Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Statistics

Master's Thesis

Machine Learning for Cyber Security: Mitigating Cyber Attacks and Detecting Malicious Activities in Network Traffic

Prepared by: Abenezer Desalgn Dana
Thesis Supervisor: Ing. Tomáš Hlavsa, Ph.D.

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

B.Sc. ABENEZER DESALGN DANA, BSc

Systems Engineering and Informatics
Informatics

Thesis title

Machine Learning for Cyber Security: Mitigating Cyber Attacks and Detecting Malicious Activities in Network Traffic

---

Objectives of thesis

General Objectives
The main objective is to evaluate the efficiency of Machine learning models to enhance Intrusion Detection system.
Specific objectives
• To study scientific and expert literature,
• To detect anomaly activities,
• Train the model and test different Machine learning algorithms and recommend the better one for future use.

Methodology

-Data collection

The data-collecting process includes acquiring both primary and secondary information. The results from Weka are primary data, whereas literature and research papers are secondary data utilized to investigate existing knowledge in the field.

-Choice of the dataset

Since 1998, the DARPA Dataset for IDS Evaluation has been the most widely used dataset among intrusion detection researchers.

-Dataset Modification and Integration

In the original dataset, the attacks are classified only as an attack and normal NSL-KDD.

To categorize it perfectly, I will group it into 4 major classes.

I. Denial of Service Attack (DOS):

II. Users to Root Attack (U2R):

III. Remote to Local Attack (R2L):

IV. Probing Attack (PROBE):

and finally, do analysis using python and WEKA.

The proposed extent of the thesis

60 – 80 pages

Keywords

WEKA, IDS, Cyber Security, Machine Learning

Recommended information sources

Dawson, M., Bacius, R., Gouveia, L. and Vassilakos, A. (2021). Understanding the Challenge of Cybersecurity in Critical Infrastructure Sectors. Land Forces Academy Review, 26(1), pp.69-75.

Ferriyan, A., Thamrin, A., Takeda, K. and Murai, J. (2021). Generating Network Intrusion Detection Dataset Based on Real and Encrypted Synthetic Attack Traffic. Applied Sciences, 11(17), p.7868.

Hastie, T., Tibshirani, R. and Friedman, J.H. (2017). The elements of Statistical Learning: Data Mining, Inference, and prediction. New York, NY, USA: Springer.

Chow, K.H., Deshpande, U., Seshadri, S. and Liu, L. (2021). SRA: Smart Recovery Advisor for Cyber Attacks. In Proceedings of the 2021 International Conference on Management of Data (pp. 2691-2695).

Kuhn, M. and Johnson, K. (2019). Applied predictive modeling. New York: Springer.

Expected date of thesis defence

2022/23 SS – FEM

The Diploma Thesis Supervisor

Ing. Tomáš Hlavsa, Ph.D.

Supervising department

Department of Statistics

Electronic approval: 21. 1. 2023

Ing. Tomáš Hlavsa, Ph.D.

Head of department

Electronic approval: 28. 2. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

Dean

Prague on 26. 03. 2023

Declaration

I hereby declare that this thesis is the result of my original work and that no part of it has been submitted for another degree or diploma at any university or other tertiary institution. Any assistance that I have received in my research work and the preparation of this thesis has been acknowledged.
I also declare that the work presented in this thesis is free from plagiarism and any other academic malpractice. All sources used in this work have been appropriately cited and referenced.

In Prague on 25.03.2023 ----------

Abenezer Desalgn Dana

Acknowledgement

First and foremost, I would like to express my gratitude to Almighty God that gave me the physical and mental strength, determination, and good health to complete this master's thesis. Secondly, I would like to extend my sincere appreciation to my supervisor, Ing. Tomas Hlavsa Ph. D, for his amazing support and valuable guidance throughout the writing process of this thesis.

I'd like to express my heartfelt gratitude to my sister Lulit(Mitu) and our university contact Marin Kozak,. A special thanks to Kidest Kasa(Mekiye) and Edema Shukule(Abba), the engineers of my life, and my loving wife Yemisirach Hailu for her care and encouragement.

Machine Learning for Cyber Security: Mitigating Cyber Attacks and Detecting Malicious Activities in Network Traffic

Abstract:

The increasing dependence on computer networks for communication and critical operations has led to a rise in cyber-attacks and malicious activities. Traditional security measures such as firewalls and intrusion detection systems are no longer sufficient to protect against modern cyber threats. Machine learning has emerged as a promising approach for detecting and mitigating cyber-attacks.

This thesis explores the use of machine learning techniques for cyber security. The primary focus is on the detection of malicious activities in network traffic. The research involves the analysis of network traffic data using various machine learning algorithms, including Decision Trees, Random Forests, Support Vector Machines, and Logistic regression.

The study includes the collection and pre-processing of network traffic data from the NSL-KDD dataset, which is a widely used benchmark dataset for intrusion detection. The dataset includes network traffic traces from various types of cyber-attacks, including denial-of-service, probing, and user-to-root attacks.

The results of the study demonstrate the effectiveness of machine learning algorithms for detecting malicious activities in network traffic. The Random Forest and Support vector machine algorithms outperformed other algorithms in terms of accuracy and detection rate. The study also shows the importance of feature selection and pre-processing techniques in achieving high performance in machine learning models.

The findings of this research have practical implications for the development of effective intrusion detection systems and cyber security measures. The use of machine learning algorithms can improve the accuracy and efficiency of current security systems and enhance the ability to detect and mitigate cyber-attacks.

Keywords: WEKA, IDS, Cyber Security, Machine Learning

Contents

LIST OF FIGURES

LIST OF TABLES

List of Abbreviations

DMZ - Demilitarized Zone

RFE- Recursive Feature Elimination

NSL-KDD - *Network Security Lab - KDD Cup 99 Dataset*

IDS- Intrusion Detection System

DoS- Denial of Service

U2R - Privilege Escalation attacks

R2L- Remote Access attacks

ROC - Receiver Operating Characteristic curve

WEKA- Waikato Environment for Knowledge Analysis

SVM - Support Vector Machines

CIA - Confidentiality Integrity and Availability

HIDS - Host Based Intrusion Detection Systems

NIDS - Network Based Intrusion Detection Systems

IDE - Integrated Development Environments

1. Introduction

Cybersecurity has become a growing concern with the widespread use of the internet and the increasing digitization of businesses and individuals' activities. Cyberattacks have become more sophisticated and widespread, causing significant financial losses to companies and individuals. Cybersecurity Ventures (2019) predicts that cybercrime's financial losses will reach up to $6 trillion annually by 2021, these attacks target large and emerging industrial companies, banking sectors, government financial agencies, and individuals, resulting in costly and disruptive effects that can create social and political crises (Ahmad & Khatun, 2020).

The increasing prevalence of cyberattacks highlights the urgent need for advanced and effective intrusion detection systems to minimize unplanned downtime and protect against potential losses. Enterprise businesses are faced with the increasing likelihood of unexpected downtime due to various cyberattacks and security breaches. Network downtime results in financial losses, reputational damage, and questions the credibility of commercial companies. Therefore, prioritizing defense mechanisms against misuse, attacks, and vulnerabilities is crucial in minimizing or avoiding unplanned downtime.

One of the most promising methods to protect systems from new, more complex, and dynamic attacks is the intrusion detection system (IDS). IDSs can detect malicious activity in the network traffic by matching known attacks (signature-based) or detecting new cyber threat anomaly activities (anomaly-based) to evaluate the relative risk of individual threats and autonomously give an effective and appropriate response to the relevant threats using an efficient intrusion prevention system (IPS).

Most IDS currently in use are either rule-based or anomaly detection-based IDS (Jia, Huang, & Zhang, 2021). However, these systems can be limited by the inability to detect new and previously unknown attacks. To address this, researchers have turned to machine learning techniques to develop intelligent models that can automatically learn from data by extracting patterns from network traffic in the training phase. Once the model learns the normal/attack traffic patterns accurately with a low false-positive rate by fulfilling the standard of the accuracy matrix, it can be deployed in the network alongside the firewall.

These algorithms used can learn from the provided training data and generalize when exposed to new untrained data, increasing the detection ability in the big network infrastructure, which may have terabytes of network traffic data.

This thesis aims to identify the best machine learning model that can enhance intrusion detection and detect malicious activities with higher accuracy and a lower false alarming rate.

2. Objectives and Methodology

2.1 Objectives

The main objective is to evaluate the efficiency of different Machine learning models to get the better understanding of the area and finally recommend the better model that can enhance Intrusion Detection which can detect malicious activities with better accuracy and low false alarming rate.

Specific objectives

- To study scientific and expert literature,
- Matching patterns of known attacks
- To detect anomaly activities
- Train the model and test different Machine learning algorithms and recommend the better one for future use.

2.2. Motivation

Cyber-attacks are a significant threat to individuals, organizations, and governments, with financial losses caused by cybercrime predicted trillions of dollars annually. Good security techniques and monitoring mechanisms are crucial for ensuring the confidentiality, integrity, and availability of data. While intrusion detection systems (IDSs) have been considered a significant breakthrough, traditional IDSs can be limited in detecting new and previously unknown attacks. Machine learning techniques provide a promising approach to developing intelligent intrusion detection systems that can effectively detect various types of cyber-attacks.

2.3. Methodology

Choice of the Dataset

To achieve the objective stated above the dataset used in this study is the NSL-KDD dataset, which is a widely used benchmark dataset for evaluating intrusion detection systems (Tavallaee et al., 2009). This dataset was chosen due to its popularity and the fact that it includes a variety of different types of cyberattacks. The dataset contains a total of 41 features and is divided into a training set of 125,973 instances and a testing set of 22,544 instances. The dataset includes four types of attacks: Probe, DoS, U2R, and R2L.

Choice of Data Mining Software

The data mining software used in this study is Python, which is a popular programming language for data analysis and machine learning. Python provides a wide range of libraries and packages that facilitate data analysis and machine learning, such as NumPy, Pandas, Matplotlib, and Scikit-learn. Scikit-learn is a powerful machine learning library in Python that provides a variety of machine learning algorithms and tools for data preprocessing, model selection, and model evaluation. It was chosen for this study due very rich machine learning library content and to get support from vast number data science community mainly python has high speed than the tools like SPSS, Statistica and even over R and SAS (Jones, 2019).

Data Pre-Processing

First, important python libraries as follows.

```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.metrics import classification_report, confusion_matrix, precision_score, recall_score, f1_score
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.calibration import calibration_curve
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
import itertools
import random
```

Figure 1: Important python libraries. (own work)

Once the libraries were imported, the downloaded data was brought into the Spider IDE by indicating the directory where the data is saved.

```
train = pd.read_csv('KDDTrain+.txt', header=None,names=col_names)
test = pd.read_csv('KDDTest+.txt', header=None,names=col_names)
```

Figure 2: Indicating the directory where data is available. (own work)

Mapping Normal as 0 and Attack as 1 (Encoding)

```
# Train Dataset

is_attack = train.attack.map(lambda a: 0 if a == 'normal' else 1)

# Test Dataset

test_attack = test.attack.map(lambda a: 0 if a == 'normal' else 1)
✓ 0.1s
```

Figure 3: Changing nominal values into binary. (own work)

Before training the models, the dataset undergoes a pre-processing stage to ensure optimal model performance. One critical step in this process is encoding the target variable, which consists of two categories: Normal and Attack. To facilitate the use of machine learning algorithms, the target variable is transformed into numerical values by mapping "Normal" instances to 0 and "Attack" instances to 1.

This encoding enables the algorithms to process the data correctly and make accurate predictions. The Label Encoder class from the sklearn.preprocessing module is used to perform this encoding.

```
# One-hot encode categorical features
categorical_features = ['protocol_type', 'service', 'flag']
for feature in categorical_features:
    encoder = LabelEncoder()
    data[feature] = encoder.fit_transform(data[feature])
    encoders[feature] = encoder
```

Figure 4: One hot encoding of categorical features (own work)

Classifying Attacks into Four Categories

The dataset consists of numerous attack types. To simplify the classification process, we group these attacks into four primary categories: Denial of Service (DoS), Probe, Privilege Escalation, and Remote Access. This approach enables a more manageable and focused analysis while still addressing the different types of cyberattacks.

Categorizing the attacks types in to four major classes.

```
# lists to hold our attack classifications

dos_attacks = ['apache2','back','land','neptune','mailbomb','pod','processtable','smurf','teardrop','udpstorm','worm']
probe_attacks = ['ipsweep','mscan','nmap','portsweep','saint','satan']
privilege_attacks = ['buffer_overflow','loadmdoule','perl','ps','rootkit','sqlattack','xterm']
access_attacks = ['ftp_write','guess_passwd','http_tunnel','imap','multihop','named','phf','sendmail','snmpgetattack',
                  'snmpguess','spy','warezclient','warezmaster','xclock','xsnoop']
✓ 0.1s
```

Figure 5: Creating list to hold the categorized attack group (own work)

By categorizing the attacks, the model can be trained to recognize these broader categories, allowing for more accurate and efficient classification. The labels used for these categories are DoS, Probe, U2R, R2L, and Normal (for non-attack instances).

Data Scaling

Before training models, data has been standardized the features of the dataset to ensure that all features in all categories DoS, Probe, R2L, and U2R. have the same scale.

```
scaler1 = preprocessing.StandardScaler().fit(X_DoS)
X_DoS=scaler1.transform(X_DoS)
scaler2 = preprocessing.StandardScaler().fit(X_Probe)
X_Probe=scaler2.transform(X_Probe)
scaler3 = preprocessing.StandardScaler().fit(X_R2L)
X_R2L=scaler3.transform(X_R2L)
scaler4 = preprocessing.StandardScaler().fit(X_U2R)
X_U2R=scaler4.transform(X_U2R)
```

Figure 6: Feature scaling of each attack group (own work)

Examining the dataset

Examining the dataset to understand its characteristics, such as data distribution is critical, and this proses of grouping and see the distribution of attack groups are done this way.

```
# Calculating Number of Occurences of Each Type of Attack

DOS = ['apache2','back','land','neptune','mailbomb','pod','processtable','smurf','teardrop','upstorm','worm']
Probe = ['ipsweep','nmap','mscan','portsweep','saint','satan']
U2R = ['buffer_overflow','loadmodule','perl','ps','rootkit','sqlattack','xterm']
R2L = ['ftp_write','guess_passwd','httptunnel','imap','multihop','named','phf','sendmail','Snmpgetattack','spy','snmpguess','warzclient',
       'warzmaster','xlock','xsnoop']
count = {'DOS':0, 'Probe':0, 'U2R':0, 'R2L':0, 'Normal':0}
for attack in train.attack:
    if attack in DOS:
        count['DOS'] += 1
    elif attack in Probe:
        count['Probe'] += 1
    elif attack in U2R:
        count['U2R'] += 1
    elif attack in R2L:
        count['R2L'] += 1
    else:
        count['Normal'] += 1
count
[223]  ✓ 0.1s
...  {'DOS': 45927, 'Probe': 11656, 'U2R': 52, 'R2L': 85, 'Normal': 68253}
```

Figure 7: Calculating number of occurrences of each type of attack. (Own work)

Feature Selection Using Recursive Feature Elimination (RFE)

Feature selection plays an important role in building effective machine learning models, as it helps in identifying the most relevant features for the task and reduces the computational complexity of the models. In this study, we employ Recursive Feature Elimination (RFE) as a feature selection technique. RFE is a recursive process that works by fitting a model, ranking the features based on their importance, and removing the least important features one by one. This process is repeated until a desired number of features are selected. RFE is particularly useful in identifying the most relevant features that contribute to the classification task and can help improve the performance of the models.

```
# Feature selection using RFE with a Decision Tree model
model = DecisionTreeClassifier(random_state=0)
rfe = RFE(model, n_features_to_select=20)
fit = rfe.fit(X, y)

# Get the selected features
selected_features = X.columns[fit.support_]
print('Selected features:', list(selected_features))
```

Figure 8: Features selecting using Recursive Feature Elimination (RFE). (Own work)

To implement RFE, the RFE function from the scikit-learn feature selection library is applied to the four major classes: DoS, Probe, R2L, and U2R, to select the most relevant features for each category. By doing so, it can be ensured that our machine learning models are trained on the most informative features, leading to more accurate and efficient classification results.

