APPROVAL OF HONORS PROGRAM SENIOR PROJECT

Candidate

Simiyon Fyuba Tamang

Project Title

The Intersection of Machine Learning and Cybersecurity:
Evaluation of Supervised Models for Intrusion Detection in Network Traffic

This Senior Project is approved as acceptable

Project Director

Dr. Jackie Lewis

Committee Member

Ms. Lucinda Boykin

Committee Member

Dr. Shane Thompson

Honors Program Co-Director

Dr. Bill Yankosky

Honors Program Co-Director

Dr. Fred Sanborn

April 28, 2026

The Intersection of Machine Learning and Cybersecurity: Evaluation of Supervised Models for Intrusion Detection in Network Traffic

Simiyon Fyuba Tamang

North Carolina Wesleyan University

Abstract

Network Intrusion Detection Systems (NIDS) are crucial in modern cybersecurity for identifying and preventing malicious network traffic. Traditional approaches, such as signature-based and anomaly-based detection, often struggle to detect and classify evolving cyber-attack patterns and may produce high false-positive rates. This experiment evaluates the effectiveness of supervised machine learning (ML) models for intrusion detection using the NSL-KDD dataset, demonstrating the strengths and limitations of the ML approach.

Four supervised models are implemented and compared under consistent experimental conditions – Logistic Regression (LR), K-Nearest Neighbors (KNN), Decision Tree (DT), and Random Forest. Data processing techniques such as encoding and feature selection are applied to improve performance, reduce dimensionality, and ensure fair model comparison. Metrics such as accuracy, precision, recall, and F1-score are used to evaluate the performance of models.

The results indicate that tree-based models achieve the highest performance compared to linear models. Overall, the results highlight the usability and scalability of models for intrusion detection but also highlight the feature engineering dependency of models.

Table of Contents

Introduction *5*

Problem Statement **6**

Objectives **6**

Research Question **7**

Literature Overview *7*

Fundamentals of Intrusion Detection **7**

Machine Learning models for intrusion detection **8**

Feature Engineering and Preprocessing **9**

Supervised Machine Learning Models in IDS **9**

- Logistic Regression (LR): 10
- Decision Trees (DT): 10
- K-Nearest Neighbors (KNN): 10

Methodology *11*

Dataset **11**

Data Preprocessing **14**

Recursive Feature Elimination (RFE) for feature selection **15**

Train-Test Data Split **16**

Implementing Supervised Models **17**

Model evaluation and visualization **18**

Findings *18*

Outcome of Preprocessing Dataset **18**

Model Performance Comparison **20**

Logistic Regression (LR) Results 20

K-Nearest Neighbors (KNN) Results 21

Decision Tree (DT) Results 22

Random Forest Classifier Results 23

F1-score Comparison 24

Important Features **25**

Discussion *26*

Role of Feature Engineering in Model Effectiveness **26**

Why Tree-Based Models perform Better Than Other Models **27**

Conclusion and Future Work *28*

Summary of Key Findings **28**

Limitations **29**

Future Research Directions **29**

Conclusion **30**

References *32*

Glossary *36*

Appendix A: Jupyter Notebook Summary *38*

Introduction

Cyber-attacks are unethical activities performed to steal, expose, corrupt, alter, damage, or destroy information through unauthorized access to the computer system or is an assault launched by attackers that use single or multiple computers to attack other computers and networks (Desai, Oza, & Naik, 2021).

The current technology revolution involves sensitive data and systems; it is important to take cyber-security seriously, and the need for Network Intrusion Detection Systems (NIDS) is very crucial. NIDS monitors internet traffic across the devices in a network. It acts as a defense line (Butun, Morgera, & Sankar, 2014); it identifies threats and protects the network from intruders and malicious attacks.

Most NIDS are signature-based, which means the system analyzes packets from a computer network and matches them against a set of signatures that trigger known intruder techniques, and it has limitations. For instance, it is challenged for signature-based NIDS to observe and analyze the changes in the network behavior and take actions to specific parts of the network. Modern networks consist of rapidly growing network protocols which generates wide range of data, which limits the accuracy of IDS, which needs to observe an online node, such as a host, server, or network, to build a baseline of normal activity and flag anything that is different from that defined baseline. The constantly changing nature of modern networks makes it hard for IDSs to define the baseline because it becomes outdated quickly and requires frequent re-analysis (Chkirbene & Hamila, 2025).

This promotes the research, development, and implementation of anomaly detection techniques capable of overcoming these limitations, such as machine learning algorithms. Machine learning models are trained on a large dataset with a variety of network data using

algorithms to improve the accuracy of detecting anomalies (Chkirbene et al., 2020). Machine learning techniques can be classified into two main categories:

1. Unsupervised - Identifies hidden structures and outliers in data without predefined labels (Chy, 2024).
2. Supervised - Learns for predicting the object class from pre-labeled (classified) objects (Almseidin et al., 2018).

Problem Statement

The Rapid advancements of Machine Learning algorithms for Network Intrusion Detection Systems still opens a debate about the effectiveness of common supervised models and the features of the sample data that affects the result of supervised models. This study aims to provide a comparative analysis of several supervised Machine Learning (ML) algorithms and the features of sample network data that influence the decision of these models.

Objectives

The main objectives of this study are as follows:

1. This study aims to review the existing literature on Network Intrusion Detection Systems (NIDS) and the implementation of machine learning algorithms.
2. To train several supervised ML models on the KDD-Cup 1999 dataset and evaluate their performance.
3. To do the comparative analysis of models in terms of accuracy, precision, recall, and F1 score.

4. To identify features of network traffic data sample that influences models the most.

Research Question

In this research, the following questions will be addressed:

1. How do various supervised ML models compare in terms of performance in detecting anomaly network traffic?
2. How is the feature engineering related to the effectiveness of models?
3. What are the practical strengths and limitations of each approach in intrusion detection?

Literature Overview

Fundamentals of Intrusion Detection

The first formal model for intrusion detection was introduced by D.E. Denning in 1985. According to Denning, Intrusion Detection Systems (IDS) play an important role in protecting computer networks by identifying unauthorized, malicious, or anomalous activities. Denning also established IDS as systems that can monitor audit records to identify system patterns and any abnormal activities in the system (Denning, 1985). This groundbreaking contribution by Denning categorized intrusion detection into two different categories based on their function – Signature-based and Anomaly detection.

Signature-based detection system (SIDS) identifies attacks by comparing incoming traffic against known threat patterns or rules, which makes it effective for known attacks but limits it to detect new threats, which are also called zero-day attacks. Techniques used by attackers are always evolving, which forces the system to be updated constantly with signature updates to remain effective. Whereas Anomaly-based detection system (AIDS) recognizes differences from

normal, typical network behavior to detect threats, typically with a higher false-positive rate (Bombay IV, 2025).

Hybrid IDS has emerged to overcome these limitations. Such IDS architecture combines both SIDS and AIDS, and these systems integrate rule-based techniques with statistical or learning-based detection methods. These hybrid architectures have formed a steppingstone for modern machine learning (ML) models and deep-learning (DP) models (Le et al., 2024).

Machine Learning models for intrusion detection

Machine Learning (ML) techniques have gained significant popularity in IDS development due to their ability to learn from labeled attack data and generalize to previously unseen cyber-threats. ML models can be grouped into three main categories: supervised, unsupervised, and ensemble methods (Bombay IV, 2025).

According to Belcie and Stryker (2025) from IBM, Supervised learning is a machine learning technique that uses labeled data sets to train artificial intelligence (AI) models to identify the underlying patterns and relationships. Supervised ML models give the best results in structured environments and offer interpretable decision-making.

Unsupervised ML models attempt to detect anomalies without labeled data. It is useful for exploratory analysis, but models struggle to differentiate subtle or evolving patterns, which often leads to unstable results under noisy traffic conditions (Yuan et al., 2024).

Ensemble Learning is the combination of predictions of multiple models with improved detection accuracy. Ensemble models such as bagging or stacking have demonstrated superior performance in comparison to IDS tasks using models standalone (Ganaie et al., 2022).

