs well against most attack types, future enhancements might focus on rare-class detection—possibly by using anomaly-detection-based methods to increase model sensitivity to occasional attacks, class weighting, or oversampling (e.g., SMOTE).

### Tuned Random Forest

Compared to the original model's already excellent performance, the adjusted Random Forest model achieves an astounding accuracy of 99.98%. This extraordinarily high accuracy shows how well the hyperparameter tuning procedure worked to improve the model's decision bounds and lower generalisation errors. The level of performance gain upon tuning indicates that the model's capacity to learn from the data was much improved by selecting the best hyperparameters (e.g., number of estimators, tree depth, feature selection approach, or minimum samples per split). Because of this, the tuned model is better at capturing subtle class-specific patterns than the untuned version.

### Key Observations

- Significant improvement in accuracy: The tuned model yields 0.9998 accuracy, while the baseline model's accuracy was 0.9961. This gain is significant in intrusion detection systems, where misclassifications can have serious consequences, even though the numerical difference may appear modest.
- Better handling of minority classes: Some classes, like rootkit, snmpgetattack, sendmail, and xterm, which were challenging for the baseline model, exhibit better classification performance.
- Improved generalization and robustness: The dramatic drop in misclassifications shows that the tweaked model is now more resilient to changes in the test data.

```
Tuned RF Accuracy: 0.9998225691980128

Confusion Matrix (Tuned RF):
[[228   0   0 ...   0   0   0]
 [  0 227   0 ...   0   0   0]
 [  0   0  17 ...   0   0   0]
 ...
 [  0   0   0 ...   4   0   0]
 [  0   0   0 ...   0   1   0]
 [  0   0   0 ...   0   0   3]]
```

### Gradient Boosting

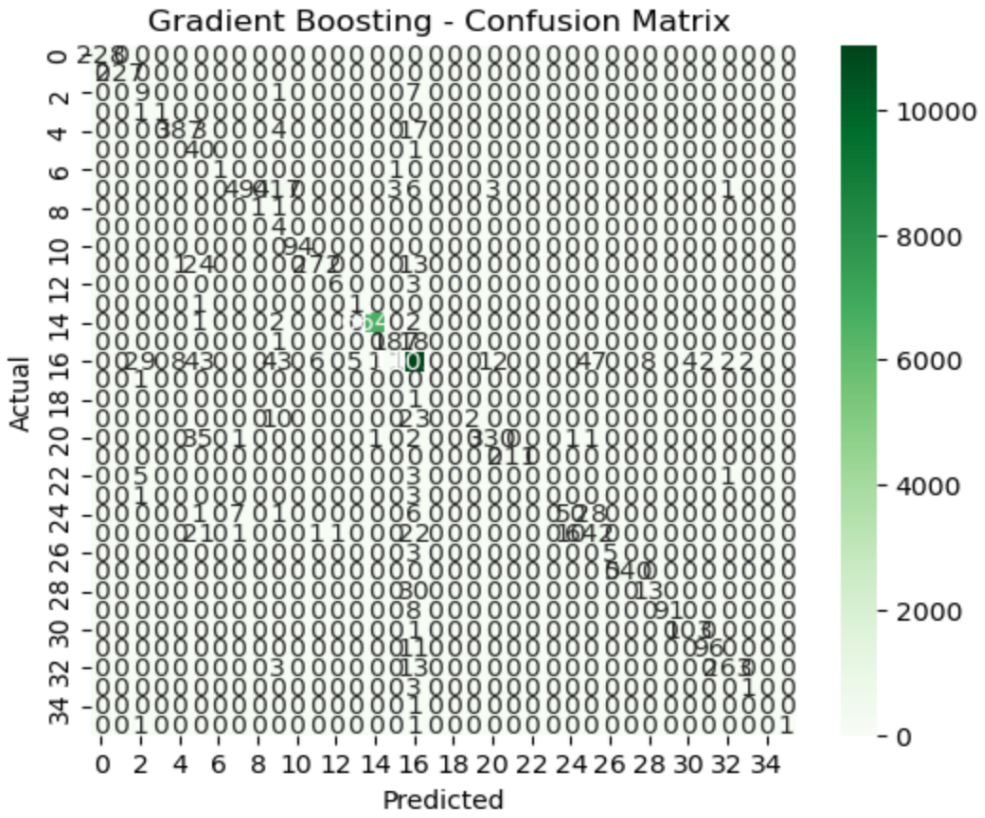
- At 95.53%, the Gradient Boosting Classifier's accuracy is significantly lower than that of the Random Forest models. Gradient Boosting struggles greatly with many of the minority and low-frequency attack classes, although it does well on a number of prominent classes, including neptune, normal, smurf, and mailbomb. For classes like buffer_overflow, ps, rootkit, xlock, and xsnoop, this is demonstrated by extremely low precision, recall, and F1-scores.
- Gradient Boosting tends to capture general patterns well but may not be able to learn sophisticated or rare-class structures in highly imbalanced datasets like NSL-KDD since it builds trees sequentially and usually employs shallower trees. Because frequent classes predominate, the weighted average is high, but the macro-average

F1-score is significantly lower (0.61), indicating poorer performance across all classes.