Choice of Data Mining Algorithms

In this study, we will evaluate the performance of four different machine learning algorithms for intrusion detection:

- Decision Tree
- Random Forest
- Support Vector Machine
- Logistic Regression

These algorithms were chosen based on their effectiveness in detecting different types of cyberattacks and their ease of implementation. Decision Tree and Random Forest are tree-based algorithms that are commonly used for classification problems. Support Vector Machine is a powerful algorithm that is effective in detecting complex decision boundaries. Logistic Regression is a simple but effective algorithm that is commonly used for binary classification problems.

Decision Tree

Decision tree feature importance

The decision tree feature importance done in the figure 9 using the clf.feature_importances_ attribute of the fitted decision tree model, sorts them in descending order using the np.argsort() function, and selects the top 13 features using the [:13] slicing. The importance of features in a decision tree is determined by how much they reduce the impurity of the target variable in a given split. The Gini impurity or entropy is commonly used to measure impurity. Features with higher importance scores are those that reduce impurity the most and are therefore more useful in making predictions.

```
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]
feature_names = X_train.columns[indices][:13]  # Get top 13 feature names

plt.figure(figsize=(12,6))
plt.title("Important Predictors")
plt.barh(range(13), importances[indices][:13],
       color="r", align="center")
plt.yticks(range(13), feature_names)
plt.ylabel("Feature Name")
plt.xlabel("Importance")
plt.show()
```

Figure 9: Decision tree feature importance plot. (Own work)

Decision tree classifier model

Decision tree classifier model using the Gini criterion for splitting and a maximum depth of 4. The trained model is then used to predict the target variable for both the training and testing data, and

the classification accuracy scores are printed out. Additionally, a classification report is printed out, which contains metrics such as precision, recall, and F1 score for each class in the target variable.

```
from sklearn import tree
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

clf = tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=4)

clf = clf.fit(X_train, y_train)
y_train_pred = clf.predict(X_train)
pred_dt = clf.predict(X_test)
```

Figure 10: Decision tree classifier model. (Own work)

The pruned version of the decision tree is plotted as shown below.

```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

plt.figure(figsize=(30, 20))  # Increase the figure size
plot_tree(clf, filled=True, feature_names=X.columns, class_names=attack_labels, rounded=True, fontsize=8)  # Reduce the font size
plt.savefig("decision_tree_full_size.png", dpi=300, bbox_inches="tight")  # Save the image
plt.show()
```

Figure 11: Pruned version of the decision tree. (Own work)

Logistic Regression

Future selection

RFE (Recursive Feature Elimination) is a technique is used to select the best feature as shown in the figure 12 below. It works by recursively removing attributes and building a model on those attributes that remain. It uses cross-validation to determine which features contribute the most to predicting the target variable and discards those that contribute the least.

```
# Create a logistic regression object
clf = LogisticRegression(random_state=0, max_iter=10000)
# Create the selector object with RFECV
selector = RFECV(clf, cv=5, step=1, scoring='accuracy')
# Fit the selector to the training data
selector.fit(X_train, y_train)

# Transform the training and test data to include only the selected features
X_train_selected = selector.transform(X_train)
X_test_selected = selector.transform(X_test)
```

Figure 12: Feature selection using Recursive Feature Elimination. (Own work)

Logistic regression Model

The equation used in logistic regression model is a linear combination of the input features, with learned weights and biases. It is represented as

$$Z = wX + b \qquad \text{eq1}$$

where Z is the weighted sum of inputs, w is the learned weights, X is the input features, and b is the bias term. The probabilities of the output classes are obtained using the sigmoid function, which maps the weighted sum Z to a value between 0 and 1. The sigmoid function is applied to the linear combination of the input features using the equation:

$$1 / (1 + \exp(-Z)) \qquad \text{eq2}$$

The dot product of the learned weights and the input features, X_test, plus the bias term, b, is computed using the equation:

$$z = w\_1x\_1 + w\_2x\_2 + ... + w\_n*x\_n + b \qquad \text{eq3}$$

where w_i represents the weight associated with the i-th feature, x_i represents the i-th feature value, b is the bias term, and z is the log-odds or logit of the predicted probability of the positive class.

In code, this equation is computed as:

```
z = np.dot(w, X_test.T) + b[:, np.newaxis]
```

where w is the array of learned weights, X_test is the matrix of test set inputs, T is the transpose operation, and b is the array of learned bias terms. In terms of the attack classes classification, the logistic regression algorithm predicted the probability of an instance belonging to a particular attack class. The learned weight and bias terms will be specific to the attack class being predicted. By training the logistic regression algorithm on a dataset of labeled instances, the algorithm can learn the weight and bias terms that best discriminate between the different attack classes. The resulting model can then be used to predict the attack class of new instances based on their features.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

clf = LogisticRegression(random_state=0, max_iter=10000)

clf = clf.fit(X_train, y_train)
y_train_pred = clf.predict(X_train)
pred_lr = clf.predict(X_test)

# Compute the predicted probabilities for each class label
probas = clf.predict_proba(X_test)

# Compute the weighted sum of the inputs using the learned weights and bias
w = clf.coef_
b = clf.intercept_
z = np.dot(w, X_test.T) + b[:, np.newaxis]
```

Figure 13: Logistic regression model. (Own work)

Support Vector Machines (SVMs)

Feature selection

Support Vector Machines (SVMs) is used from the scikit-learn library to create a model with a linear kernel and a random state of 0. Then we are creating a Recursive Feature Elimination (RFE) object with 13 features and fitting it to the training data. RFE selects the best features for the given model by recursively removing the least important features based on the feature ranking.

```
# Create an SVM model
clf = SVC(kernel='linear', random_state=0)
# Create the RFE object with 13 features
selector = RFE(clf, n_features_to_select=13, step=1)
# Fit the RFE object to the training data
selector.fit(X_train, y_train)
```

Figure 14: Recursive Feature Elimination (RFE) for logistic regression model. (Own work)

Support vector machine model

SVM model with the selected features from the feature selection step. It then fits the model to the training data and makes predictions on the test data.

```
# Create a SVM model with the selected features
clf = SVC(kernel='linear', random_state=0)

# Fit the model to the training data
clf.fit(X_train_selected, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test_selected)
```

Figure 15: Support vector machine model. (Own work)

Random forest model

Random forest model Random forest model with 100 trees and a maximum depth of 3 using RandomForestClassifier(n_estimators=100, max_depth=3, random_state=0). The model trained on the training data using rf.fit(). We obtain the feature importance scores using the feature_importances_ attribute and sort them in descending order using pd.Series().sort_values(). Finally, we select the top 13 features and their corresponding feature scores using the [:13]

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
import pandas as pd

# Create a random forest model with 100 trees and max_depth=3
rf = RandomForestClassifier(n_estimators=100, max_depth=3, random_state=0)

# Perform Recursive Feature Elimination
selector = RFECV(estimator=LogisticRegression(max_iter=10000), step=1, cv=5, scoring='accuracy')
selector.fit(X_train, y_train)

# Fit the random forest model on the selected features
rf.fit(X_train.iloc[:, selector.support_], y_train)

# Get the feature importances
feature_importances = pd.Series(rf.feature_importances_, index=X_train.iloc[:, selector.support_].columns)
```

Figure 16: Random Forest model. (Own work)

A decision tree for random forest generated as shown below.

```
Plot a decision tree from the random forest model
t.figure(figsize=(20, 10))
ot_tree(rf.estimators_[0], feature_names=X_train.columns, class_names=['Normal','DoS','Probe','Privilege','Access'], filled=True, max_depth=3)
t.show()
```

Figure 17: Decision tree for random forest. (Own work)

Model performance

The performance of the models will be evaluated using various performance metrics, including accuracy, precision, recall, and F1-score. The accuracy of the models will be the primary metric used to evaluate their effectiveness in detecting various types of cyberattacks accurately. To ensure the accuracy and reliability of the results, the experiments will be repeated multiple times using different random seeds, and the results will be averaged.

The results of the model evaluation will be analyzed to identify the most effective machine learning models for intrusion detection. The strengths and limitations of different machine learning approaches for intrusion detection will be discussed. The results will be compared to previous research to determine the performance of the proposed models compared to other models. Based on the results and analysis, a conclusion will be drawn, and recommendations for future research will be provided.

In summary, the proposed methodology involves using the NSL-KDD dataset to evaluate the performance of different machine learning models for intrusion detection. The models will be trained and evaluated using various performance metrics, and the results will be analyzed to identify the most effective models for intrusion detection. The proposed methodology will provide insights into the strengths and limitations of different machine learning approaches for intrusion detection.

Model Performance Evaluation

To evaluate the performance of the machine learning models in the context of this thesis, multiple performance metrics will be used. These metrics will help assess the models' effectiveness in accurately detecting various types of cyberattacks. The selected metrics are:

Accuracy: The proportion of correctly classified instances out of the total instances. It is calculated as

(True Positives + True Negatives) / eq 4

(True Positives + False Positives + True Negatives + False Negatives)

Precision: The proportion of true positive instances among the instances predicted as positive. It is calculated as

True Positives / (True Positives + False Positives) eq 5

Recall (Sensitivity) : The proportion of true positive instances among the actual positive instances. It is calculated as True Positives / (True Positives + False Negatives) eq 6

F1 Score: The harmonic means of precision and recall, which provides a balanced measure of both metrics. It is calculated as

2 * (Precision * Recall) / (Precision + Recall) eq 7

Receiver Operating Characteristic (ROC) Curve

A graphical representation of a classifier's performance, showing the trade-off between true positive rate (sensitivity) and false positive rate (1 - specificity) at various thresholds.

These metrics will be used to compare the performance of different machine learning models in detecting various types of cyberattacks. By analyzing these metrics, we can identify the strengths and weaknesses of each model and determine the most effective models for intrusion detection.

Additionally, visualizations such as confusion matrices, ROC curves, and calibration plots will be used to provide further insights into the performance of the models. These visualizations will help identify any potential issues with the models, such as misclassifications, and support the interpretation of the performance metrics.

In summary, the proposed methodology for evaluating the performance of the machine learning models includes the use of various metrics, formulas, and visualizations to assess the effectiveness of the models in detecting different types of cyberattacks. This comprehensive evaluation will enable us to draw meaningful conclusions and provide recommendations for future research in the field of intrusion detection.

What is a Confusion Matrix

A confusion matrix is a combination of expected and actual categorization data that is used in a specific system. For the performance analysis, the data obtained for such a system is examined. A confusion square matrix including positive and negative rates is constructed during predictive analysis (both true and false)

	Predicted As Normal	**Predicted As Attack**
Actually Normal	**TN**	**FP**
Actually Attack	**FN**	**TP**

Table 1: Confusion Matrix: Source: (Smith, 2020)

3. Literature review

3.1. Computer security background

Computers and the Internet are now vital in practically every aspect of our life. Since the invention of the personal computer, the number of users has grown at an exponential rate, and it is today hard to imagine a company, university, or even small shops using computers to save all its customer, purchase, and inventory data in an electronic database or the computer.

With the ability to link many computers and networks, the need to safeguard all this data and computers from intruders (hackers) who want to steal secret information for their own gain or just delete or change vital data appeared.

Simultaneously with the growth of the Internet was the security requirements grew exponentially, and there was no way to stop them. Equilibrium among privacy and use resources is a difficult notion to grasp; the network must also be adaptable enough to meet the needs required to track down the cybercriminals.

With the increasing use of computers and the internet, cyber security has become a major concern for both individuals and businesses. The growth in the number of users and the interconnectivity of networks has resulted in an increased risk of cyber-attacks and the need for robust security measures to protect against these attacks (NIST, 2018).

There are various security methods available to secure computer systems, but it is nearly impossible to create an invulnerable system. A robust security strategy and a thorough risk analysis, combined with well-educated users, can significantly improve the security of a computer system (Kumar, 2015).

The three primary pillars of computer security are confidentiality, integrity, and availability (CIA), which represent the goals of computer security (Kizza, 2017). Confidentiality is about the protection of sensitive information from unauthorized access. Integrity ensures that data is accurate, complete, and has not been tampered with by an unauthorized party. Availability refers to the ability of authorized users to access the system and its resources. Any activity that jeopardizes the confidentiality, integrity, or availability of information or computer resources is considered an intrusion.

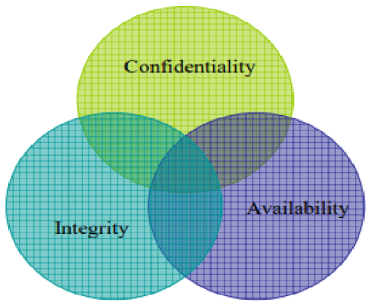


Figure 18: System Integrity: Source: (Ruiz, I.P. and Ramón, 2008)

One method of identifying an intrusion is through the characterization of unusual behavior and system usage (Jin et al., 2020). This type of intrusion detection aims to measure a user's typical behavior and detect any deviations from it. There are three primary types of attacks based on who carries them out: external penetration, internal penetration, and resource abuse (Kizza, 2017). External penetration refers to attacks from outside the network firewall, while internal penetration is carried out by unauthorized users within the firewall. Resource abuse occurs when authorized users misuse data or resources in unexpected and undesired ways.

To identify and prevent intrusions, it is essential to distinguish between the three primary types of attacks: external penetration, internal penetration, and resource abuse. Active and passive security measures can be employed to protect computer systems from these types of attacks. Active security measures, such as firewalls and intrusion prevention systems, are used to filter access to specific services or connections and intervene in the event of an attack. Passive security measures, such as intrusion detection systems, are used to alert the system to potential threats (Chen et al., 2020).

In a survey of big data and security analytics conducted by Ponemon Institute (2018), it was found that most big companies and industries use a variety of security solutions to defend themselves against cyber threats. The survey revealed that the most used security measures were firewalls, endpoint protection, identity and access management, intrusion detection, and intrusion prevention systems.