Feature Engineering and Preprocessing

ML models built using different algorithms require feature engineering and are consistent in terms of performance even when the distribution of traffic changes. Many studies have addressed these limitations by integrating Recursive Feature Elimination (RFE) and SHAP-based explanations to reduce redundancy and enhance interpretability (Mercha et al., 2023). In this thesis, RFE was used for feature selection before training the best-performing models.

Data preprocessing is as effective as feature selection for high-performance Network Intrusion Detection Systems, since it ensures that data from raw network traffic is transformed into a structured format suitable for machine learning models (Bombay IV, 2025). Data cleaning is the first and most important step of data processing, which involves removing duplicate records, handling missing values, and features (columns) that might cause data leakage (Lundqvist et al., 2025). In this thesis, scaling was used for the Logistic Regression model (Gradient-Based) and K-Nearest Neighbor model (Distance-Based) to avoid the bias of algorithms toward attributes with larger numeric ranges.

Supervised Machine Learning Models in IDS

The supervised Machine Learning models are trained using labeled data to create a trained model (Chkirbene & Hamila, 2025). The primary advantage of supervised learning models in IDS is their ability to learn the patterns and decision boundaries between normal and malicious traffic. Supervised model gets the leverage of direct feedback during training, which allows them to optimize classification performance with respect to known attack patterns, whereas unsupervised models must infer normal behavior without labels Buczak and Guven, 2016). The effectiveness of these models is evaluated through metrics such as Accuracy, Precision, Recall, and the F1-score, although their performance is often significantly different

depending on the specific anomaly category and the structure of the dataset (Bibers et al., 2024). Some of the supervised models that are the foundation of research on supervised IDS:

- *Logistic Regression (LR):* Logistic Regression is a supervised classification algorithm that approximates the likelihood that a piece of data will be a member of a specific class by using the logistic (sigmoid) function. It is extremely popular in intrusion detection because it is interpretable and effective at binary classification (Segun Omotayo et al., 2025). A comprehensive study by Graham and Lloris (2025) on logistic regression highlights that LR offers several advantages for intrusion detection, it is robust and easy to understand the factor contributing to an intrusion while being less intensive compared to more complex algorithms, making it suitable for environment where resources are limited. However, LR has limitations; the model can be significantly affected by outliers in the training data and skewed predictions.
- *Decision Trees (DT):* The decision tree (DT) is a widely used machine learning algorithm according to the division method. The DT contains features and leaves, where leaves are the class label (Chkirbene & Hamila, 2025). The DT model makes decisions and their possible consequences in a tree-like structure, making them easy to interpret and visualize (Graham & Lloris, 2025). While DT can show impressive results in terms of detection accuracy and speed, it can overfit the training data, which leads to poor generalization in the face of new threats (Omarov et al., 2023)
- *K-Nearest Neighbors (KNN):* The K-Nearest Neighbor is a simple supervised ML model that uses the concept of feature similarity to classify a given sample of data (Ali et al., 2023). KNN determines the identity of a sample based on its neighbors and how far away it is from them, which allows the algorithm to effectively determine the class of a data

sample. Overfitting or incorrect categorization can take place depending on the value of parameter *K*.

Methodology

This study experiments with multiple supervised machine learning algorithms to analyze and compare models for intrusion detection. The methodology for this study is designed to train, test, and evaluate all models based on same dataset and under same conditions so that their performance can be compared fairly. The entire process for this study was done in a JupyterLab environment using Python, utilizing Python libraries for data science and machine learning, such as Pandas and Scikit-learn. The complete implementation of the methodology, including data preprocessing, model training, and evaluation results, is available in a public GitHub repository (Tamang, 2026).

The major steps for this research are as follows:

1. Data collection and exploration
2. Data preprocessing and feature encoding
3. Feature selection using Random Forest Classifier (RFC)
4. Train and test data split
5. Model training
6. Mode evaluation and comparison

Dataset

The dataset used for this experiment is the NSL-KDD intrusion detection dataset, which is a publicly available dataset in Kaggle. This dataset is a newer and refined version of the KDD

Cup 1999 dataset, which has been the most wildly used data set for the evaluation of anomaly detection methods. This data set is prepared by Stolfo et al. (2000) and is built based on the data captured in DARPA'98 IDS evaluation program. DARPA'98 is about 4 gigabytes of compressed raw (binary) tcpdump data of 7 weeks of network traffic, which can be processed into about 5 million connection records, each with about 100 bytes (Lippmann et al., 2000).

NSL-KDD dataset has reduced redundancy and better class balance, which minimizes tasks for data cleaning and handling missing data. These characteristics make the dataset favorable among researchers, as it allows more focus on analysis rather than data preparation. The dataset contains network traffic records, where each row represents a single network connection, and each column represents different features of that connection. In total, the dataset contains 25,192 rows with 41 feature columns and one class label. The class variable represents whether a network connection is normal or an anomaly. After an exploratory data analysis of the dataset, the result showed that approximately 53% normal traffic and 47% anomalous traffic, it means the dataset is relatively balanced. This balance reduces the risk of bias while training models and makes performance metrics such as F1-score meaningful.

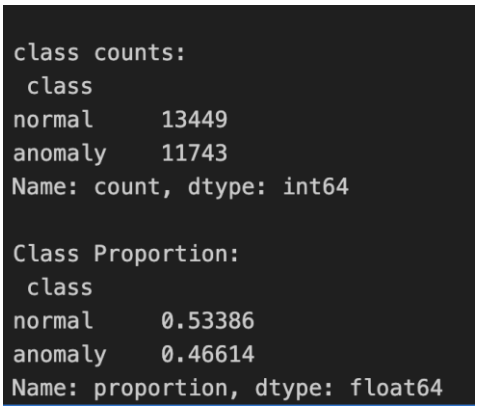


Figure 1: *Class Distribution of Network Traffic (Normal vs. Anomaly)*

Features of the dataset (NSL-KDD):

```
categorical_cols:
 Index(['protocol_type', 'service', 'flag', 'class'], dtype='object')

numerical_cols:
 Index(['duration', 'src_bytes', 'dst_bytes', 'land', 'wrong_fragment',
       'urgent', 'hot', 'num_failed_logins', 'logged_in', 'num_compromised',
       'root_shell', 'su_attempted', 'num_root', 'num_file_creations',
       'num_shells', 'num_access_files', 'num_outbound_cmds', 'is_host_login',
       'is_guest_login', 'count', 'srv_count', 'serror_rate',
       'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate',
       'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count',
       'dst_host_srv_count', 'dst_host_same_srv_rate',
       'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
       'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
       'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
       'dst_host_srv_rerror_rate'],
      dtype='object')
```

Figure 2: *Categorical and Numerical Features in the NSL-KDD Dataset*

The dataset contains three categories of attributes:

1. Numerical features: Columns such as *duration*, *src_bytes*, and *dst_bytes* contain information of network activity such as the duration of the connection, number of bytes transmitted, number of connection errors, count of connections to the same host and, so on. These columns are quantitative attributes.
2. Categorical features: Features of network connection, such as the type of protocol, service, and the status of the protocol connection is captured in columns such as *protocol_type*, *service*, and *flag.* These columns are categorical columns, as they represent qualitative attributes of a network connection.

3. Target column: Lastly, the *class* column indicates whether the connection is normal or an intrusion.

Data Preprocessing

The machine learning algorithms cannot directly use categorical columns, it requires label encoding. In this case, two class variables are converted into numerical values, where 1 represents normal traffic, and the other represents intrusion. Target variable encoding allows the model to process the target variable because of the binary nature of the class label.

```
# Label-encode the target (class)
label_encoder = LabelEncoder()
df['class'] = label_encoder.fit_transform(df['class'])

# One-hot encode categorical feature columns
categorical_cols = ['protocol_type', 'service', 'flag']

encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
encoded_array = encoder.fit_transform(df[categorical_cols])

# Convert encoded array to DataFrame with proper column names
encoded_df = pd.DataFrame(
    encoded_array,
    columns=encoder.get_feature_names_out(categorical_cols),
    index=df.index
)

# Drop original categorical columns and add encoded ones
df = pd.concat([df.drop(columns=categorical_cols), encoded_df], axis=1)
```

Figure 3: *Data preprocessing using label encoding and one-hot encoding in Python*

Similarly, One-Hot Encoding is also applied to other categorical columns to prevent the model from assuming any numerical or ordinal relationship between categories. In one-hot encoding, the original categorical columns are removed and replaced with newly created binary

columns for each unique category in these features. This approach increases the number of columns from 41 to 119, which provides detailed network information for the models.