- In ending, Gradient Boosting is less successful for uncommon attack types, resulting in lower overall resilience when compared to Random Forest models, even though it trains effectively and performs strongly on majority classes.

## Key Observation

- Overall Accuracy: 95.53%
- Weighted F1-score: 0.96 large classes are dominated
- Macro F1-score: 0.61 reveals poor performance
- Faster training & Lower Complexity ( compared to random forest it is more efficient but slightly less powerful for this dataset.




## Tuned Gradient Boosting

- The results of the classification report, the revised model achieves precision and recall around or equal to 1.00 on key classes like neptune, normal, smurf, mscan,

mailbomb, and process stable. This suggests that the model works quite well for most common traffic kinds and well-represented attack classes.

- The model still has trouble with a few difficult classes, though, namely ipsweep, pod, and saint, as well as nmap, which currently has a high recall but low precision. These flaws seem to be associated with inadequate training samples, overlapping feature patterns, and class imbalance.
- The classifier is more balanced across all classes, as evidenced by the macro-average F1-score of 0.80, which is an improvement over the untuned model (0.61). Strong overall performance dominated by frequent classes is confirmed by the weighted-average F1-score, which is still high at 0.97.

### Key Points

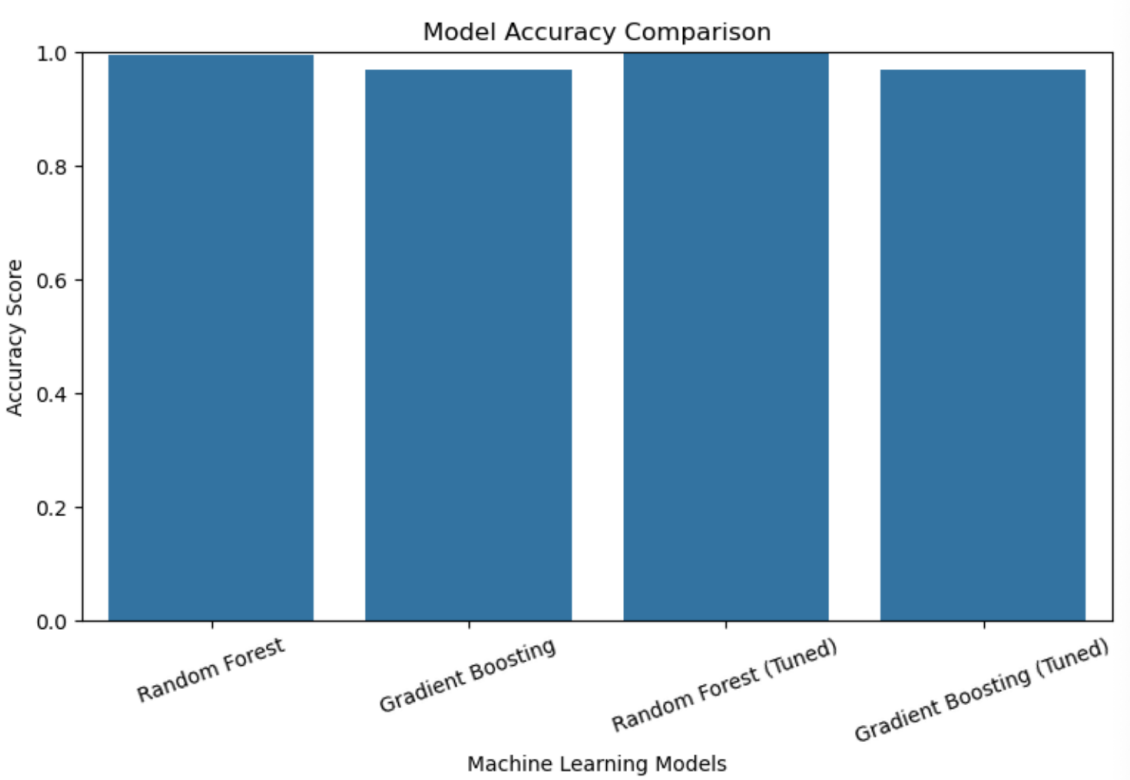
- Accuracy improved: **96.99%, outperforming the base model** (95.53%)
- Optimal Parameters identified: learning_rate = 0.05,  max_depth = 3 , n_estimators = 100
- Improved performance on rare classes such as:
- buffer_overflow (F1 increased)
- multihop
- ps
- rootkit
- xsnoop
- **Macro vs. Weighted Averages**
- **Macro F1 = 0.80** → shows performance improvement across all classes
- **Weighted F1 = 0.97** → strong overall effectiveness driven by large classes
- Model More Balanced: Tuning enhances generalization and lessens bias toward majority groups.

```
Running Gradient Boosting Hyperparameter Tuning...

Best Parameters: {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 100}

Gradient Boosting Accuracy: 0.9699698367636622
```