In summary, computer security is an essential aspect of modern life due to the widespread use of computers and the internet. While there are various security methods available, it is nearly impossible to create an invulnerable system. A combination of active and passive security

measures, as well as well-educated users, can significantly improve the security of computer systems. The use of a variety of security solutions, such as firewalls, intrusion detection, and IPS, is common among big companies and industries to protect against cyber threats.

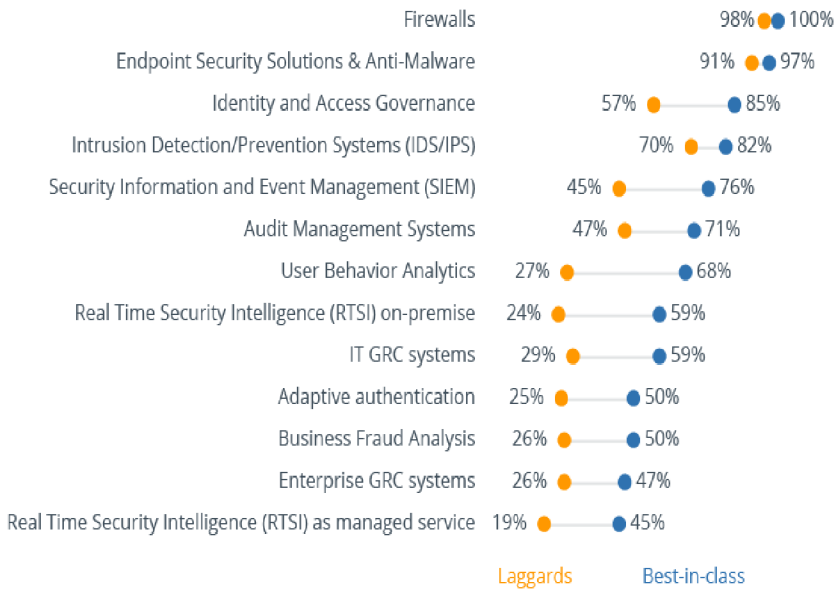


Figure 19: Security Analytics Survey (Oppmann, 2022)

3.2. Overview of Intrusion Detection

The increasing prevalence of cyberattacks highlights the urgent need for advanced and effective intrusion detection systems (IDS) to minimize unplanned downtime and protect against potential losses. An IDS is a security solution that monitors network traffic for signs of unauthorized access, misuse, and/or policy violations (Mishra, 2019).

Intrusion detection systems can be categorized into two types: signature-based and anomaly-based detection systems. Signature-based IDS matches known attack signatures against the traffic passing through the system. On the other hand, anomaly-based IDS detects abnormal network traffic behavior and activities that do not conform to the expected usage patterns (Ghebleh et al., 2021).

To address the limitations of traditional IDS systems, researchers have turned to machine learning (ML) and artificial intelligence (AI) techniques. By leveraging the ability of ML models to automatically learn from data, these systems can learn to identify new and previously unknown attacks. A significant advantage of using ML-based IDS is their ability to adapt to new attacks by detecting and responding to previously unseen patterns of network traffic (Ghebleh et al., 2021).

Some of the most used ML algorithms for network intrusion detection are decision trees, artificial neural networks (ANN), and support vector machines (SVM) (Jain & Mishra, 2019). The choice of algorithm depends on the type of data and the required performance metrics. For example (Jain and Mishra ,2019) evaluated the performance of different ML algorithms on the NSL-KDD dataset and found that a multi-layer perceptron neural network (MLP-NN) outperformed other algorithms in terms of detection accuracy, recall, and F1 score.

In recent years, deep learning (DL) approaches have also been applied to network intrusion detection systems. DL models, such as convolutional neural networks (CNN) and recurrent neural networks (RNN), have been shown to achieve high accuracy and improved performance in detecting new and complex attacks (Li et al., 2022). For example, (Li et al.,2022) proposed a hybrid CNN-LSTM model for network intrusion detection that outperformed other state-of-the-art models on the NSL-KDD dataset.

3.3. Related Works

The paper titled, "Comparison of Machine Learning Techniques for Intrusion Detection System," authors (Shahid et al, 2021) compare the performance of four machine learning algorithms for intrusion detection: Random Forest, Support Vector Machine, Logistic Regression, and Gaussian Naive Bayes. The study aims to determine which algorithm is best suited for detecting intrusion attacks in network traffic.

The authors collected and preprocessed network traffic data from the NSL-KDD dataset and used it to train and test each algorithm. They evaluated the performance of each algorithm using several performance metrics, including accuracy, precision, and recall, F1-score, and ROC curve.

The results of the study showed that Random Forest had the highest accuracy (99.13%) and F1-score (99.13%).
The study also found that Random Forest and Support Vector Machine had the best performance in detecting different types of attacks, including DoS, Probe, R2L, and U2R. The authors conclude that Random Forest and Support Vector Machine are suitable for intrusion detection systems in detecting network attacks with high accuracy and low false positives.

The paper titled "Comparative Analysis of Machine Learning Techniques for Intrusion Detection System" by (Yassin et al.,2021) compares the performance of four machine learning algorithms (random forest, decision tree, k-nearest neighbors, and support vector machine) for intrusion detection on the NSL-KDD dataset. The authors evaluate the algorithms based on four metrics: accuracy, precision, recall, and F1-score. They also conduct a feature selection process to identify the most important features for intrusion detection.

The results show that random forest and support vector machine outperform decision tree and k-nearest neighbors in terms of accuracy, precision, recall, and F1-score. The authors also find that the selected subset of features leads to better performance compared to using all features.
Comparative Analysis of Machine Learning Techniques for Intrusion Detection" by (Balaji et al., 2022) evaluated the performance of five machine learning algorithms (K-Nearest Neighbor, Support Vector Machines, Decision Trees, Random Forest, and Gradient Boosting) in detecting intrusion attacks in a simulated network environment. The authors used the NSL-KDD dataset and performed feature selection using mutual information gain.

Their results showed that Random Forest and Gradient Boosting outperformed the other algorithms in terms of detection accuracy, false positive rate, and F1-score. The authors also found that mutual information gain-based feature selection significantly improved the performance of the algorithms, especially for K-Nearest Neighbor and Decision Trees.

In conclusion, the study provided insight into the performance of different machine learning algorithms for intrusion detection and showed the effectiveness of mutual information gain-based feature selection in improving the performance of certain algorithms.

3.4. Types of Intrusion Detection System

3.4.1. Host -based Intrusion Detection System

Host-based intrusion detection systems (HIDS) monitor and analyze the events and activities that occur within a single host or endpoint. HIDS aim to detect any suspicious or malicious activity that could compromise the security of the host or the entire network. HIDS have access to a wealth of information about the host's system calls, processes, and other system-level activity, which makes them capable of detecting attacks that cannot be identified by network-based intrusion detection systems (NIDS) (Huang et al., 2016).

HIDS typically work by comparing the observed system activity to a predefined set of rules or profiles that describe normal behavior. Any deviation from the established rules or profiles is considered an anomaly and is flagged as a potential security threat. The detected anomalies are then analyzed by security administrators or automated systems to determine whether they represent actual attacks or false positives (Pawar & Bhavsar, 2018). One of the main advantages of HIDS is their ability to detect insider threats, which are attacks carried out by authorized users with access to the system. HIDS can also detect attacks that are specifically designed to evade network-based detection systems, such as those that use encrypted channels or tunneling protocols (Huang et al., 2016).

Several studies have evaluated the effectiveness of HIDS in detecting various types of attacks. For example, (Cui et al., 2018) proposed a HIDS based on machine learning algorithms for detecting malicious code execution on Windows operating systems. Their results showed that the proposed HIDS achieved high accuracy and low false positive rates in detecting different types of malicious code.

3.4.2. Network -based Intrusion Detection System

Network-based intrusion detection systems (NIDS) are designed to monitor network traffic and detect potential intrusion attempts. They analyze network packets and identify patterns that match

known attack signatures or anomalous behavior that deviates from normal network activity. NIDS can be deployed at various locations in the network topology, such as at the perimeter, core, or on specific subnets.

One common type of NIDS is signature-based detection. Signature-based NIDS uses a database of pre-defined attack signatures to match against network traffic and identify potential attacks. Researchers in (S. Singh & K. Singh, 2019) proposed a signature-based NIDS system that uses a multi-layered approach to detect attacks in real-time. The system uses multiple detection engines, including a rule-based engine, signature-based engine, and machine learning engine, to increase the accuracy of the detection. Anomaly-based NIDS, on the other hand, use machine learning and statistical techniques to identify abnormal network behavior that may indicate an attack. Researchers in (N. A. O. Al-Nassiri et al., 2019) proposed an anomaly-based NIDS that uses clustering and classification techniques to identify potential attacks. The system utilizes K-means clustering to group network traffic into different clusters and then uses a support vector machine (SVM) to classify each cluster as either normal or anomalous.

Another type of NIDS is hybrid detection, which combines signature-based and anomaly-based detection techniques. Researchers in (A. U. Hassan et al., 2019) proposed a hybrid NIDS that utilizes a combination of signature-based detection, machine learning, and deep learning techniques. The system uses a convolutional neural network (CNN) to extract features from network traffic and a random forest classifier to classify the traffic as normal or malicious.

Intrusion prevention systems (IPS) are another type of NIDS that not only detect potential attacks but also take preventive measures to stop them. Researchers in (K. M. Alqudah et al., 2019) proposed an IPS that uses a combination of signature-based and behavior-based detection techniques. The system uses Snort, a popular open-source NIDS, for signature-based detection and Bro, another open-source network security monitoring tool, for behavior-based detection. The IPS also utilizes a honeypot, a decoy system designed to attract and detect attacks, to provide an additional layer of security. Therefore network-based intrusion detection systems are critical for identifying and preventing potential attacks in computer networks. They can be deployed at various locations in the network topology and utilize different detection techniques, such as signature-based, anomaly-based, and hybrid detection, to improve their accuracy. The use of

intrusion prevention systems provides an additional layer of security by not only detecting but also preventing potential attacks.

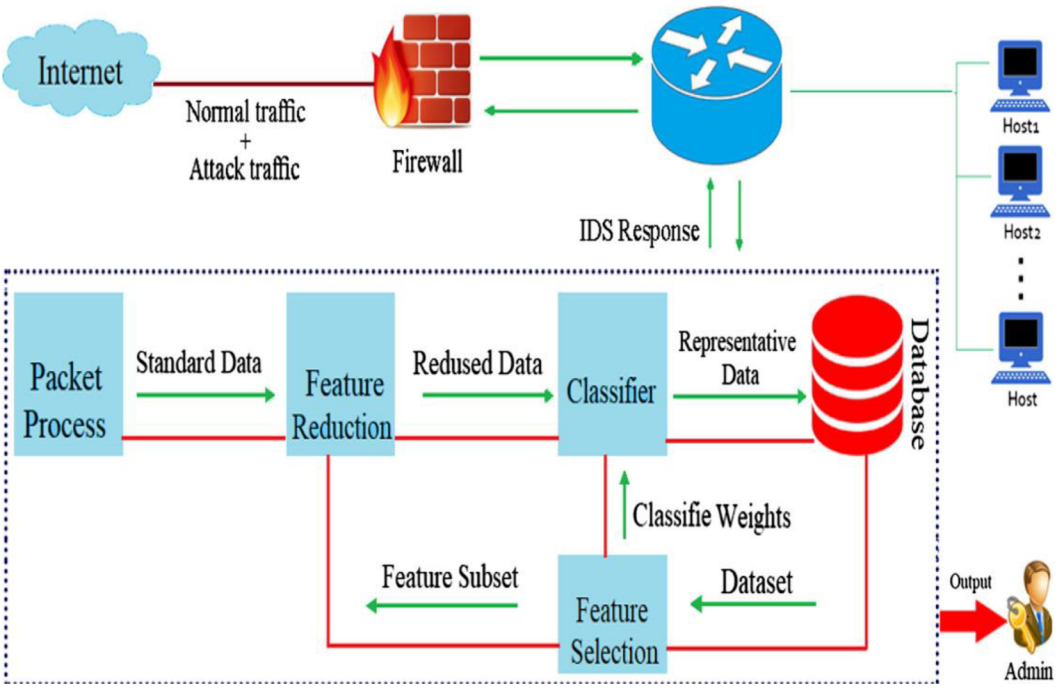


Figure 20: Architecture of a Network-Based IDS security System (Hosseinzadeh, 2022)

3.4.3. Strength of network-based IDS

Network-based intrusion detection systems (NIDS) have several strengths that make them an effective security solution for protecting against network attacks. Firstly, NIDS can monitor network traffic in real-time, providing immediate alerts to potential threats. This allows for quick responses to prevent attacks from causing significant damage.

Secondly, NIDS can detect network-based attacks that originate from external sources and internal sources, including unauthorized network access attempts, malware infections, and denial-of-service (DoS) attacks. Thirdly, NIDS can analyze network traffic at different levels, including packet-level, session-level, and application-level. This provides a comprehensive view of network activity, making it easier to detect and prevent sophisticated attacks. Lastly, NIDS can be used to

monitor large-scale networks and can be easily deployed and managed, making them an effective solution for protecting against network attacks in complex environments.

3.4.4. Weakness of network-based IDS

There are several weaknesses of network-based IDS that should be considered, including:
Inability to detect attacks within encrypted traffic: Network-based IDS cannot detect attacks within encrypted traffic since they cannot analyze the contents of the encrypted packets (NIST, 2018). False positives: Network-based IDS can generate a high number of false positives, particularly when detecting anomalies, as legitimate network activity may be interpreted as an attack (Liu et al., 2019).

Dependence on network topology: Network-based IDS may not be effective in large and complex networks, where the topology is constantly changing. The IDS may miss attacks that occur outside its range of monitoring or that are not visible due to network segmentation (Chen et al., 2020).
Performance degradation: Network-based IDS may suffer from performance degradation due to the high volume of network traffic that needs to be analyzed. This can result in delays and missed attacks (Jain & Mishra, 2019). Susceptibility to evasion techniques: Attackers may use evasion techniques to bypass network-based IDS, such as fragmentation, tunneling, or packet obfuscation (Ghebleh et al., 2021).
Overall, network-based IDS can be an effective security solution for detecting network-based attacks. However, it is important to consider the limitations and weaknesses of the system and to use it in combination with other security measures to provide comprehensive protection against cyber threats.

3.5. Category according to the learning method

3.5.1. Signature-based detection

Signature-based IDS, also known as rule-based IDS, compares the traffic passing through a system with a database of known attack signatures (Ghebleh et al., 2021). These signatures are pre-defined patterns of network traffic that correspond to known attacks. When the IDS detects a match between the traffic passing through the system and an attack signature, it generates an alert or takes action to prevent the attack (Mishra, 2019).

One advantage of Signature-based IDS is their ability to detect known attacks with high accuracy and low false positives (Agyeman et al., 2020). Signature-based IDS are also relatively simple to implement and can operate in real-time (Jain & Mishra, 2019).