After preprocessing, the dataset is divided into two components. The feature matrix (*x*) contains all input variables, while the target vector (*y*) contains the array of encoded class attributes. This is a major step required for supervised learning, where models learn patterns in the input features to predict the target variable.

Recursive Feature Elimination (RFE) for feature selection

Data preprocessing increases the dimensionality of the dataset, and Recursive Feature Elimination (RFE) is performed to focus on the most informative attributes. RFE involves repeatedly training a model, ranking the best features, and removing the least important ones.

```
rfc = RandomForestClassifier (
    n_estimators = 100,
    random_state = 42,
    n_jobs = -1
)

rfe = RFE (estimator = rfc, n_features_to_select = 10)
rfe. fit(x,y)

feature_map = [(support, col) for support, col in zip(rfe.get_support(), x.columns)]

selected_features = [col for support, col in feature_map if support]

selected_features
```

Figure 4: *RFE-based feature selection using Random Forest*

In this study, the Random Forest Classifier (RFC) algorithm is implemented as the estimator for this process. The Recursive Feature Elimination (RFE) algorithm selects the 10 best features that contribute most to the performance of the ML model. RFE not only reduces the dimensionality of the dataset but also improves efficiency while maintaining the strong performance of predictive ML models.

```
['src_bytes',
 'dst_bytes',
 'count',
 'same_srv_rate',
 'diff_srv_rate',
 'dst_host_srv_count',
 'dst_host_same_srv_rate',
 'dst_host_same_src_port_rate',
 'protocol_type_icmp',
 'flag_SF']
```

Figure 5: *Selected Features Using Recursive Feature Elimination (RFE)*

Train-Test Data Split

```
x_train, x_test, y_train, y_test = train_test_split (
    x_selected, y,
    test_size = 0.3,
    random_state = 42,
    stratify = y
)

x_train.shape
```

Figure 6: *Stratified train-test split of the dataset using a 70:30 ratio*

The dataset with selected columns is split into a training and a testing dataset using a 70:30 ratio. Where 70% of the rows are used to train the models, while 30% is used for testing the model. The split ensures that the proportion of rows with both normal and anomalous traffic are consistent in both train and test datasets for unbiased evaluation of model performance.

Implementing Supervised Models

```
models = {
    'LogisticRegression': Pipeline([
        ('scaler', StandardScaler()),
        ('lr', LogisticRegression(max_iter=1000, random_state=42))
    ]),
    'KNN': Pipeline([
        ('scaler', StandardScaler()),
        ('knn', KNeighborsClassifier(n_neighbors=3))
    ]),
    'DecisionTreeClassifier': DecisionTreeClassifier(random_state=42),
    'RandomForestClassifier': RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
}
```

```
preds = {}
for name, model in models.items():
    model.fit(x_train, y_train)
    preds[name] = model.predict(x_test)
```

Figure 7: *Implementation of supervised machine learning models and prediction pipeline*

Logistic Regression (LR) is a baseline linear classifier in this study. It is combined with *StandardScaler* using a pipeline to standardize numerical features, ensuring equal contribution to the model by all the features.

K-Nearest Neighbors (KNN) model is trained to predict the class of a network connection based on the majority class of its nearest neighbors in feature space. KNN is a distance-based classifier, so it is also combined with *StandardScaler* using a pipeline to prevent the dominance of larger numeric ranges while calculating distance.

Decision Tree (DT) and Random Forest Classifier (RFC) models are trained without scaling because they are robust to outliers.

All four supervised models are trained using the same training data and tested on the same test dataset to ensure fair performance comparison.

Model evaluation and visualization

The metrics used for the evaluation of models include accuracy, precision, recall, F1-score, and confusion matrix. These metrics are visualized using user friendly graphics to provide detailed insights into how well each model performed to distinguish normal and anomalous traffic.

In addition, the RFC model provides feature importance scores that indicates how much each selected features contributes to the final prediction of models. These scores are visualized using a bar chart to highlight the most important features.

Findings

This section presents the experimental results achieved after implementing four supervised ML models on the NSL_KDD dataset. To ensure fair comparison, all the models were trained and tested under identical conditions using the same selected features and stratified 70:30 train-test split.

Outcome of Preprocessing Dataset

The NSL-KDD training dataset contained 25,192 network traffic records with 41 original features and one class label. The exploratory analysis showed the dataset contains 13,449 (53.38%) normal traffic records, 11,743 (46.61%) anomalous network traffic records, and there were no missing values across any feature columns. The balanced class distribution reduced bias toward either class, which allows metrics such as F1-score to be meaningful.

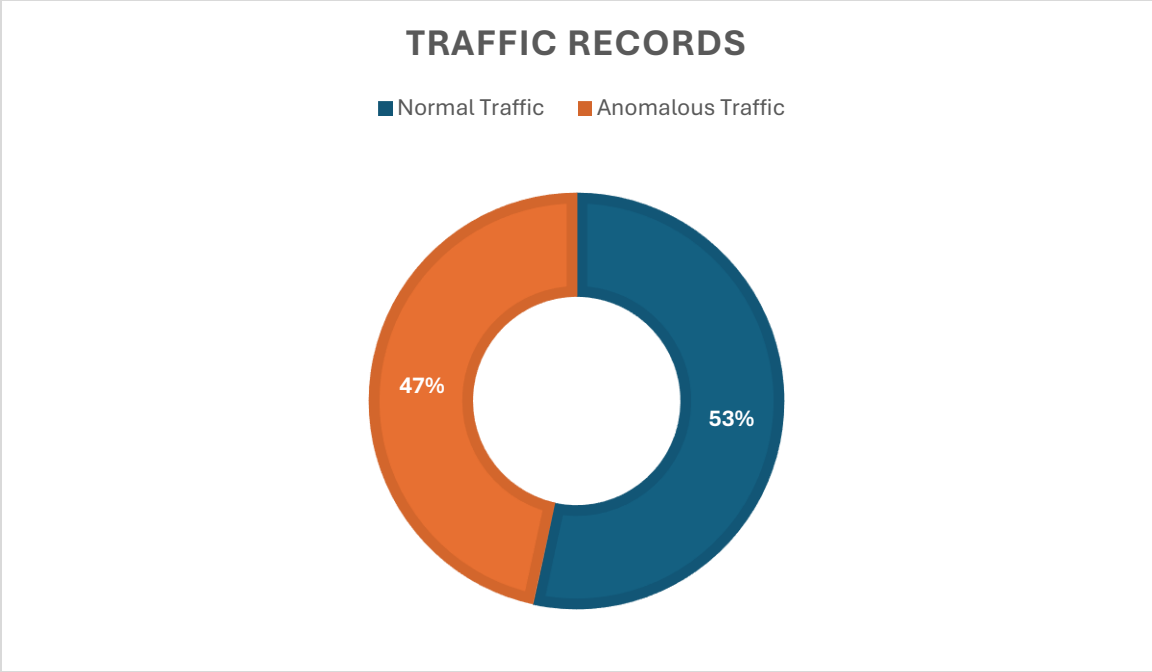


Figure 8: *Exploratory Analysis of NSL-KDD training dataset*

After applying Label Encoding to the target variable and One-Hot Encoding to categorical features, the number of feature columns increased from 41 to 119. For the sake of higher efficiency and reduced dimensionality, Recursive Feature Elimination (RFE) was performed using Random Forest, which selected the top 10 most important features:

Top 10 features
src_bytes
dst_bytes
count
same_srv_rate
diff_srv_rate
dst_host_srv_count
dst_host_same_srv_rate
dst_host_same_src_port_rate
protocol_type_icmp
flag_SF

These features include information regarding traffic volume characteristics and connection behavior patterns, which are important indicators of intrusion activity.