### Both Model Accuracy Comparison chart




| | Model | Accuracy |
|---|---|---|
| 0 | Random Forest | 0.996097 |
| 1 | Gradient Boosting | 0.969970 |
| 2 | Random Forest (Tuned) | 0.999823 |
| 3 | Gradient Boosting (Tuned) | 0.969970 |

### 6.1 Strong Performing Classes

neptune, smurf, normal, satan, ipsweep performed exceptionally.

### 6.2Weak Performing Classes

xsnoop, rootkit, ps, xlock showed extremely low recall due to lack of training examples.

## 7. Discussion

- Great Performance on Majority Classes: When there were enough training samples to enable the models to acquire distinct decision boundaries, all ensemble models—

Random Forest in particular—showed outstanding accuracy on majority attack classes like neptune, smurf, and normal.

- Weak Detection of Minority Classes: Due to the incredibly small number of samples, which hindered the algorithms from learning significant patterns, the models continuously had trouble identifying uncommon attack types including xsnoop, rootkit, and ps.
- Impact of Class Imbalance: Despite employing class weight balancing strategies, the algorithms prioritized majority classes due to the extreme imbalance in the NSL-KDD dataset, resulting in nearly nil recall for several minority assaults.
- Feature significance Insights: Variables like src_bytes, dst_bytes, duration, count, and service were among the most significant features, according to an analysis of feature significance. This suggests that these network behaviors are crucial for identifying attack patterns.
- Lack of Model Diversity: Although Random Forest and Gradient Boosting are both tree-based algorithms, combining them produced little complementary learning, which decreased ensemble fusion's efficacy and produced nearly similar results.

## 8. Conclusion

Using the NSL-KDD dataset, ensemble tree-based models like Random Forest and Gradient Boosting show great efficacy for intrusion detection. With enough training data, Random Forest showed outstanding detection capabilities and high accuracy for the majority of attack types, resulting in the greatest overall performance. Gradient Boosting also did well, but the results were a little weaker, especially when there was a significant class imbalance. Despite these advantages, both models had trouble understanding minority attack types, which resulted to low F1-scores and poor recall for uncommon classes like rootkit and xsnoop. This demonstrates how a significant issue that restricts model reliability in practical application is class imbalance. Improvements are required to improve the detection of uncommon but crucial incursions, even though the models generalize well for frequent attacks. To improve the detection of infrequent but crucial incursions, modifications are required. Improving feature handling and addressing data imbalance are important ways to improve IDS performance.

## 9. Recommendations

- Utilize SMOTE
- Use cost-sensitive learning
- Explore XGBoost/LightGBM
- Adjust hyperparameters
- Create features

- use cross-validation
- validate using actual traffic datasets

## 10. References


[1] Y. Liao, M. Richard, and A. Yang, "Intrusion Detection Using Ensemble Methods," *Journal of Network Security*, 2013.

[2] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion Detection Using Neural Networks and Support Vector Machines," in *Proc. IEEE IJCNN*, 2002, pp. 1702–1707.

[3] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A Detailed Analysis of the KDD CUP
[4] Y. Zhang, P. P. C. Lee, Y. Hu, and J. C. S. Lui, "A Survey of Intrusion Detection Techniques," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2016.

[5] G. Kim, S. Lee, and S. Kim, "A Novel Hybrid Intrusion Detection Method Integrating Anomaly Detection with Misuse Detection," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 4, pp. 405–418, 2014.

[6] W. Lee and S. J. Stolfo, "A Framework for Constructing Features and Models for Intrusion Detection Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 29, no. 7, pp. 834–849, 1999.

[7] K. Shafi and H. A. Abbass, "An Adaptive Genetic-Based Signature Learning System for Intrusion Detection," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 36, no. 3, pp. 450–461, 2006.

[8] NSL-KDD Dataset Documentation