However, Signature-based IDS have limitations. They are unable to detect new or previously unknown attacks that do not match any of the pre-defined attack signatures (Ghebleh et al., 2021). This limitation is particularly significant given the increasing prevalence of new and sophisticated cyber threats that do not have pre-existing signatures (Li et al., 2022).

Despite their limitations, Signature-based IDS remain an important component of a comprehensive network security strategy. They are particularly effective in detecting well-known and widely used attacks that have established signatures, such as the Ping of Death and SQL injection attacks (Mishra, 2019).

3.5.2. Anomaly Based Detection

Anomaly-based intrusion detection systems (IDS) are designed to detect network attacks based on deviations from normal system behavior, as opposed to looking for specific known attack patterns like signature-based IDS. Anomaly-based IDS rely on creating a baseline of normal system behavior and then identifying any activity that deviates from that baseline. If the system identifies activity that falls outside of the normal behavior pattern, it raises an alarm or takes action to block the suspicious activity.

One approach for anomaly-based IDS is statistical anomaly detection, which utilizes statistical methods to establish a baseline of normal behavior and then detect deviations from that baseline. One such method is Principal Component Analysis (PCA), which reduces the dimensionality of data and identifies correlations among variables to determine normal behavior patterns. Other statistical methods used in anomaly-based IDS include clustering, classification, and regression analysis (Xu et al., 2018).

Another approach for anomaly-based IDS is machine learning-based anomaly detection. Machine learning algorithms can learn to identify normal and abnormal behavior patterns based on historical data. These algorithms can be trained on both labeled and unlabeled data, making them capable of detecting previously unknown attacks. Examples of machine learning algorithms used in anomaly-based IDS include decision trees, neural networks, and support vector machines (SVMs) (Alazab et al., 2020).

A third approach for anomaly-based IDS is behavior-based detection, which focuses on identifying behavior patterns that deviate from the user's normal behavior, instead of detecting anomalies in the network traffic itself. This method is useful for detecting insider attacks or other attacks that involve user behavior. The behavior patterns are established through profiling the user's behavior over a period and identifying any deviations from the normal pattern (Saini & Singla, 2019).

In summary, anomaly-based IDS are designed to detect attacks based on deviations from normal behavior patterns, rather than identifying known attack signatures. Statistical methods, machine learning algorithms, and behavior-based detection are commonly used approaches for developing anomaly-based IDS. These methods offer several advantages, such as the ability to detect previously unknown attacks and identifying abnormal user behavior patterns.

3.5.3. Hybrid Intrusion Detection

Hybrid IDS is a combination of two or more IDS types and aims to overcome the limitations of each individual IDS. It can provide more accurate and effective intrusion detection by utilizing multiple detection methods. The combination of signature-based and anomaly-based IDS is a common approach for hybrid IDS.

One study by (Velayutham et al. ,2017) proposed a hybrid IDS that combines signature-based and anomaly-based IDS to improve the accuracy of intrusion detection. The study evaluated the performance of the proposed hybrid IDS on the NSL-KDD dataset and found that it outperformed both signature-based and anomaly-based IDS.

Another study by (Kaur and Singh ,2018) proposed a hybrid IDS that combines machine learning and rule-based techniques to improve the accuracy of intrusion detection. The study evaluated the performance of the proposed hybrid IDS on the KDDCup'99 dataset and found that it achieved better performance compared to individual IDS.

A third study by (Sari et al., 2019) proposed a hybrid IDS that combines three IDS types: signature-based, anomaly-based, and stateful protocol analysis. The study evaluated the performance of the proposed hybrid IDS on the DARPA 2000 dataset and found that it achieved higher detection rates and lower false positive rates compared to individual IDS.

Therefore, hybrid IDS combines multiple IDS types to improve the accuracy and effectiveness of intrusion detection. The combination of signature-based and anomaly-based IDS is a common approach for hybrid IDS, but other combinations of IDS types and techniques can also be used.

3.6. Where IDS Should be Placed in Network Topology

The placement of IDS in a network topology is crucial in maximizing its effectiveness in detecting and preventing intrusions. IDS can be placed in different locations within the network topology, including at the perimeter, internal network, and endpoints (hosts).

Placing IDS at the network perimeter allows it to monitor all incoming and outgoing traffic from external networks. This can help to detect and prevent attacks before they can reach the internal network. However, this approach may not be effective in detecting internal threats or attacks that bypass perimeter defenses (Jain & Mishra, 2019).

Internal network IDS is placed within the internal network to monitor traffic between hosts, servers, and other devices. This can help to detect and prevent internal threats, such as insider attacks, and attacks that bypass perimeter defenses. However, this approach may require additional resources and can be challenging to implement in large or complex networks (Chen et al., 2020).

End-point IDS is placed on individual hosts to monitor and protect against local threats, such as malware infections or unauthorized access. This approach can be effective in detecting and preventing local threats but may not be sufficient to protect against network-level attacks (Mishra, 2019).

The optimal placement of IDS depends on the organization's specific security needs and the network topology. A combination of multiple IDS placed at different locations within the network can provide a more comprehensive security posture.

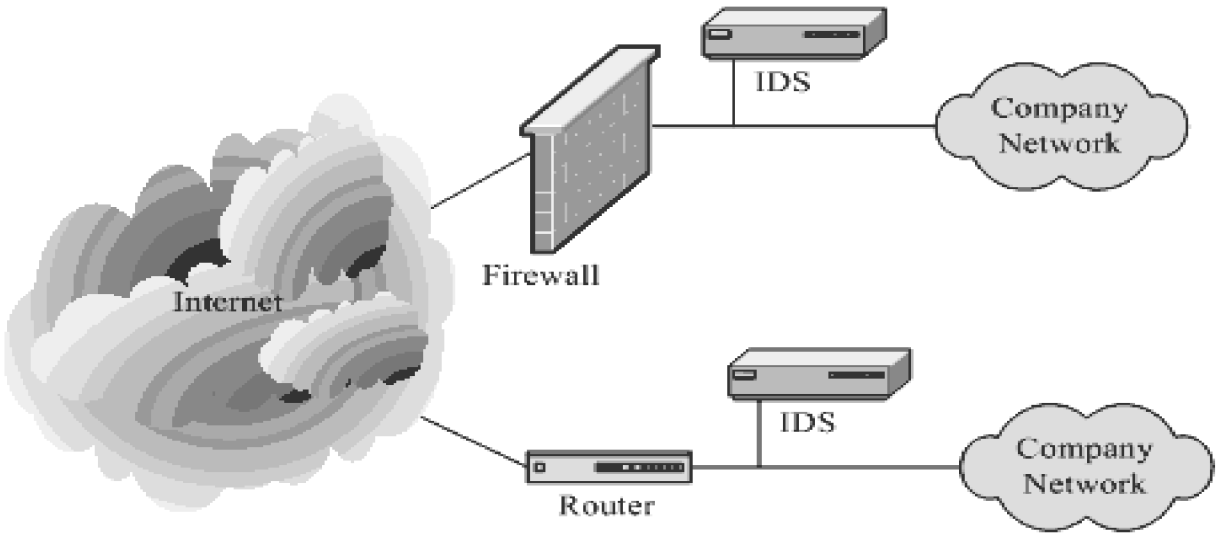


Figure 21: Typical Location for an Intrusion Detection System (Rehman, 2003)

There are several options for integrating IDS technologies into our network, each with its own set of benefits and drawbacks.

As a result, the IDS's placements inside a network have various features. Then we'll see a variety of options inside the same network

Demilitarized Zone (DMZ)

In network security, a Demilitarized Zone (DMZ) refers to a separate, isolated subnetwork that functions as an intermediary between an organization's internal network and the external, untrusted Internet (Liska, 2014). The DMZ provides a layer of protection by hosting publicly accessible services and resources, such as web servers, email servers, and application servers, thus shielding the internal network from direct exposure to external threats (Cole & Northcutt, 2011). By implementing a DMZ, organizations can mitigate the risk of unauthorized access, enhance security measures, and maintain the integrity of sensitive data within their internal networks.

Assume you have a network with a firewall separating the Internet from the demilitarized zone (DMZ). As indicated in the next diagram, there is one that separates the DMZ from the organization's intranet, and another that separates the DMZ from the intranet.

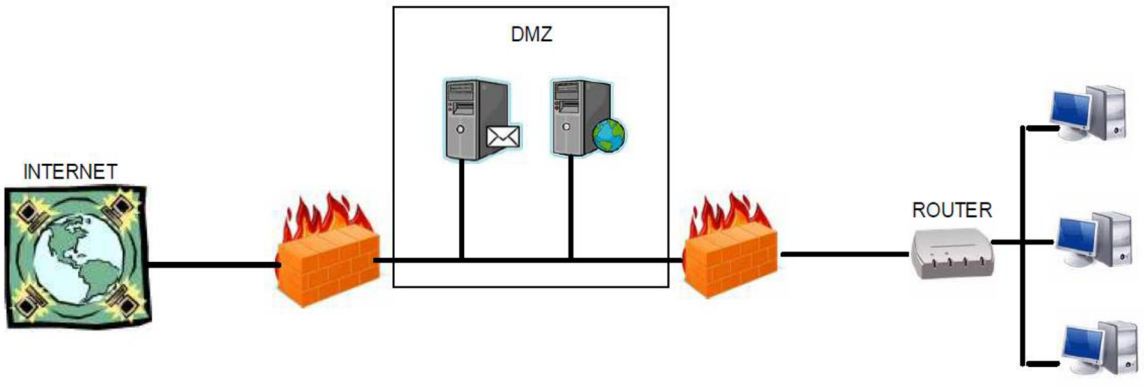


Figure 22: Near DZS (Ruiz, I.P. and Ramón, 2008)

3.6.1. In front of the external firewall

Placing our ids in this position will help to capture any types off attacks coming to attack the external firewall and the organization network from the outer layer. In this location the false alarm will be high this can be considered as a disadvantage.

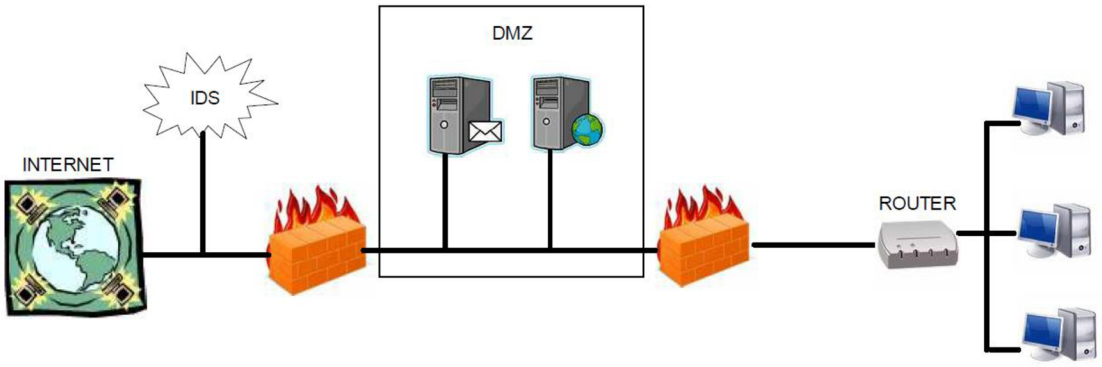


Figure 23: In front of The External Firewall (Ruiz, I.P. and Ramón, 2008)

3.6.2. Behind the external firewall

This location is inside the DZS zone which is in the middle of the firewalls in this location ids catch; the attack passed the main firewall this is very important to get feedback to the second firewall to configure it accordingly.

this area has a low false alarm rate but can be harmed by the saturation of a large amount of data from the network traffic.

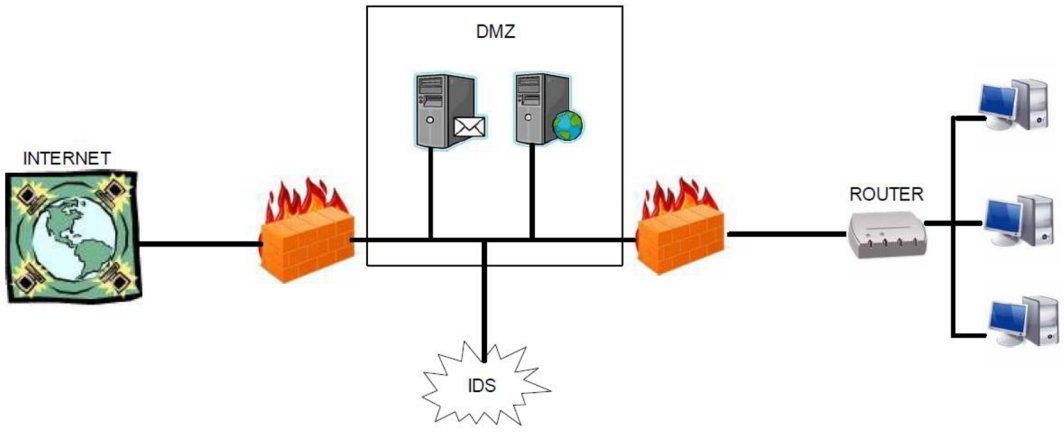


Figure 24: Behind the External Firewall (Ruiz, I.P. and Ramón, 2008)

3.6.3 Behind the second firewall

At this point, we place the IDS near the internal network right side to the second firewall. The amount of network traffic riches here is very small and the significance of the IDS is also very less compared to the other positions.

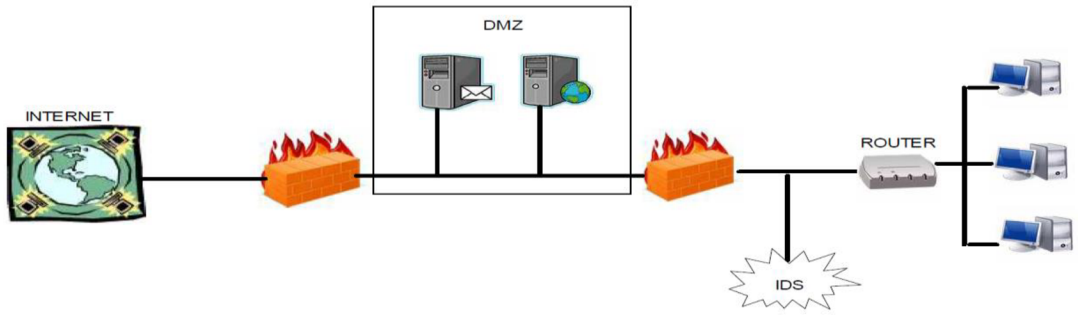


Figure 25: Behind the Second Firewall (Ruiz, I.P. and Ramón, 2008)

3.7. Intrusion detection system administration

It is important to have an IDS policy in place before deploying an IDS on your network. The policy should include rules for identifying intruders and responding to them appropriately (Rehman, 2003). The components of an IDS policy should be defined, including specific rules and how they will be implemented. Additional components can be added based on individual needs (Cisco Systems, 2022).