Model Performance Comparison

Four different supervised models were trained and evaluated. Performance of different models was assessed using accuracy, precision, recall, F1-score, confusion matrix, cross-validation, and ROC-AUC. The models used for this experiment were Logistic Regression (LR), K-Nearest Neighbors (KNN), Decision Tree (DT), and Random Forest Classifier (RFC). The detailed results and performance metrics for each model are presented below:

Logistic Regression (LR) Results

In this study, Logistic Regression was used as the baseline supervised learning model. The model was implemented on the test dataset using a pipeline with a StandardScaler to normalize the feature values before training because Logistic Regression is sensitive to the scale of numerical values.

Confusion matrix:

	Predicted Normal	**Predicted Anomaly**
Actual Normal	3280	243
Actual Anomaly	155	3880

Observing the confusion matrix, the model correctly classified 3280 normal traffic networks and 3880. However, some misclassifications also occurred. A total of 243 normal network traffic records were incorrectly classified as anomalies, which are considered false Positives. In the context of IDS, false positives are the normal traffic records that are falsely flagged as a security threat.

For anomaly, 155 anomalous connections were incorrectly classified as normal, which are false negatives. False negatives are important in IDS because they represent potential attacks or anomalous traffic that go undetected by the system.

Classification Report:

Class	Precision	Recall	F1-Score	Support
0 (Normal)	0.95	0.93	0.94	3523
1 (Anomaly)	0.94	0.96	0.95	4035

K-Nearest Neighbors (KNN) Results

Like LR model, the KNN model was also implemented as a distance-based classification algorithm combined with a StandardScaler within a pipeline to normalize feature values. Scaling is important for KNN because the algorithm relies on distance calculations to determine the nearest neighbors.

Confusion matrix:

	Predicted Normal	**Predicted Anomaly**
Actual Normal	3452	71
Actual Anomaly	80	3955

The KNN model correctly classified 3452 normal traffic and 3955 anomalous traffic. Only 71 normal connections were misclassified as anomalies, representing false positives. Whereas 80 anomalous traffic records were incorrectly predicted as normal. While false negatives still exist, the number is lower than in LR model indicating better detection capability.

Classification Report:

Class	Precision	Recall	F1-Score	Support
0 (Normal)	0.98	0.98	0.98	3523

1 (Anomaly)	0.98	0.98	0.98	4035

KNN achieved precision, recall, and F1-score values of approximately 0.98 for both network traffic classes, indicating very balanced performance. The precision and recall value of 0.98 indicates that the model nearly predicted all the traffic of both classes correctly. The overall accuracy of KNN was approximately 98%, which represents significantly better performance than the LR model.

Decision Tree (DT) Results

The Decision Tree (DT) model was implemented as a tree-based model that recursively splits the dataset based on feature values. Unlike LR and KNN, DT does not require feature scaling because it is not sensitive to the magnitude of feature values.

Confusion matrix:

	Predicted Normal	**Predicted Anomaly**
Actual Normal	3506	17
Actual Anomaly	30	4005

The DT model correctly classified 3506 normal traffics and 4005 anomalous traffics, showing very high accuracy. Only 17 normal records were incorrectly predicted as anomalies, representing false positives. This means the model rarely flagged normal traffic as anomalous. However, 30 anomalous records were incorrectly classified as normal, representing false negatives.

Classification Report:

Class	Precision	Recall	F1-Score	Support
0 (Normal)	0.99	1.00	0.99	3523
1 (Anomaly)	1.00	0.99	0.99	4035

The DT model achieved precision values close to 1.00, which means that almost every prediction made by the model was correct. A very strong balance between precision and recall is demonstrated by the F1-score of 0.99 for both classes.

Overall, DT delivered very strong results with an approximately 99% accuracy rate. DT clearly outperformed LR and KNN. This performance difference shows how well tree-based models capture complex nonlinear relationships between features in network traffic data.

Random Forest Classifier Results

The Random Forest Classifier was the most advanced model used in this study. Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the class that is the mode of the classes (classification) of the individual trees. This technique mitigates overfitting and enhances predictive accuracy (Graham & Lloris, 2025). The confusion matrix and classification report for the Random Forest model are shown below.

Confusion matrix:

	Predicted Normal	**Predicted Anomaly**
Actual Normal	3502	21
Actual Anomaly	13	4022

Classification Report:

Class	Precision	Recall	F1-Score	Support
0 (Normal)	1.00	0.99	1.00	3532
1 (Anomaly)	0.99	1.00	1.00	4035

The Random Forest model correctly classified 3502 normal and 4022 anomalous traffic records, it's very high classification accuracy. The model had 21 false positives and 13 false negatives. High precision and recall values close to 1.00, resulting in F1-scores of approximately 1.00 for both classes, show the strong model performance.

F1-score Comparison

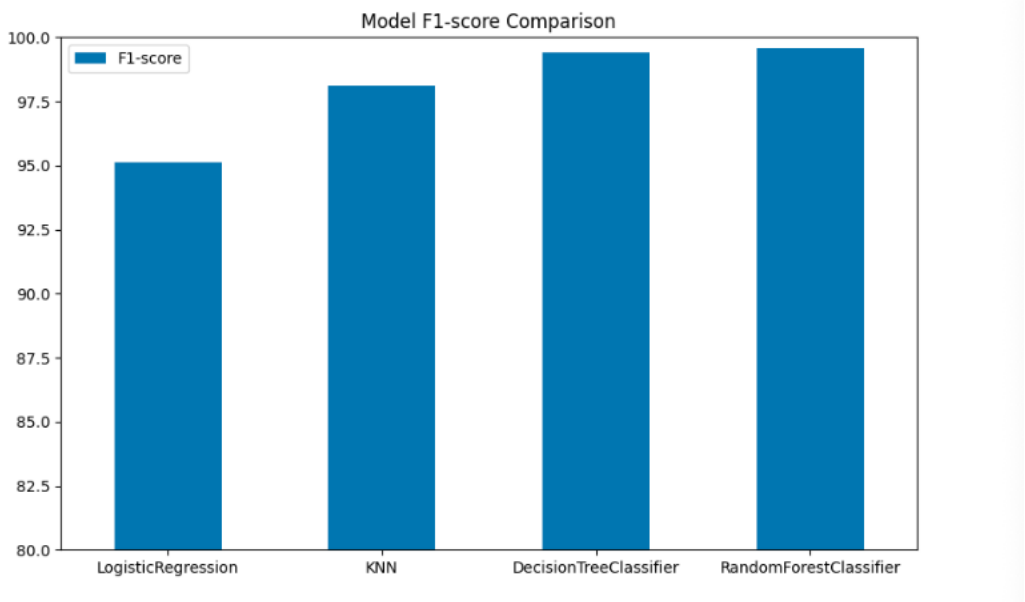


Figure 9: *Bar Chart of Model F1-score Comparison*

F1-score is an important metric in intrusion detection systems because it considers both false positives and false negatives. It makes the model observation informative since the model's strength to identify different class is highlighted by the metrics. Observing the chart, all models

achieved strong performance with an F1-score above 0.95. However, there are differences that exist among these models.

Logistic Regression model with the lowest score of 0.95 indicates the limitations of linear classifiers when handling complex patterns in network traffic data. Similarly, the distance-based model KNN model shows improved performance over LR, achieving an F1-score of 0.98, highlighting that the distance-based methods can capture localized relationships between network traffic features more effectively than linear models.

On the other hand, tree-based models outperformed both models mentioned above, achieving an F1-score of 0.99-1.00. These results suggest that tree-based models are capable of modeling non-linear relationships and complex feature interactions, which makes them well-suited for classifying network traffic.

Overall, the F1-score comparison demonstrates that tree-based models outperform linear and distance-based models for intrusion detection tasks, with Random Forest providing the most accurate and balanced performance.

Important Features

```
rf_importances = models['RandomForestClassifier'].feature_importances_
feat_imp = pd.Series(rf_importances, index=selected_features).sort_values(ascending=True)
```

```
plt.figure(figsize=(8,6))
feat_imp.plot(kind='barh')
plt.title("Random Forest Feature Importance")
plt.xlabel("Importance")
plt.show()
```

Figure 10: *Random Forest feature importance plot*

For a better understanding of the contribution of attributes to the models, feature importance analysis was performed using a Random Forest Classifier. The model assigned an

importance score to each selected feature based on how much it contributes. These importance values were visualized using a horizontal bar chart, where the x-axis represents the importance score, and the y-axis lists the selected features.

The results indicate that features such as *src_bytes, dst_bytes*, *count*, and other connection-related attributes have higher importance compared to others. This shows that traffic volume and connection behavior play a significant role in distinguishing between normal and anomalous network activity.

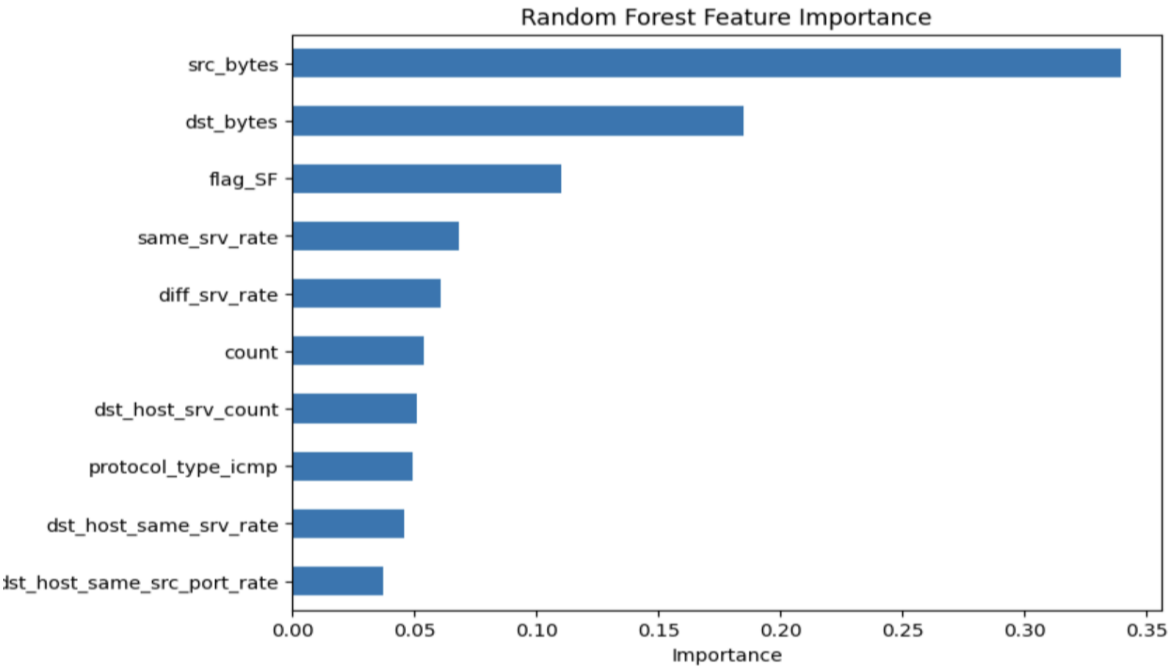


Figure 11: *Feature importance scores computed by Random Forest*

Discussion

Role of Feature Engineering in Model Effectiveness

Feature Engineering played an important role in improving the efficiency and effectiveness of the machine learning models used in this study. The original dataset consists of both numerical and categorical features, and most ML algorithms cannot directly process

categorical attributes. One-hot encoding was applied to convert these features into numerical form. One-hot encoding encodes categorical features as a one-hot numeric array (Li et al., 2018).

The transformation increased the total number of features from 41 to 119, which increased the dimensionality of the dataset, leading to increased computational complexity. To address this issue, Recursive Feature Elimination (RFE) was used to select the most informative features. Using Random Forest as the estimator, the dataset was reduced to the ten most important features. The selected features were mainly related to connection frequency, traffic volume, and behavior, which are commonly associated with abnormal network activity. This demonstrates that proper feature selection can significantly improve model efficiency while maintaining strong predictive performance.

Why Tree-Based Models perform Better Than Other Models

The result of the performance comparison between models indicates that tree models, particularly Decision Tree and Random Forest, achieved better performance results compared to linear and distance-based models. This performance difference can be explained by fundamental differences in how different models learn patterns from data. While linear models operate on the assumption of a linear relationship between features and the target class, which often fails to capture the non-linear correlations present in modern attack patterns, tree-based models utilize a hierarchical structure to recursively split data. This allows the tree-based model to learn “if-then” rules that better represent patterns in network traffic (Bibers et al., 2024).

Additionally, after one-hot-encoding, the dimensionality of the dataset increased from 41 to 119 features. High-dimensional data can negatively impact the performance of linear models, whereas tree-based models can handle such features. The ability to identify important features by tree-based models can also be observed in this study - the Random Forest further improves

performance through ensemble learning by combining multiple decision trees. This reduces the impact of noise and improves the stability of the model, making the predictions more robust.

The results clearly support these observations. LR model achieved an F1-score of 0.95, while Decision Tree and Random Forest achieved 0.99 and 1.00, respectively. Overall, these results confirm the superior performance of tree-based models in intrusion detection systems.

Conclusion and Future Work

Summary of Key Findings

This study presents a comparative analysis of multiple supervised machine learning models for network intrusion detection using the NSL-KDD dataset. The results highlight the strengths of different models and the importance of feature engineering on the performance of models:

- Model Performance: Tree-based models outperformed linear models. Random Forest achieved the highest performance with an F1-score close to **1.00**, followed by Decision Tree (**0.99**), K-Nearest Neighbors (**0.98**), and Logistic Regression (**0.95**).
- Model Capability: The LR model was a strong baseline model, but it was limited in capturing nonlinear relationships in network traffic data. Whereas KNN improved performance by using the distance-based algorithm, tree-based models effectively captured complex interactions between the feature columns.
- Impact of Feature Engineering: One-hot encoding increased the dimension of the dataset by increasing the number of feature columns. It was effectively filtered and reduced using the Recursive Feature Elimination (RFE) algorithm. Selecting the best features helped maintain high model performance, which demonstrates the importance of a smaller set of relevant features for better efficiency.

- Feature Importance: The most influential features were related to traffic volume and connection behavior, such as *src_bytes, dst_bytes*, and connection-based metrics. These features played a key role in distinguishing between normal and anomalous traffic.

Limitations

While the results and metrics of this study demonstrate strong performance of supervised ML models for NIDS, there are several limitations that should be considered:

- Limitation of Dataset: The dataset used for this experiment is the NSL-KDD dataset, which is a benchmark dataset commonly used in research. However, it does not fully represent modern network environments or evolving cyberattack patterns, which could affect its application in the real world.
- No Real-Time detection: This experiment is solely focused on evaluating models using offline data but in real-time detection, there are other important performance metrics for NIDS such as latency, scalability, and response time of models, which were not measured.
- Dependent on Feature Engineering: The performance of models was influenced by preprocessing and feature selection techniques. It is possible that different feature engineering approaches may produce different results. The importance of feature engineering in this experiment highlights the dependency of models on data preparation methods.

Future Research Directions

Considering the findings and limitations of this study, there are several opportunities for future research:

- **Identifying attack type:** Future studies should extend this work by developing models that not only classify whether the traffic is normal or not but also classify the different types of attacks.
- **Use of Real-World Datasets:** More recent and real-world network traffic datasets should be used to better train and evaluate models to better demonstrate their effectiveness in modern cybersecurity environments.
- **Test with Real-Time Application:** For practical end-user use, the trained models can be integrated into web-based applications or web services. A preprocessing pipeline should be made for user-uploaded network traffic data to ensure it replicates the same steps used during model training, including encoding, feature selection, and scaling where required.

Conclusion

This experiment highlights the effectiveness of supervised machine learning models, particularly tree-based models, in improving network intrusion detection systems. While all the models demonstrated strong performance, tree-based algorithms outperformed linear approaches due to their ability to capture complex patterns and interactions with network traffic data.

The importance of feature engineering while training supervised models is also emphasized by the results. Recursive Feature Elimination selected the 10 best features, which allowed the models to focus on the most informative features while maintaining high performance. Overall, the findings suggest that ML based approaches can significantly improve the ability to detect anomalous network activity. Specifically, tree-based models offer a strong combination of accuracy and stability, which makes them well suited for intrusion detection systems. Furthermore, integrating these models into real-time systems and evaluating their

performance on modern dataset will contribute highly towards developing reliable cybersecurity solutions.