- Who will keep an eye on the IDS? There should be alerting methods that offer information about intruder behavior depending on the IDS. These alerting systems can be as basic as text files, or they can be more complex, integrating with centralized network management systems such as HP Open View or MySQL databases. Someone must keep an eye on intruder behavior, and the policy must specify who is accountable (s). Pop-up windows or web interfaces can also be used to monitor intruder behavior in real-time. Operators in this situation must be familiar with alerts and their implications in terms of severity levels.

- Who will deal with incidents and how will they be dealt with? There's no purpose in establishing an IDS if you don't know how to handle incidents. You should involve some government entities depending on the seriousness of the occurrence.
- What will the escalation procedure look like (level 1, level 2, etc.)? The escalation procedure is essentially a technique for dealing with incidents. Which events should be escalated to senior management should be explicit should be stated in the policy.

With the IDS policy in place, you'll know how many IDS sensors and other resources you'll need for your network infrastructure. You will be able to determine the cost of ownership of IDS more precisely with this information.

3.8. How to Protect IDS Itself

One of the most pressing concerns is how to safeguard the system on which your IDS software is installed. If the IDS's security is breached, you may begin to get false alarms or no alarms at all (Rehman, 2003). Before launching a cyberattack, the intruder may deactivate IDS. There are a variety of ways to secure your system, ranging from very basic tips to more advanced techniques. A few of them are listed below (Rehman R.U.,2003).

- Turn off any services that are running on the IDS sensor/server. The most typical technique of compromising a system is through a network attack.
- New vulnerabilities are uncovered, and suppliers provide updates. This is a near-constant, non-stop process. The platform on which you're running IDS should be patched with your vendor's most recent versions. If IDS is installed on a Microsoft Windows workstation, for example, you should have all of Microsoft's current security updates loaded every time nonstop.

3.9. Machine Learning Approaches and Implementation on Cyber-Security

Machine learning (ML) has been increasingly utilized in the field of cybersecurity due to its ability to detect and respond to previously unseen attacks automatically. Several studies have shown that

ML approaches can effectively complement traditional security methods and enhance the accuracy of intrusion detection systems (IDSs) (Kumar et al., 2021; Shahid et al., 2021).
One of the most used ML techniques in cybersecurity is supervised learning, which involves training an algorithm on labeled data to identify patterns and make predictions. Decision trees, random forests, and support vector machines are some of the supervised learning algorithms that have been applied to intrusion detection (Yasin et al., 2021). Another ML technique that has been shown to achieve high accuracy in detecting network intrusions is deep learning, which uses neural networks to learn complex patterns and relationships in data (Li et al., 2022).

To implement ML-based IDSs, several steps are typically involved, including data preprocessing, feature selection, model training, and model evaluation. Data preprocessing involves cleaning and preparing the data for analysis, while feature selection involves selecting the most relevant features to be used in the model. Model training involves feeding the preprocessed data into the ML algorithm to learn patterns and make predictions, while model evaluation involves assessing the performance of the model on unseen data using metrics such as accuracy, precision, recall, and F1-score (Balaji et al., 2022).

Despite the potential benefits of ML-based IDSs, there are also some challenges to their implementation. One of the primary challenges is the difficulty in acquiring large and diverse datasets for training and testing the models. Another challenge is the need for high computational power and resources to train and evaluate the models, especially for deep learning algorithms (Jain & Mishra, 2019).

3.9.1. Supervised Learning

Supervised learning is the task of inferring from labeled training data. The training data consists of training examples, with each example comprising an input object (usually a vector) and the desired output value (also known as the supervisory signal) (Alpaydin, 2010). The training data is analyzed by a supervised learning algorithm, which generates an output. This output is an inferred function that can be used to map new examples. An ideal situation will allow the algorithm to accurately establish the class labels for cases that haven't been seen, which requires that a learning algorithm will generalize from the training data to unknown scenarios in a "reasonable" time frame (Bishop, 2006).

"Regression" and "classification" problems are frequently associated with supervised learning challenges. In a regression problem, we aim to anticipate results within a continuous output, which means we try to map input variables to some continuous function. Instead, in a classification task, we're attempting to anticipate discrete output values. To put it another way, we're attempting to categorize input variables into distinct groups (James et al., 2013).
The most prevalent method for training neural networks and decision trees is supervised learning. Both methods relay heavily on the data provided by the specified categories. The classification is used to determine the network's error and then adjust the network to minimize it in the case of neural networks, and the classifications are used in the case of decision trees to determine which attributes provide the most information that can be used to solve the classification puzzle (Han et al., 2011).

3.9.2. Unsupervised Learning

The task of inferring a function from unlabeled data to characterize the underlying structure is known as unsupervised learning. The learner is given only unlabeled training data and is required to make predictions for all points that are not visible. Unsupervised learning allows us to tackle challenges with little or no prior knowledge of what we should accomplish our objectives. We can infer structure from data even if we don't know the variables' effects. This structure can be derived by clustering the data based on the relationships between the variables. There is no feedback from the prediction results with unsupervised learning (Alpaydin, 2010).

Unsupervised learning challenges include clustering and dimensionality reduction. Unsupervised learning methods such as the self-organizing map (SOM) and adaptive resonance theory (ART) are often utilized in neural network models (Hinton and Salakhutdinov, 2006). The fundamental disadvantage of supervised anomaly detection is the requirement to label the training data, which makes the process error-prone, costly, and time-consuming, as well as difficult to find new threats. Unsupervised anomaly detection tackles these difficulties by allowing online learning and boosting detection accuracy by allowing training on unlabeled data sets (Patcha and Park, 2007). When compared to supervised anomaly detection systems, unsupervised anomaly detection is relatively new (Chandola et al., 2009).

Overall, both supervised and unsupervised learning algorithms have been successfully applied in intrusion detection systems to detect and prevent cyber-attacks. Supervised learning algorithms are useful for detecting known attacks and require labeled data to train the model, while unsupervised learning algorithms are useful for detecting unknown attacks and can discover patterns and anomalies in unlabeled data. The choice of algorithm depends on the specific needs and requirements of the intrusion detection system.

3.10. Model selection preliminaries

A major issue with Intrusion Detection Systems is determining how to efficiently distinguish between normal and abnormal activities from many raw data variables. Also, how to properly implement automatic intrusion rules based on the network's raw data. Different data mining and machine learning approaches have been developed to analyze the information to solve such challenges, such as classification, grouping, association, and so on. Four of the most popular and commonly used algorithms from those strategies are covered in below.

The main goal for selecting a good Machin learning algorithm is to categorize fresh/unseen test samples appropriately. The purpose of model selection is to make this procedure easier. To put it another way, we transfer from one model (kernel) to another to improve generalization. The model we chose will determine how we respond to any training data, i.e., which classifier we use to predict future cases. As a result, model selection is inextricably linked to how we determine the "best fitting" classifier from a given model. After all, how effectively we generalize is determined by the best-fitting classifier.

Let Sn = { (x1, y1), . . . ,(xn , yn) } represents a our training set of n labeled datas. If we select model Fi then we get the best fitting discriminant function f ˆ i ∈ Fi by minimizing (Tommi Jaakkola 2006)

$$J = (\theta, \theta_o) = \sum_{t=1}^{n} loss(yt, f(x_t; \theta, \theta_o)) + \lambda_n \dots \|\theta\|^2 \qquad \text{eq 8.}$$

3.11. Machine learning Algorithms

3.11.1. Support Vector Machine

Support Vector Machines (SVMs) were first proposed by Vapnik in the 1960s and have recently gained much attention due to advances in methodology and theory, as well as applications to regression and density estimation. SVMs are a type of supervised learning model used for classification and regression analysis, capable of handling multiple continuous and categorical variables (Boser et al., 1992). The basis of SVMs is the concept of decision planes, which determine the decision boundaries by separating the data into different classes.

SVMs use a training algorithm to create a model that assigns new examples to one of two categories, making it a non-probabilistic binary linear classifier, given a set of training examples, each labeled as belonging to one of the two categories. SVM models represent instances as points in space, mapped in a way that separates the different categories by a large gap. New instances are then mapped into the same space and classified according to which side of the gap they fall on.

SVMs can perform non-linear classification by using the kernel approach, which implicitly maps the inputs to a high-dimensional space. The kernel approach allows SVMs to identify more complex decision boundaries that may not be possible with linear classifiers. In essence, SVMs attempt to find an optimal hyperplane that can separate the data into two distinct classes.

Overall, SVMs are a powerful and versatile machine learning tool that has been successfully applied in various domains, including image classification, natural language processing, and financial forecasting. (Cortes and Vapnik, 1995) (Suykens and Vandewalle, 1999) (Shawe-Taylor and Cristianini, 2004)

One helpful idea behind SVMs is the notion of the maximum margin, which refers to the largest separation between the two decision boundaries that still correctly classify all training examples. By maximizing the margin, SVMs aim to achieve good generalization performance and avoid overfitting the training data. (Burges, 1998)

As shown in Figure 9 there are several lines that provide a solution to the problem. Is one of them superior to the others, we may create a criteria to measure the merit of the lines intuitively: When a line passes too near to the points, it becomes noise sensitive and fails to generalize appropriately. As a result, our objective should be to discover the line that passes through all the points as far as feasible. The SVM method then operates by locating the hyperplane that provides the greatest minimum distance between the training instances. Within SVM's theory, this distance is given the crucial designation of margin twice. As a result, the best separating hyperplane maximizes the margin.

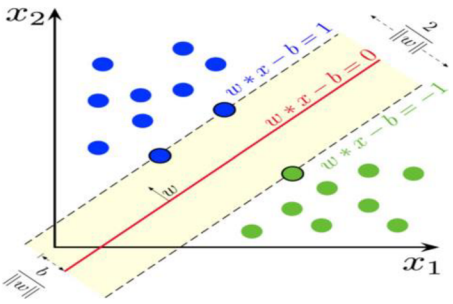


Figure 26: Separating Hyperplane Maximizes the Margin (Hosseinzadeh,2020)

It is a type of machine learning algorithm. Let D be a classification dataset with n points in a d-dimensional space D = {(xi, yi)}, with i = 1, 2, ..., n and yi is either +1 or -1. A hyperplane is a plane that has more than one axis (x) produces a d-dimensional linear discriminant function that divides the original space into two half spaces:

$$h(x) = wx + w1x1 + w2x2 + \cdots + wdxd + b \qquad \text{eq 9.}$$

Where w represents a d-dimensional weight vector, b represents a scalar bias, and x represents the training instances that are closest to the hyperplane. Support vectors are the training samples that are closest to the hyperplane in overall. The canonical hyperplane is the name given to this depiction. H(x) = 0 for points on the hyperplane defined by all locations with wx = -b.

If the dataset is linearly separated, a separating hyperplane may be identified such that h(x) 0 for all points marked -1 and h(x) > 0 for all points marked +1, according to (Cortes, C., & Vapnik, V., 1995). h(x) is a linear classifier or linear discriminant in this example, predicting the category for any point. Furthermore, because the weight vector w is perpendicular to the hyperplane, it gives the normal direction, but the bias b sets a hyperplane's offsets in d-dimensional space.

According to Cristianini and Shawe-Taylor (2000), in a linearly separable dataset, a separating hyperplane can be identified such that $h(x) > 0$ for all points labeled +1 and $h(x) < 0$ for all points labeled -1. The SVM algorithm attempts to maximize the margin, which is the distance between the hyperplane and the closest data points from both classes, while ensuring that no points lie on the wrong side of the hyperplane. To achieve this, the SVM solves a quadratic optimization problem that involves the Lagrange multipliers. After obtaining the multipliers, the SVM model can predict the class of new examples based on the sign of the discriminant function h(x).

3.11.2 The Random Forest Algorithm

Random Forest is a decision tree-based ensemble algorithm that combines the predictions of multiple decision trees. In Random Forest, each decision tree is trained on a random subset of the training data and a random subset of the input features. This helps to reduce the variance of the model and avoid over fitting.

Assuming an unknown joint distribution P(X,Y) for a p-dimensional random variable X = (X1, X2, ..., Xp) representing the input or predictor variables and a random variable Y representing the output variable, the goal of Random Forest is to learn a function f(X) that can accurately predict the value of Y. This is done by constructing an ensemble of decision trees, where each tree is trained on a randomly selected subset of the training data and a randomly selected subset of the input features.

The prediction function f(X) for a new input X is obtained by aggregating the predictions of all the individual decision trees. In regression problems, the aggregation function is usually the mean of the individual predictions, while in classification problems, the most common class predicted by the individual trees is selected.

The Random Forest algorithm is formally defined as follows:

Given a training set of n data points {(X1, Y1), (X2, Y2), ..., (Xn, Yn)} and a set of m input features {F1, F2, ..., Fm}, where $m << n$.

For each tree in the forest, randomly select a bootstrap sample of the training data of size n',

Where $n' < n$.

For each tree in the forest, randomly select a subset of the input features of size m',

Where m' < m.

Train each tree on the bootstrap sample and the subset of input features. For each input X, obtain the prediction f(X) by aggregating the predictions of all the individual decision trees. The prediction function f(X) is usually defined as the mean of the individual predictions for regression problems, and the most common class predicted by the individual trees for classification problems. The Random Forest algorithm can be further optimized by tuning the hyperparameters, such as the number of trees in the forest, the size of the bootstrap sample, and the size of the subset of input features. These hyperparameters can be optimized using techniques such as cross-validation.

The Random Forest algorithm is effective in a wide range of applications, including bioinformatics, image processing, and financial forecasting. It is also relatively easy to implement and computationally efficient.

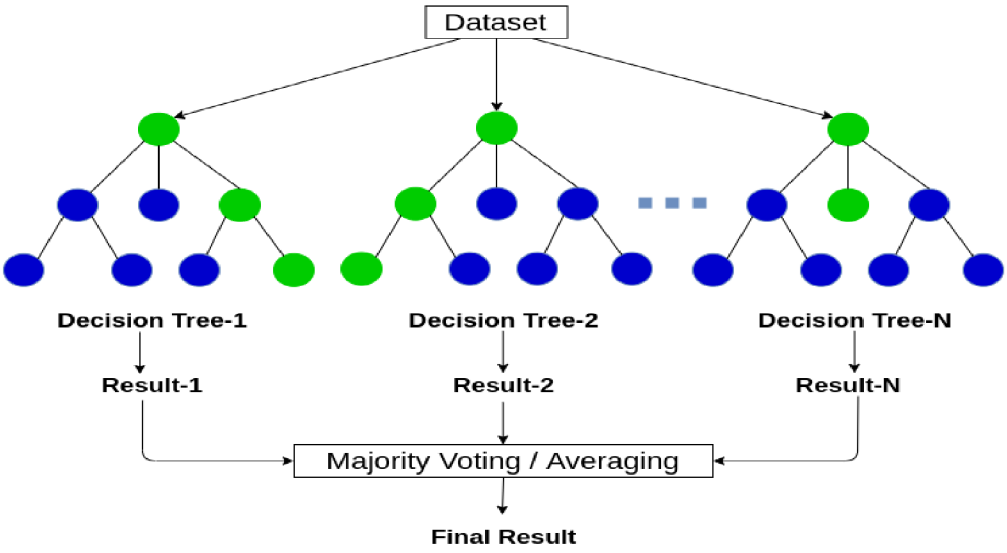


Figure 27: Random Forest Algorithm (Goyal C, 2022)

3.11.3. Logistic Regression

Logistic Regression is a statistical method used for analyzing a dataset in which there are one or more independent variables that determine an outcome (Cramer, 2002). The outcome is gaind with a dichotomous variable, which means it will have only two possible outcomes. Logistic Regression is used to predict the probability of the outcome occurring based on the given inputs.

The main formula for logistic regression is:

$$P(Y=1|X) = 1 / (1 + e^{-(\beta_0 + \beta_1X_1 + \beta_2X_2 + ... + \beta_nX_n)}) \quad \text{eq 10.}$$

Where:

$P(Y=1|X)$ represents the probability of the outcome (Y) occurring given the values of the independent variables (X).

β_0, β_1, β_2, ..., β_n are the coefficients that represent the impact of each independent variable on the probability of the outcome.

e is the base of the natural logarithm (approximately 2.71828).

In contrast to linear regression, which models the relationship between independent and dependent variables using a straight line, logistic regression models the relationship using a sigmoid (S-shaped) curve called the logistic function (Hosmer, Lemeshow, & Sturdivant, 2013).

The logistic function is known as the sigmoid function, is: $f(x) = 1 / (1 + e^{(-x)})$. A graph of the logistic function would show the S-shaped curve, where the x-axis represents the linear combination of independent variables ($\beta_0 + \beta_1X_1 + \beta_2X_2 + ... + \beta_nX_n$), and the y-axis represents the probability of the outcome ($P(Y=1|X)$).

The coefficients (β_0, β_1, β_2, ..., β_n) in the logistic regression formula are estimated using the maximum likelihood estimation method, which aims to find the values of the coefficients that maximize the likelihood of observing the given data (Agresti, 2002).

Therefore, logistic regression is a powerful statistical method used for predicting the probability of a binary outcome based on one or more independent variables. It uses the logistic function to

model the relationship between independent and dependent variables, and the coefficients are estimated using the maximum likelihood estimation method.

3.11.4. Decision tree

Decision trees are a widely used and interpretable machine learning technique for both classification and regression tasks (James et al., 2013). They recursively split the data into subsets based on the values of input features, with the goal of minimizing impurity (i.e., maximizing the homogeneity) within each subset (Quinlan, 1986).

To build a decision tree, various criteria can be used for selecting the best feature to split the data. Two common criteria are information gain and Gini impurity.

Information gain (IG) is based on the concept of entropy (Shannon, 1948), which measures the impurity of a dataset D:

$$Entropy(D) = -\sum(p(c) * \log2(p \quad \text{eq 11.}$$

Here, p(c) is the proportion of samples belonging to class c in dataset D. Information gain for a feature F is defined as:

$$IG(F) = Entropy(D) - \sum(|Dv| / |D| * Entropy(Dv)) \quad \text{eq 12.}$$

- Dv represents the subset of the dataset D with a specific value v of feature F, and $|Dv|$ and $|D|$ denote the cardinalities of Dv and D, respectively.
- Gini impurity (GI) measures the probability of misclassifying a randomly chosen element from dataset D if it were labeled based on the class distribution in D (Breiman et al., 1984):

$$GI(D) = 1 - \sum(p(c)^2) \quad \text{eq 13.}$$

- Gini impurity for a feature F is defined as:

$$GI(F) = \sum(|Dv| / |D| * GI(Dv)) \quad \text{eq 14.}$$

The decision tree algorithm can be outlined as follows:

Select the best feature to split the data: Evaluate information gain or Gini impurity for each feature and choose the feature that maximizes information gain or minimizes Gini impurity (James et al., 2013).

Split the dataset based on the selected feature: Create child nodes for each possible value of the feature and assign the corresponding data points to each child node.

Recursively repeat steps 1 and 2 for each child node: Continue to split the data in each child node until a stopping criterion is met (e.g., a maximum depth is reached, or the remaining data points are all from the same class).

Assign a class label or regression value to each leaf node: For classification tasks, the most common class label in the leaf node is assigned, while for regression tasks, the mean target value of the data points in the leaf node is assigned (James et al., 2013).

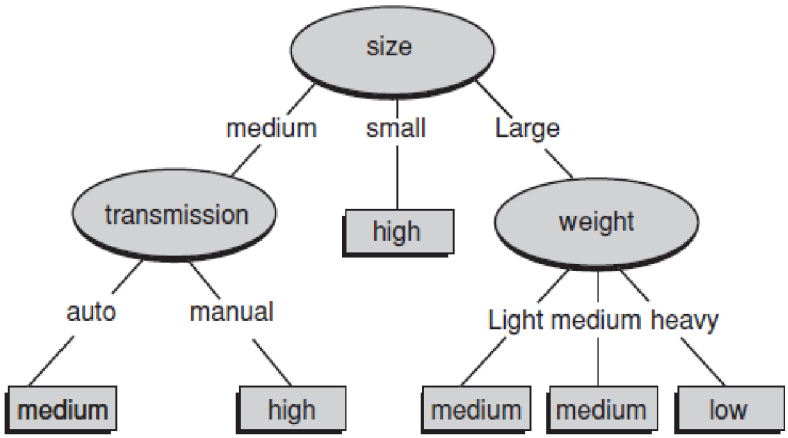


Figure 28: Example of Simple Decision Tree (Sumathi and Sivanandam, 2006)

Decision trees offer several advantages, including interpretability, handling of mixed data types, and minimal preprocessing requirements. However, they are also be prone to overfitting and instability. Overfitting can be mitigated by pruning or limiting the depth of the tree, while instability can be addressed using ensemble methods, such as random forests or boosting (Breiman, 2001; Schapire, 2003).

4: Practical part

4.1. Implementation Tools

Programming languages:

Python: A widely used programming language for data analysis and machine learning with extensive libraries and support.

Integrated Development Environments (IDEs):

Jupiter Notebook: An interactive web-based environment for Python, R, and other languages, allowing you to combine code, text, and visualizations in a single document.

Visual Studio Code: A versatile and powerful code editor with support for Python, R, and many other languages, as well as extensions for data science and machine learning tasks.

Data manipulation and analysis libraries:

Pandas (Python): A library for data manipulation and analysis, providing data structures like DataFrames and Series, as well as functions for data cleaning, aggregation, and transformation.

NumPy (Python): A library for numerical computing in Python, offering support for arrays, linear algebra, and mathematical functions.

Data visualization libraries:

Matplotlib (Python): A library for creating static, animated, and interactive visualizations in Python.

Seaborn (Python): A library based on Matplotlib for creating statistical graphics in Python, offering a higher-level interface for statistical graphics.

Machine learning and deep learning libraries:

Scikit-learn (Python): A comprehensive library for machine learning in Python, offering a wide range of algorithms for classification, regression, clustering, and dimensionality reduction.

Random Forest (R): A library for creating random forest models for classification and regression tasks in R.

4.3 Data source and description

The dataset used in this master's thesis was obtained from the link place below. (Data source: https://www.unb.ca/cic/datasets/nsl.html.The NSL-KDD intrusion detection dataset has been widely employed for training and evaluating models in the domain of network security, as it allows for the comparison of various approaches. The dataset is based on a real-world military network, which includes three "target" workstations running diverse operating systems and services. Three additional workstations generate traffic by simulating different IP addresses. A sniffer records all network communication using the TCP dump format. NSL-KDD is composed of compressed raw (binary) TCP dump data collected over a period of seven weeks, which can be converted into approximately five million training connections (Lavallee et al., 2009).

The model is typically trained on the training portion of the dataset and then tested on the test portion to assess its performance. Each training dataset consists of 41 features and is labeled as either normal or a specific type of threat. A reduced version of the dataset is used for memory-constrained machine learning algorithms, containing around 67352 normal individual connection vectors and 58630 attack. The training dataset is composed of 53% normal connections and 47% attack connections.

- 53% normal connections
- 47% attack connections

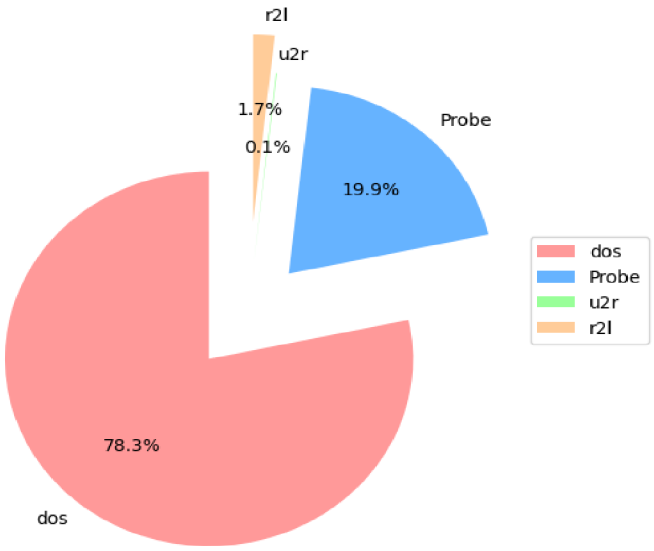


Figure 29: Distribution of Attack Data Only. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

dos: Denial of Service, **probe**: Probe, **u2r**: Privilege Escalation, **r2l**: Remote Access attacks.

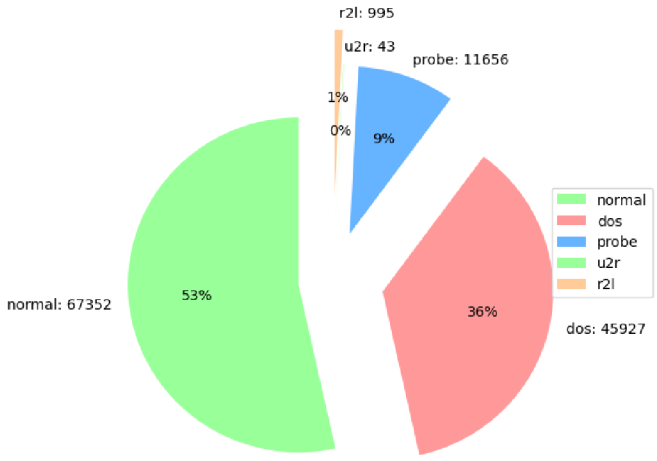


Figure 30: Distribution of full data. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

dos: Denial of Service, **probe**: Probe, **u2r**: Privilege Escalation, **r2l**: Remote Access attacks.

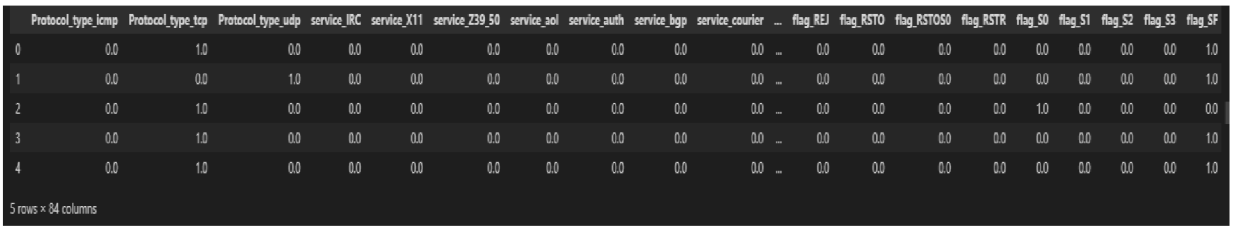
4.4. Data Pre-Processing

4.4.1. Mapping Normal as 0 and Attack as 1 (Encoding)

Before training the ML models, the dataset undergoes a pre-processing stage to ensure optimal model performance.

- Protocol type: This feature represents the type of protocol used in the network connection. There are three possible values: 'tcp', 'udp', and 'icmp'.
- Service: This feature represents the network service used in the connection, such as 'http', 'ftp', 'smtp', 'ssh', etc. There are a total of 70 different service types in the dataset.
- Flag: This feature represents the status of the connection, such as 'SF' (successful), 'S0' (connection attempt seen, no reply), 'REJ' (connection attempt rejected), etc. There are 11 possible flag values in the dataset.
- The main problem in terms of data organization in this dataset are there are three categorical futures and this categorical feature should be encoded to the numerical values and the result become binary vectors (1, 0, 0), (0, 1, 0), (0, 0, 1) So that this will be understandable by the model.

This One Hot Encoding enables the algorithms to process the data more efficiently and make accurate predictions. The One Hot Encoding uses class called LabelEncoder() from the sklearn.preprocessing module is used to perform this encoding.



	Protocol_type_icmp	Protocol_type_tcp	Protocol_type_udp	service_IRC	service_X11	service_Z39_50	service_aol	service_auth	service_bgp	service_courier	...	flag_REJ	flag_RSTO	flag_RSTOS0	flag_RSTR	flag_S0	flag_S1	flag_S2	flag_S3	flag_SF
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

5 rows × 84 columns

Figure 31: Distribution feature columns One Hot Encoded. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

Protocols and Occurrence of Attacks for Each Protocol is available in the plot below.

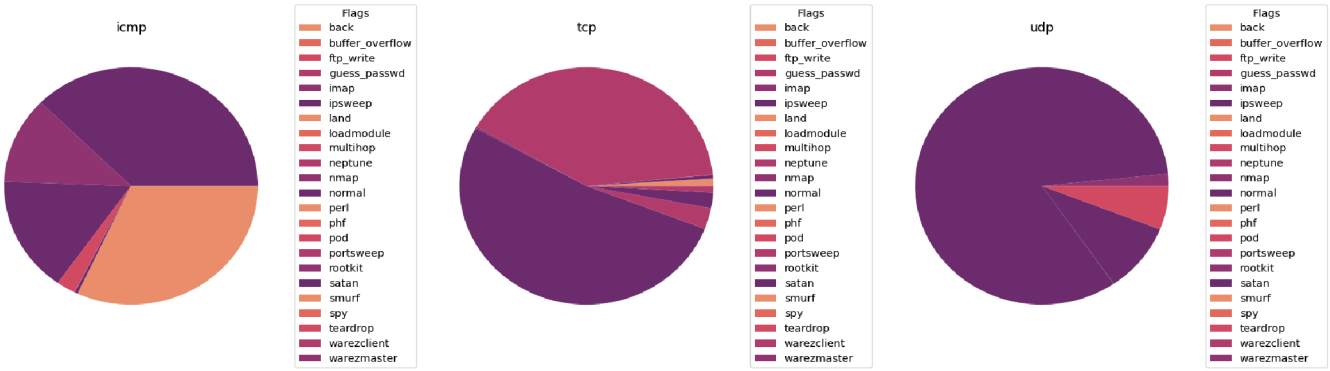


Figure 32: Protocols and Occurrence of Attacks. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

Icmp: Internet Control Message Protocol, **tcp**: Transmission Control Protocol, **udp**: User Datagram Protocol.

4.4.2. Classifying Attacks into Four Categories

The dataset consists of numerous attack types. To simplify the classification process, we group these attacks into four primary categories: Denial of Service (DoS), Probe, Privilege Escalation(U2R), and Remote Access(**R2L**). This approach enables a more manageable and focused analysis while still addressing the different types of cyberattacks.