References

Ali, T. E., Chong, Y. W., & Manickam, S. (2023). Machine Learning Techniques to Detect a DDoS Attack in SDN: A Systematic Review. *Applied Sciences 2023, Vol. 13, Page 3183*, *13*(5), 3183. https://doi.org/10.3390/APP13053183

Almseidin, M., Alzubi, M., Kovacs, S., & Alkasassbeh, M. (n.d.). *Evaluation of Machine Learning Algorithms for Intrusion Detection System*.

Belcie, I., & Stryker, C. (2025, November 17). Supervised Learning. *IBM*. https://www.ibm.com/think/topics/supervised-learning#1509394340

Bibers, I., Arreche, O., & Abdallah, M. (2024). *A Comprehensive Comparative Study of Individual ML Models and Ensemble Strategies for Network Intrusion Detection Systems*. https://arxiv.org/pdf/2410.15597

Bombay IV, L. T. (2025). *A comparative analysis of machine learning and deep learning approaches for network intrusion detection* (master's thesis, University of Idaho). https://ieeexplore.ieee.org/document/9573685

Buczak, A. L., & Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys and Tutorials*, *18*(2), 1153–1176. https://doi.org/10.1109/COMST.2015.2494502

Butun, I., Morgera, S. D., & Sankar, R. (2014). *A survey of intrusion detection systems in wireless sensor networks*. IEEE Communications Surveys & Tutorials, 16(1), 266–282. https://doi.org/10.1109/SURV.2013.050113.00191

Chkirbene, Z., Erbad, A., Hamila, R., Mohamed, A., Guizani, M., & Hamdi, M. (2020). TIDCS: A Dynamic Intrusion Detection and Classification System Based Feature Selection. *IEEE Access*, *8*, 95864–95877. https://doi.org/10.1109/ACCESS.2020.2994931

Chkirbene, Z., & Hamila, R. (2025). Supervised machine learning for intrusion detection: A survey and performance evaluation. *Innovation and Technological Advances for Sustainability - Proceedings of the International Conference on Innovation and Technological Advances for Sustainability, ITAS 2023*, 402–408. https://doi.org/10.1201/9781003496724-39

Denning, D., & Neumann, P. G. (1985). Requirements and model for IDES-a real-time intrusion-detection expert system (Vol. 8). Menlo Park: SRI International. https://faculty.nps.edu/dedennin/publications/IDESReportSRI1985.pdf

Desai, V. P., Oza, D., & Naik, D. (2021). Data mining approach for cyber security. *International Journal of Computer Applications Technology and Research*, *10*(01), 035–041. https://doi.org/10.7753/ijcatr1001.1007

Ganaie, M. A., Hu, M., Malik, A. K., Tanveer, M., & Suganthan, P. N. (2022). Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, *115*. https://doi.org/10.1016/j.engappai.2022.105151

Kamrul, M., & Chy, H. (2024). *Proactive fraud defense: Machine learning's evolving role in protecting against online fraud*. https://doi.org/10.30574/wjarr.2024.23.3.2811

Le, T. T. H., Shin, Y., Kim, M., & Kim, H. (2024). Towards unbalanced multiclass intrusion detection with hybrid sampling methods and ensemble classification. *Applied Soft Computing*, *157*, 111517. https://doi.org/10.1016/J.ASOC.2024.111517

Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyschogrod, D., Cunningham, R. K., & Zissman, M. A. (2000). Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation.

Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX), 2, 1012.

Lloris, J., & Lloris, J. (2025). *Comparative Study of Supervised Learning Algorithms for Intrusion Detection with a Focus on Logistic Regression*. https://doi.org/10.20944/PREPRINTS202505.0420.V1

Lundqvist, J., Hadzic, A., Kirkeluten, T. M., Pedersen, H., Holth, J., Johansson, M. H., & Halkjelsvik, M. P. N. (2025). Lightweight Machine Learning Models for Intrusion Detection on IoT Devices. *Norsk IKT-Konferanse for Forskning Og Utdanning*, *37*(3). https://doi.org/10.5324/JRXDJB92

Mercha, E. M., Chakir, E. M., & Erradi, M. (2023). Trans-IDS: A Transformer-Based Intrusion Detection System. *Proceedings of the International Conference on Security and Cryptography*, *1*, 402–409. https://doi.org/10.5220/0012085800003555

Omarov, B., Abdinurova, N., & Abdulkhamidov, Z. (n.d.). A Novel Framework for Detecting Network Intrusions Based on Machine Learning Methods. *IJACSA) International Journal of Advanced Computer Science and Applications*, *14*(7), 2023. Retrieved January 27, 2026, from www.ijacsa.thesai.org

Segun Omotayo, O., Abiodun, O., Olamatanmi Mebawondu, J., & Christianah Akinduyite, O. (2023). An ensemble models to detect intrusion using CIC-IDS-2017 and CIC-IDS-2018 as benchmarks. *ISA Journal of Multidisciplinary (ISAJM)*, *2*(5), 14–23. https://doi.org/10.5281/zenodo.17073568

Song, T., & He, X. (2021). *A concise method for feature selection via normalized frequencies*.

Stolfo, S. J., Fan, W., Lee, W., Prodromidis, A., & Chan, P. K. (2000). Cost-based modeling for fraud and intrusion detection: Results from the JAM project. *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX)*, 2, 1130.

Tamang, S (2026). *evaluation-of-supervised-ml-models*. (2026). GitHub. https://github.com/Simiyon-Tamang/evaluation-of-supervised-ml-models

Yuan, X., Han, S., Huang, W., Ye, H., Kong, X., & Zhang, F. (2024). A simple framework to enhance the adversarial robustness of deep learning-based intrusion detection system. *Computers and Security*, *137*, 1–19. https://doi.org/10.1016/j.cose.2023.103644

Glossary

Artificial Intelligence (AI) - The simulation of human intelligence in machines that enables systems to learn, reason, and make decisions.

Anomaly Detection – A data analysis process of identifying unusual patterns, items, events, or observation that deviate significantly from a dataset's normal pattern.

Accuracy – A performance metric that measures the proportion of correctly classified instances among all predictions made by a model.

Confusion Matrix – A table used to evaluate classification performance by displaying true positives, true negatives, false positives, and false negatives.

Cyber-Attack – An unauthorized attempt to access, damage, or disrupt computer systems or networks.

Feature Engineering - The process of transforming raw data into meaningful features to improve model performance.

F1-Score - A metric that combines precision and recall evaluating a model's overall classification performance.

False Positive (FP) - A normal instance incorrectly classified as an anomaly.

False Negative (FN) - An anomalous instance incorrectly classified as normal.