Denial of Service (DoS) attacks: These attacks to disrupt the availability of a network or service by overwhelming it with a flood of requests or traffic. The list of future in this category are: apache2, back, land, neptune, mailbomb, pod, processtable, smurf, teardrop, udpstorm, worm.

Probe attacks (Probe): These attacks involve scanning and gathering information about a target network, often to identify vulnerabilities that can be exploited. The list of futures in this category are ipsweep, mscan, nmap, portsweep, saint, satan.

Privilege Escalation attacks (U2R) these attacks exploit vulnerabilities in a system to gain unauthorized access to resources or elevated privileges. The list of futures in this category are buffer_overflow, loadmdoule, perl, ps, rootkit, sqlattack, xterm.

Remote Access attacks (R2L): These attacks focus on gaining unauthorized access to a cyber often to steal sensitive information or perform malicious actions. The list of futures in this category

are ftp_write, guess_passwd, http_tunnel, imap, multihop, named, phf, sendmail, snmpgetattack, snmpguess, spy, warezclient, warezmaster, xclock, xsnoop. Hence the attack labels will be - DOS, Probe, Privilege, Access, Normal The data was farther encoded the attack types (DoS, Probe, U2R, R2L) with numerical labels (1, 2, 3, 4) this numbers are an arbitrary choice made to represent the categorical attack types as numerical values. Machine learning algorithms typically require numerical inputs, and assigning a unique numerical label to each attack type allows the algorithm to process the data.

The numbers assigned to the attack types do not have any inherent meaning or order; they simply serve to represent the different attack categories. The choice of numbers (1, 2, 3, 4) is arbitrary, and other numerical representations (such as 0, 1, 2, 3) or different encoding techniques (e.g., one-hot encoding) could also be used. The main point to use it is to ensure that the encoding is consistent across the dataset and is properly interpreted by the machine learning algorithm.

4.4.3. Data Scaling

Before training models, it is crucial to standardize the features of the dataset to ensure that all features have the same scale. This process helps improve the performance of the models and allows them to converge faster during training. In this study, Standard Scaler () function from the scikit-learn preprocessing library is used to scale the features of both the training and testing data for each of the four categories: DoS, Probe, R2L, and U2R.

For each category, Standard Scaler instance on the training data was fitted and apply the same transformation to both the training and testing data. This approach ensures that the scaling process is consistent between the training and testing sets, which is important for obtaining reliable model performance results.

4.5 Exploratory Data Analysis

4.5.1 Data Profiling and Visualization

Exploring the Attack Column in Detail

Training **Test**

```
normal                67343
neptune               41214
satan                  3633
ipsweep                3599
portsweep              2931
smurf                  2646
nmap                   1493
back                    956
teardrop                892
warezclient             890
pod                     201
guess_passwd             53
buffer_overflow          30
warezmaster              20
land                     18
imap                     11
rootkit                  10
loadmodule                9
ftp_write                 8
multihop                  7
phf                       4
perl                      3
spy                       2
Name: attack, dtype: int64
```

```
normal                9711
neptune               4657
guess_passwd          1231
mscan                  996
warezmaster            944
apache2                737
satan                  735
processtable           685
smurf                  665
back                   359
snmpguess              331
saint                  319
mailbomb               293
snmpgetattack          178
portsweep              157
ipsweep                141
httptunnel             133
nmap                    73
pod                     41
buffer_overflow         20
multihop                18
named                   17
ps                      15
sendmail                14
rootkit                 13
xterm                   13
teardrop                12
xlock                    9
land                     7
xsnoop                   4
ftp_write                3
worm                     2
loadmodule               2
perl                     2
sqlattack                2
udpstorm                 2
phf                      2
imap                     1
Name: attack, dtype: int64
```

Figure 33: Exploring Distribution of Training and Test set. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

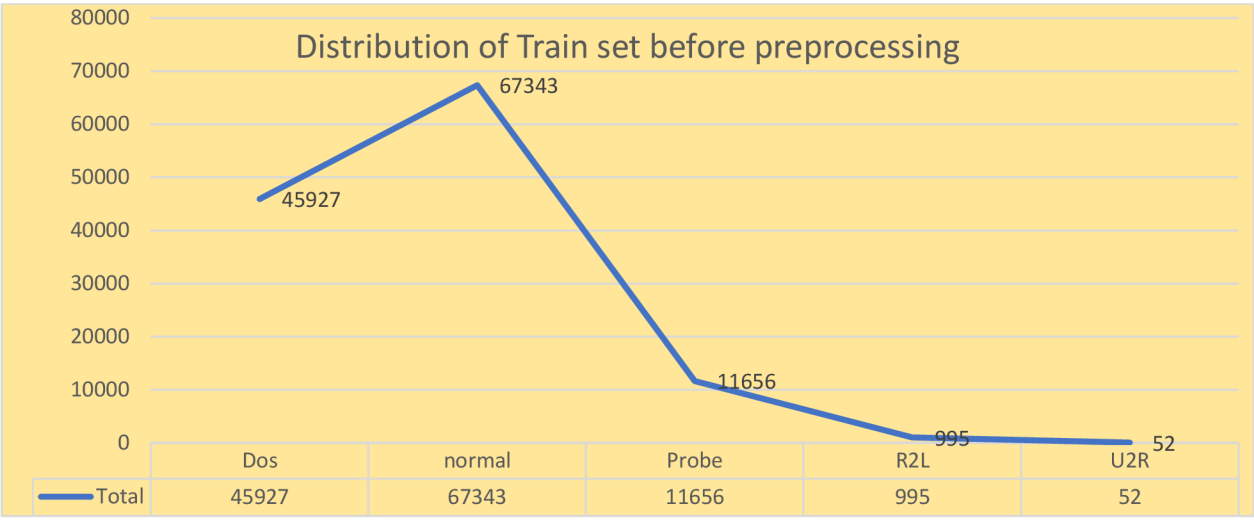


Figure 34: Attack Distribution of NSL-KDD Dataset. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

Dos: Denial of Service, **probe**: Probe, **U2R**: Privilege Escalation, **R2L**: Remote Access attacks.

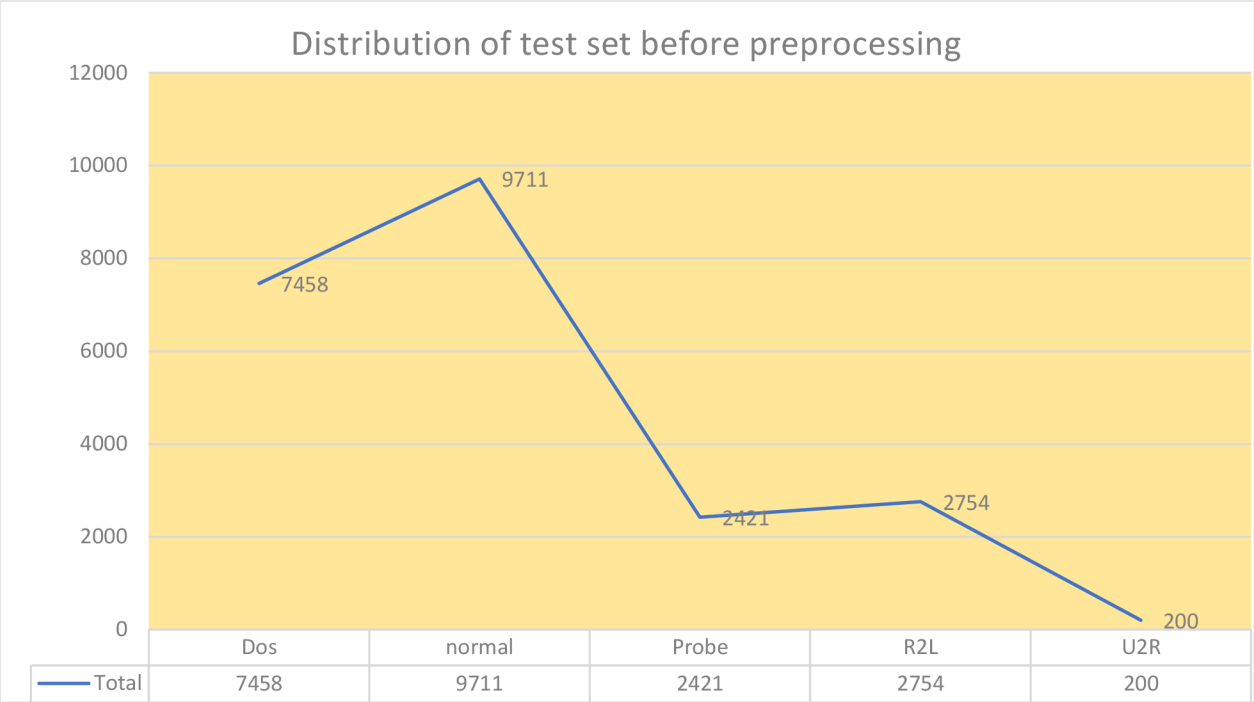


Figure 35: Attack distribution of NSL-KDD Dataset. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

Dos: Denial of Service, **probe**: Probe, **U2R**: Privilege Escalation, **R2L**: Remote Access attacks.

As shown in the distribution there is a high number of unbalanced data and class size. This unbalanced situation can lead to classification mistakes, with the model skewed toward the majority class. To counteract the detrimental effects of training on an unbalanced dataset in the next more steps explain them in detail.

Identifying and changing Nominal data into binary data

This stage basically involves cleaning the dataset to remove duplicate entries; however, because the NSL-KDD dataset has already been cleaned, this step is no longer necessary. Because the dataset comprises both numerical and non-numerical occurrences, a pre-processing procedure must be performed.

There are 3 groups which have a nominal value and very hard to train the model these categories are shown in the figure below.

Categorical features

Training set:

Feature 'protocol_type' has 3 categories.

Feature 'service' has 70 categories.

Feature 'flag' has 11 categories.

Feature 'label' has 23 categories.

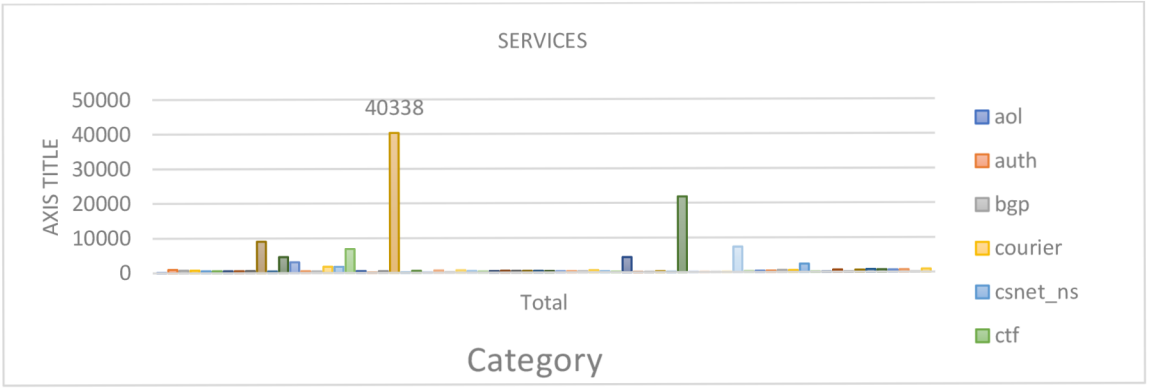


Figure 36: Service. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

Dos: Denial of Service, **probe**: Probe, **U2R**: Privilege Escalation, **R2L**: Remote Access attacks.

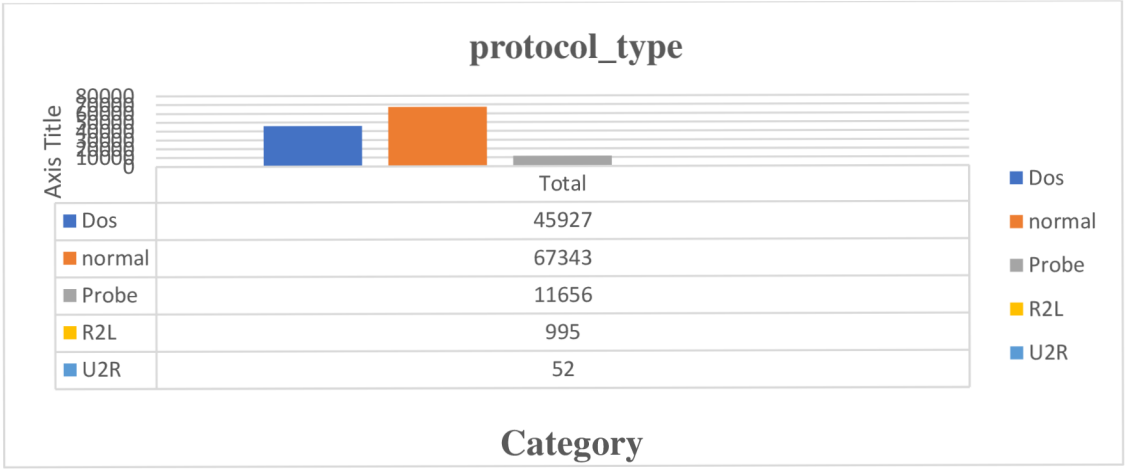


Figure 37: Protocol Type. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

Dos: Denial of Service, **probe**: Probe, **U2R**: Privilege Escalation, **R2L**: Remote Access attacks.

One hot encoding

Because the estimator (classifier) defined in scikit-learn works well with numerical inputs, the conversion of the categorical data above is changed to numerical values as shown below in the image using one-hot encoding approach. This method converts each category feature with m potential inputs into n binary features, one of which is active at any given moment.

According to the analysis on the training set four categorical futures were found as listed below.