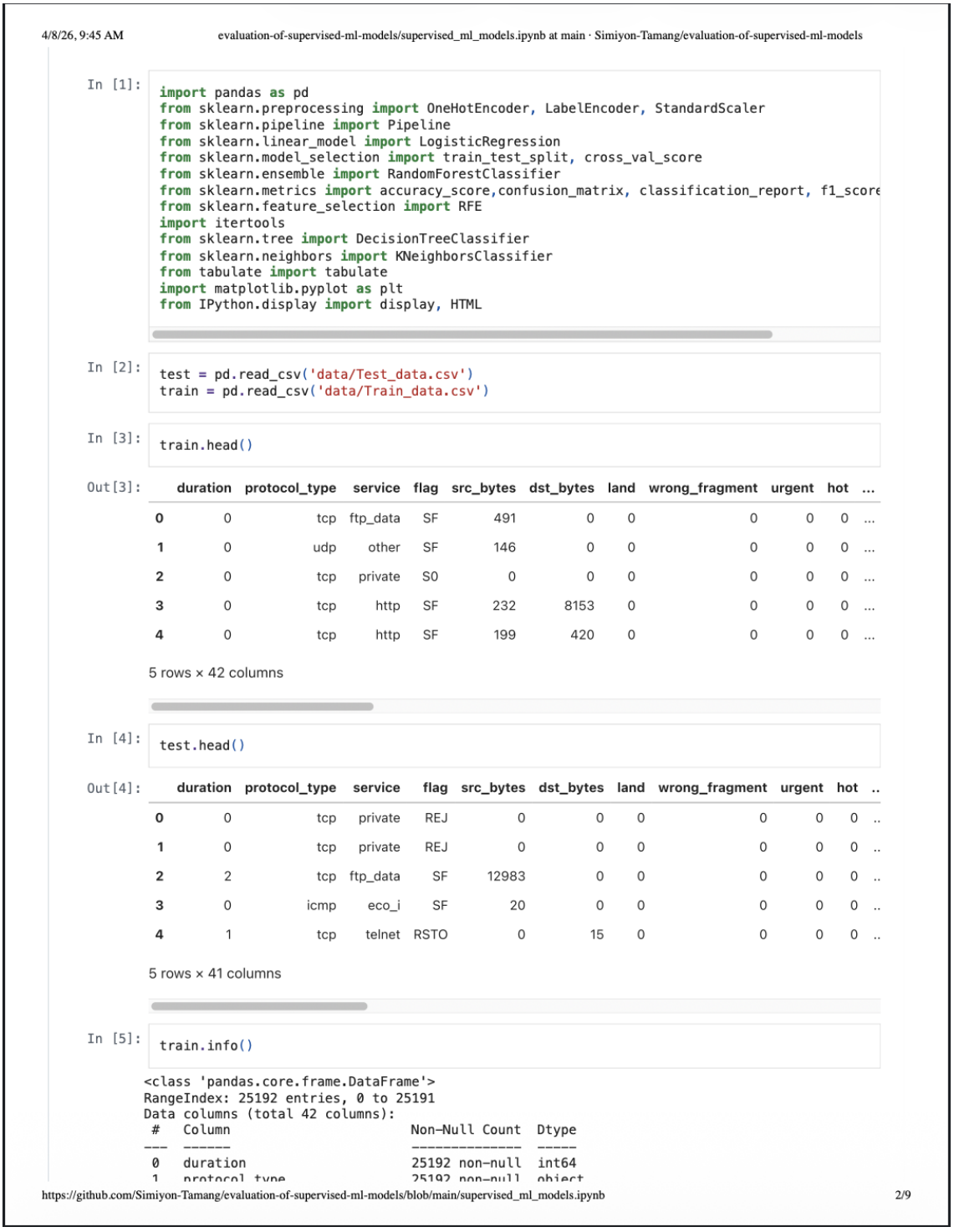
Intrusion Detection System (IDS) - A system that monitors network or system activity to detect malicious behavior or policy violations.

Machine Learning (ML) - A subset of AI that allows systems to learn patterns from data and make predictions without explicit programming.

Network Intrusion - Unauthorized access or malicious activity within a computer network aimed at compromising data or system integrity.

Appendix A: Jupyter Notebook Summary

This appendix summarizes the Python implementation used in this study. The full notebook is available on GitHub (Tamang, 2026).



4/8/26, 9:45 AM evaluation-of-supervised-ml-models/supervised_ml_models.ipynb at main · Simiyon-Tamang/evaluation-of-supervised-ml-models

In [1]:
```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,confusion_matrix, classification_report, f1_score
from sklearn.feature_selection import RFE
import itertools
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from tabulate import tabulate
import matplotlib.pyplot as plt
from IPython.display import display, HTML
```

In [2]:
```
test = pd.read_csv('data/Test_data.csv')
train = pd.read_csv('data/Train_data.csv')
```

In [3]:
```
train.head()
```

Out[3]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...
1	0	udp	other	SF	146	0	0	0	0	0	...
2	0	tcp	private	S0	0	0	0	0	0	0	...
3	0	tcp	http	SF	232	8153	0	0	0	0	...
4	0	tcp	http	SF	199	420	0	0	0	0	...

5 rows × 42 columns

In [4]:
```
test.head()
```

Out[4]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	..
0	0	tcp	private	REJ	0	0	0	0	0	0	..
1	0	tcp	private	REJ	0	0	0	0	0	0	..
2	2	tcp	ftp_data	SF	12983	0	0	0	0	0	..
3	0	icmp	eco_i	SF	20	0	0	0	0	0	..
4	1	tcp	telnet	RSTO	0	15	0	0	0	0	..

5 rows × 41 columns

In [5]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25192 entries, 0 to 25191
Data columns (total 42 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   duration                     25192 non-null  int64
 1   protocol_type                25192 non-null  object
```

https://github.com/Simiyon-Tamang/evaluation-of-supervised-ml-models/blob/main/supervised_ml_models.ipynb 2/9

```
 2   service                      25192 non-null  object
 3   flag                         25192 non-null  object
 4   src_bytes                    25192 non-null  int64
 5   dst_bytes                    25192 non-null  int64
 6   land                         25192 non-null  int64
 7   wrong_fragment               25192 non-null  int64
 8   urgent                       25192 non-null  int64
 9   hot                          25192 non-null  int64
 10  num_failed_logins            25192 non-null  int64
 11  logged_in                    25192 non-null  int64
 12  num_compromised              25192 non-null  int64
 13  root_shell                   25192 non-null  int64
 14  su_attempted                 25192 non-null  int64
 15  num_root                     25192 non-null  int64
 16  num_file_creations           25192 non-null  int64
 17  num_shells                   25192 non-null  int64
 18  num_access_files             25192 non-null  int64
 19  num_outbound_cmds            25192 non-null  int64
 20  is_host_login                25192 non-null  int64
 21  is_guest_login               25192 non-null  int64
 22  count                        25192 non-null  int64
 23  srv_count                    25192 non-null  int64
 24  serror_rate                  25192 non-null  float64
 25  srv_serror_rate              25192 non-null  float64
 26  rerror_rate                  25192 non-null  float64
 27  srv_rerror_rate              25192 non-null  float64
 28  same_srv_rate                25192 non-null  float64
 29  diff_srv_rate                25192 non-null  float64
 30  srv_diff_host_rate           25192 non-null  float64
 31  dst_host_count               25192 non-null  int64
 32  dst_host_srv_count           25192 non-null  int64
 33  dst_host_same_srv_rate       25192 non-null  float64
 34  dst_host_diff_srv_rate       25192 non-null  float64
 35  dst_host_same_src_port_rate  25192 non-null  float64
 36  dst_host_srv_diff_host_rate  25192 non-null  float64
 37  dst_host_serror_rate         25192 non-null  float64
 38  dst_host_srv_serror_rate     25192 non-null  float64
 39  dst_host_rerror_rate         25192 non-null  float64
 40  dst_host_srv_rerror_rate     25192 non-null  float64
 41  class                        25192 non-null  object
dtypes: float64(15), int64(23), object(4)
memory usage: 8.1+ MB
```

In [6]:
```
train.describe()
```

Out[6]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	
count	25192.000000	2.519200e+04	2.519200e+04	25192.000000	25192.000000	25192.00000	25192.
mean	305.054104	2.433063e+04	3.491847e+03	0.000079	0.023738	0.00004	0
std	2686.555640	2.410805e+06	8.883072e+04	0.008910	0.260221	0.00630	2
min	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00000	0.
25%	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00000	0.
50%	0.000000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.00000	0.
75%	0.000000	2.790000e+02	5.302500e+02	0.000000	0.000000	0.00000	0.
max	42862.000000	3.817091e+08	5.151385e+06	1.000000	3.000000	1.00000	77.

8 rows × 38 columns

In [7]:
```
print ("\nclass counts:\n",train['class'].value_counts())
print("\nClass Proportion:\n", train['class'].value_counts(normalize = True))
```

```
class counts:
 class
normal     13449
anomaly    11743
Name: count, dtype: int64

Class Proportion:
 class
normal     0.53386
anomaly    0.46614
Name: proportion, dtype: float64
```

In [8]:

```
print("Missing values per column:\n", train.isnull().sum())
display(train.head())
```

```
Missing values per column:
 duration                      0
protocol_type                  0
service                        0
flag                           0
src_bytes                      0
dst_bytes                      0
land                           0
wrong_fragment                 0
urgent                         0
hot                            0
num_failed_logins              0
logged_in                      0
num_compromised                0
root_shell                     0
su_attempted                   0
num_root                       0
num_file_creations             0
num_shells                     0
num_access_files               0
num_outbound_cmds              0
is_host_login                  0
is_guest_login                 0
count                          0
srv_count                      0
serror_rate                    0
srv_serror_rate                0
rerror_rate                    0
srv_rerror_rate                0
same_srv_rate                  0
diff_srv_rate                  0
srv_diff_host_rate             0
dst_host_count                 0
dst_host_srv_count             0
dst_host_same_srv_rate         0
dst_host_diff_srv_rate         0
dst_host_same_src_port_rate    0
dst_host_srv_diff_host_rate    0
dst_host_serror_rate           0
dst_host_srv_serror_rate       0
dst_host_rerror_rate           0
dst_host_srv_rerror_rate       0
class                          0
dtype: int64
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	c
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	
1	0	udp	other	SF	146	0	0	0	0	0	...	
2	0	tcp	private	S0	0	0	0	0	0	0	...	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	

4	0	tcp	http	SF	199	420	0	0	0	0	0	...

5 rows × 42 columns

Identify categorical & numeric features

In [ ]:
```
categorical_cols = train.select_dtypes(include = ['object']).columns
numerical_cols = train.select_dtypes(exclude = ['object']).columns

print("categorical_cols:\n", categorical_cols)
print("\nnumerical_cols:\n", numerical_cols)
```

```
categorical_cols:
 Index(['protocol_type', 'service', 'flag', 'class'], dtype='object')

numerical_cols:
 Index(['duration', 'src_bytes', 'dst_bytes', 'land', 'wrong_fragment',
       'urgent', 'hot', 'num_failed_logins', 'logged_in', 'num_compromised',
       'root_shell', 'su_attempted', 'num_root', 'num_file_creations',
       'num_shells', 'num_access_files', 'num_outbound_cmds', 'is_host_login',
       'is_guest_login', 'count', 'srv_count', 'serror_rate',
       'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate',
       'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count',
       'dst_host_srv_count', 'dst_host_same_srv_rate',
       'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
       'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
       'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
       'dst_host_srv_rerror_rate'],
      dtype='object')
```

In [10]:
```
df = train.copy()

# Label-encode the target (class)
label_encoder = LabelEncoder()
df['class'] = label_encoder.fit_transform(df['class'])

# One-hot encode categorical feature columns
categorical_cols = ['protocol_type', 'service', 'flag']

encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
encoded_array = encoder.fit_transform(df[categorical_cols])

# Convert encoded array to DataFrame with proper column names
encoded_df = pd.DataFrame(
    encoded_array,
    columns=encoder.get_feature_names_out(categorical_cols),
    index=df.index
)

# Drop original categorical columns and add encoded ones
df = pd.concat([df.drop(columns=categorical_cols), encoded_df], axis=1)

df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25192 entries, 0 to 25191
Columns: 119 entries, duration to flag_SH
dtypes: float64(95), int64(24)
memory usage: 22.9 MB
```

Out[10]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	nu
0	0	491	0	0	0	0	0	0	0	
1	0	146	0	0	0	0	0	0	0	

2	0	0	0	0	0	0	0	0	0
3	0	232	8153	0	0	0	0	0	1
4	0	199	420	0	0	0	0	0	1

5 rows × 119 columns

In [11]:
```
x = df.drop(columns = ['class'])
y = df['class']
```

In [12]:
```
rfc = RandomForestClassifier (
    n_estimators = 100,
    random_state = 42,
    n_jobs = -1
)

rfe = RFE (estimator = rfc, n_features_to_select = 10)
rfe. fit(x,y)

feature_map = [(support, col) for support, col in zip(rfe.get_support(), x.columns)]

selected_features = [col for support, col in feature_map if support]

selected_features
```

Out[12]:
```
['src_bytes',
 'dst_bytes',
 'count',
 'same_srv_rate',
 'diff_srv_rate',
 'dst_host_srv_count',
 'dst_host_same_srv_rate',
 'dst_host_same_src_port_rate',
 'protocol_type_icmp',
 'flag_SF']
```

In [13]:
```
x_selected = x[selected_features]
```

In [14]:
```
x_train, x_test, y_train, y_test = train_test_split (
    x_selected, y,
    test_size = 0.3,
    random_state = 42,
    stratify = y
)

x_train.shape
```

Out[14]:
```
(17634, 10)
```

In [15]:
```
x_test.shape
```

Out[15]:
```
(7558, 10)
```

In [16]:
```
models = {
    'LogisticRegression': Pipeline([
        ('scaler', StandardScaler()),
        ('lr', LogisticRegression(max_iter=1000, random_state=42))
    ]),
```

```
    'KNN' : Pipeline([
        ('scaler', StandardScaler()),
        ('knn', KNeighborsClassifier(n_neighbors=3))
    ]),
    'DecisionTreeClassifier': DecisionTreeClassifier(random_state=42),
    'RandomForestClassifier': RandomForestClassifier(n_estimators=100, random_state=42, n_jo
}
```

In [17]:

```
preds = {}
for name, model in models.items():
    model.fit(x_train, y_train)
    preds[name] = model.predict(x_test)
```

In [18]:

```
for name in preds:
    display(HTML(f"<h3>{name}</h3>"))
    print("Confusion Matrix:\n", confusion_matrix(y_test, preds[name]))
    print("\nClassification Report:\n", classification_report(y_test, preds[name]))
```

LogisticRegression

```
Confusion Matrix:
 [[3280  243]
 [ 155 3880]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.93      0.94      3523
           1       0.94      0.96      0.95      4035

    accuracy                           0.95      7558
   macro avg       0.95      0.95      0.95      7558
weighted avg       0.95      0.95      0.95      7558
```

KNN

```
Confusion Matrix:
 [[3452   71]
 [  80 3955]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      3523
           1       0.98      0.98      0.98      4035

    accuracy                           0.98      7558
   macro avg       0.98      0.98      0.98      7558
weighted avg       0.98      0.98      0.98      7558
```

DecisionTreeClassifier

```
Confusion Matrix:
 [[3506   17]
 [  30 4005]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      3523
           1       1.00      0.99      0.99      4035

    accuracy                           0.99      7558
   macro avg       0.99      0.99      0.99      7558
weighted avg       0.99      0.99      0.99      7558
```

```
    'KNN' : Pipeline([
        ('scaler', StandardScaler()),
        ('knn', KNeighborsClassifier(n_neighbors=3))
    ]),
    'DecisionTreeClassifier': DecisionTreeClassifier(random_state=42),
    'RandomForestClassifier': RandomForestClassifier(n_estimators=100, random_state=42, n_jo
}
```

In [17]:

```
preds = {}
for name, model in models.items():
    model.fit(x_train, y_train)
    preds[name] = model.predict(x_test)
```

In [18]:

```
for name in preds:
    display(HTML(f"<h3>{name}</h3>"))
    print("Confusion Matrix:\n", confusion_matrix(y_test, preds[name]))
    print("\nClassification Report:\n", classification_report(y_test, preds[name]))
```

LogisticRegression

```
Confusion Matrix:
 [[3280  243]
 [ 155 3880]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.93      0.94      3523
           1       0.94      0.96      0.95      4035

    accuracy                           0.95      7558
   macro avg       0.95      0.95      0.95      7558
weighted avg       0.95      0.95      0.95      7558
```

KNN

```
Confusion Matrix:
 [[3452   71]
 [  80 3955]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      3523
           1       0.98      0.98      0.98      4035

    accuracy                           0.98      7558
   macro avg       0.98      0.98      0.98      7558
weighted avg       0.98      0.98      0.98      7558
```

DecisionTreeClassifier

```
Confusion Matrix:
 [[3506   17]
 [  30 4005]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      3523
           1       1.00      0.99      0.99      4035

    accuracy                           0.99      7558
   macro avg       0.99      0.99      0.99      7558
weighted avg       0.99      0.99      0.99      7558
```

In [23]:
```
rf_importances = models['RandomForestClassifier'].feature_importances_
feat_imp = pd.Series(rf_importances, index=selected_features).sort_values(ascending=True)
```

In [24]:
```
plt.figure(figsize=(8,6))
feat_imp.plot(kind='barh')
plt.title("Random Forest Feature Importance")
plt.xlabel("Importance")
plt.show()
```