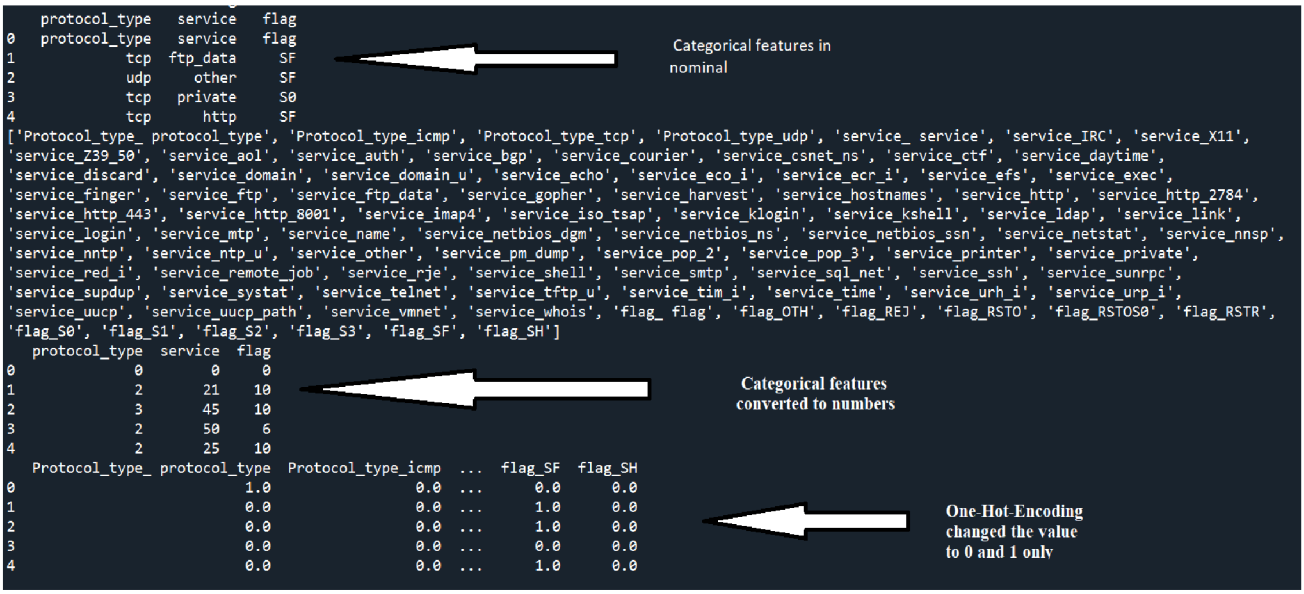


Figure 38: One hot Encoding. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

4.6 Feature Selection Using Recursive Feature Elimination (RFE)

Feature selection plays an important role in building effective machine learning models, as it helps in identifying the most relevant features for the task and reduces the computational complexity of the models. Recursive Feature Elimination (RFE) is used as a feature selection technique.

RFE is a recursive process that works by fitting a model, ranking the features based on their importance, and removing the least important features one by one. This process is repeated until a desired number of features are selected. RFE is particularly useful in identifying the most relevant features that contribute to the classification task and can help improve the performance of the models.

To implement RFE, RFE function from the scikit-learn feature selection library. RFE is applied to each of the four categories: DoS, Probe, R2L, and U2R, to select the most relevant features for each category. By doing so, it can be ensured that the machine learning models are trained on the most informative features, leading to more accurate and efficient classification results.

Future reduction is carried out to eliminate useless, redundant, or noisy futures in the dataset. In this feature selection stage, the futures that are more relevant is filtered out. The main reasons for limiting the number of features is to remove the irrelevant characteristics that indicate connections between features and target classes that are purely coincidental and do not accurately reflect the situation.

As a best practice recursive feature elimination (RFE) perfectly implemented the procedure and got a satisfactory result the sample is done in the decision tree model.
During a recursive feature elimination (RFE) with the number of features passed as a parameter to identify the features achieved is sufficient number of features." This might imply that RFE is simply used to acquire the best features, or that it is also utilized to achieve the rank. It can be only utilized 13 of the 42 best futures from the dataset. In the RFE procedure, each attack classes have different amounts of features for each attack category: 13 for DoS, 13 for Probe, 13 for R2L, and 13 for U2R.
.

5. Results and Discussion

This section discusses the experiments conducted as part of this study and the results obtained. The performance of various machine learning models is also examined in this chapter. Throughout the entire process, the NSL-KDD dataset was used to develop a model, as outlined in the previous chapter. It was observed that some models perform better against specific attack classes, while their performance declines in other attack classes based on the evaluation findings. Machine learning algorithms known to be effective against attack classes were trained and assessed using the RFE method. To improve the results of several machine learning algorithms, meticulously pre-processed the data before fitting it into the model is needed.

To further enhance the outcomes of various machine learning algorithms, should be carefully pre-processed the data before feeding it into the model. In this research, WEKA, an open source is software is used to do the exploratory analysis and some preprocessing tasks, but the model training is performed by Python so the output that you see here is python generated.

In this chapter of the thesis, we will be evaluating the effectiveness of four different predictive models, namely logistic regression, decision tree, support vector machine, and random forest, in predicting the detecting the attack group. The result will be presented in the form of a confusion matrix and a receiver operating characteristics (ROC) curve for each model, with the accuracy, precision, recall, F1 score, and ROC being presented in a table for comparison. These metrics will be used to rank the predictive power of each model.

The confusion matrix provides us with a visual representation of the performance of each model by showing the number of true positives, true negatives, false positives, and false negatives. The ROC curve, on the other hand, provides a graphical representation of the trade-off between the true positive rate and the false positive rate at different classification thresholds. By comparing these metrics across the four different models, we can gain a deeper understanding of their respective strengths and weaknesses, and ultimately determine which model is most effective for predicting which attack group is categorized correctly.

The modeling process, the foundational theory and the step-by-step execution of the overall approaches is covered in the methodology and practical section.

5.1 Decision tree

Based on the results of the model, it has achieved an impressive performance on both the training set and the test set, with scores of 0.94 and 0.95 respectively. This indicates that the model can generalize well to unseen data.

Looking at the classification report, we see that the model has performed extremely well in identifying normal traffic, with a precision of 0.99, recall of 0.95, and an F1 score of 0.97. This suggests that the model can accurately identify normal traffic with a high degree of confidence.

For the other attack groups, the model has also performed well, with a precision ranging from 0.67 to 0.95 and a recall ranging from 0.67 to 0.97. This indicates that the model can accurately identify most of the different types of cyber-attacks in the network traffic.

However, for the rare attack groups such as U2R and R2L, the model has low precision and recall, which suggests that more data may be required to improve the model's performance in detecting these types of attacks. This indicates that the model may be under fitting for these rare attack types.

To address this issue, additional data specifically related to these rare attack types may be needed to improve the model's ability to accurately detect them. Alternatively, more complex machine learning models or different feature engineering techniques could be explored to improve the model's performance for these rare attack types.

Model overview

In the built model clf **= tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=5)** the criterion parameter specifies the quality of the split using the Gini impurity measure. Gini impurity is a measure of the probability of incorrectly classifying a randomly chosen element in the dataset if it were randomly labeled according to the distribution of labels in the subset.

Where (pi) is the probability of class (i) in the subset, and (c) is the total number of classes.

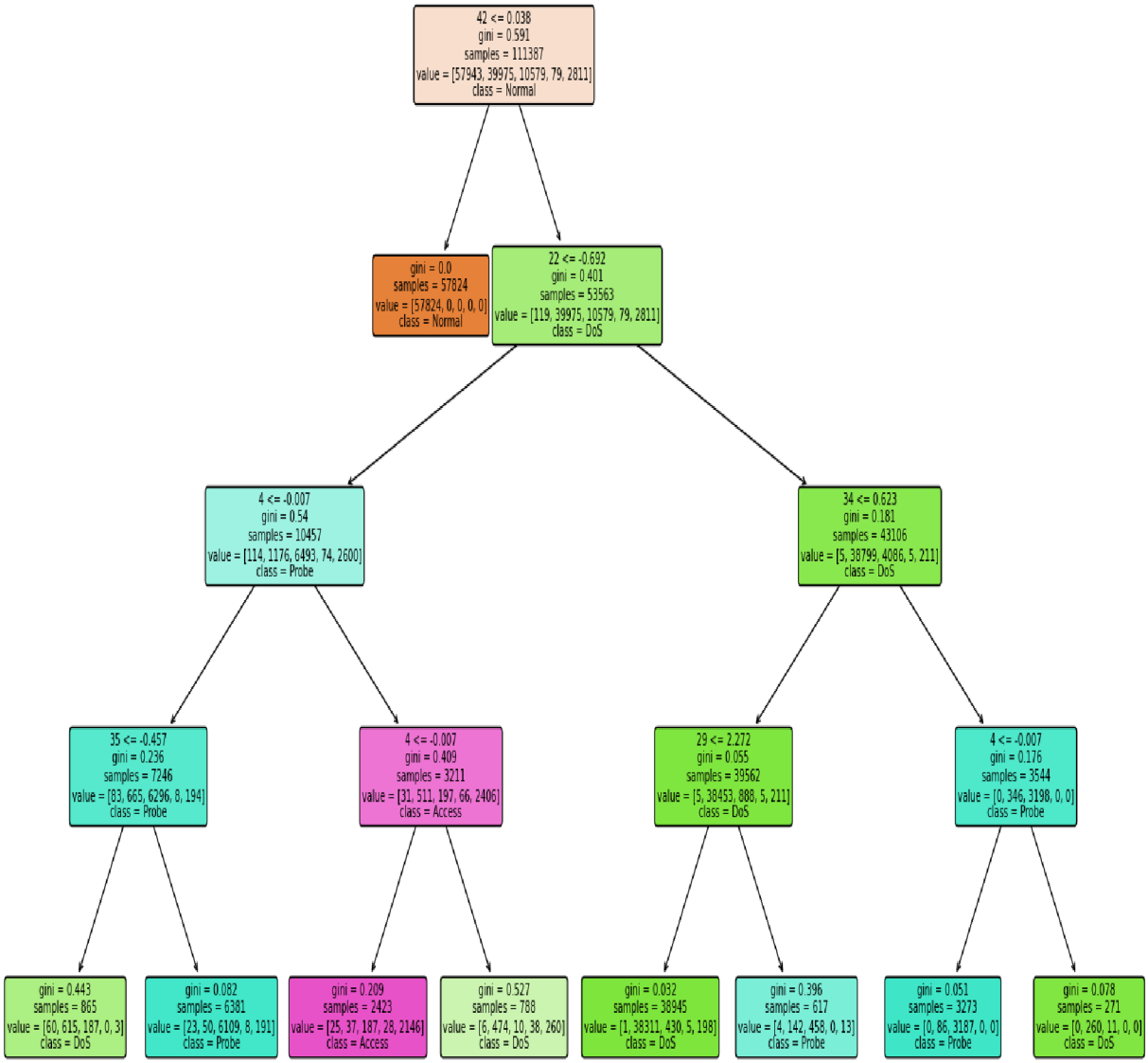


Figure 39: Decision tree. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

In the Table IV and Figure 40 displays the top thirteen features with the highest feature importance in the decision tree model, which were identified by recursive future elimination mechanism.

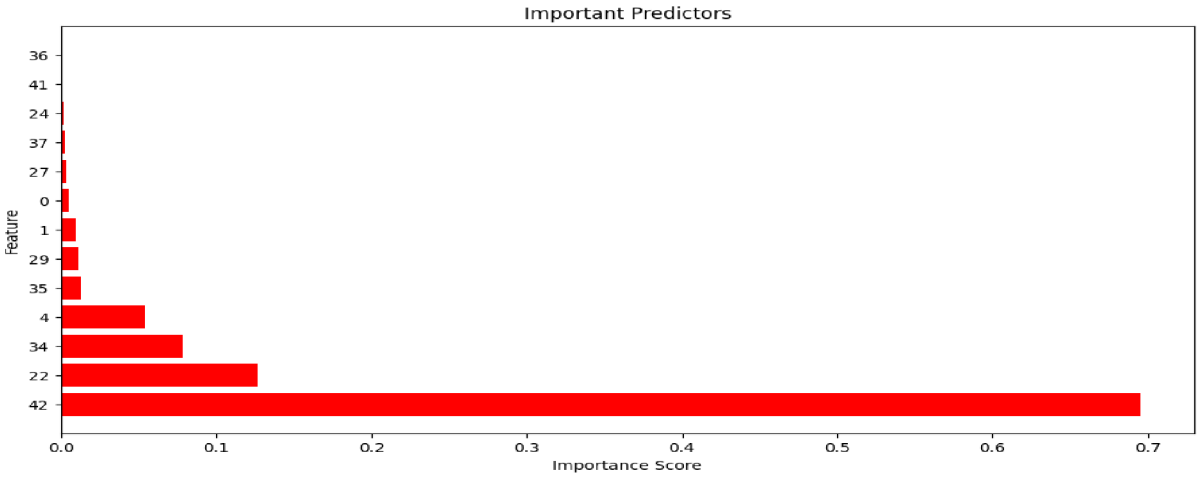


Figure 40: List of Important predictors. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

In the above figure 40 the name of important predictors by rank is listed below

Feature code	Feature Name	Feature Importance score
42	is_guest_login	0.6951
22	dst_host_same_src_port_rate	0.1262
34	flag	0.0784
4	dst_host_diff_srv_rate	0.0541
35	same_srv_rate	0.0125
29	duration	0.0105
1	srv_rerror_rate	0.0092
0	rerror_rate	0.0045
27	dst_host_srv_diff_host_rate	0.0035
37	srv_count	0.0025
24	dst_host_srv_rerror_rate	0.0017
41	src_bytes	0.0010
5	dst_host_same_src_port_rate	0.0002

Table 2: Selected features as important predictors. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own table.

The decision tree output shows the feature code, feature name, and feature importance score for each feature used in the tree. The feature code is simply a numerical identifier for each feature. The feature importance score is a measure of how much each feature contributes to the decision made by the tree. It is calculated as the total reduction of the impurity measure (such as Gini impurity or entropy) that is achieved by splitting the data on that feature. The higher the feature importance score, the more important the feature is in making decisions in the tree.

In this specific output, the most important feature is feature 42, which corresponds to the "is_guest_login" feature. This feature has a feature importance score of 0.6951, meaning it is highly influential in the decision making of the tree. The next most important feature is feature 22, which corresponds to the "dst_host_same_src_port_rate" feature, with a feature importance score of 0.1262. Features 34, 4, 35, and 29 also have non-zero feature importance scores and contribute to the decision making of the tree.

Overall model performance is evaluation.

```
Train score: 0.9439521667699103
Test score: 0.9464314570428225
Classification report:
              precision    recall  f1-score   support

           0       0.99      0.95      0.97     19264
           1       0.95      0.97      0.96     13412
           2       0.80      0.92      0.86      3498
           3       0.00      0.00      0.00        29
           4       0.67      0.67      0.67       927

    accuracy                           0.95     37130
   macro avg       0.68      0.70      0.69     37130
weighted avg       0.95      0.95      0.95     37130
```

Figure 41: Overall model performance Decision tree. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

Overall, the model has achieved a weighted average F1 score of 0.95, which indicates that it is a strong performer in predicting the different types of cyber-attacks in the network traffic. However, there is still room for improvement in detecting rare attack types such as U2R and R2L, which may require further investigation and data collection but for the sake of test followed a complex and attack group specific future engineering based on Recursive future Elimination the feature for each attack type as show in the table 6 and the dataset spited in to four places based on the and trained them individually and the test result was promising by detecting even the class types with less data on it .

Cross validation accuracy: A cross-validation score of 93.8% means that the model has a high degree of accuracy in predicting the target variable across different folds of the dataset, indicating that it is likely to generalize well to new, unseen data. It also suggests that the model is not overfitting to the training data, as the performance is consistent across different splits of the data.

Confusion matrix

The rows represent the true class labels, while the columns represent the predicted class labels. For example, in the first row and first column, we can see that there were 18,929 instances where the true label was Normal, and the model correctly predicted it as Normal (true positive). In the second row and first column, we can see that there were 307 instances where the true label was DoS but the model predicted it as Normal (false negative).

Overall, the confusion matrix shows us how well the model is performing for each class. We can see that the model is performing well for the Normal and DoS classes, with high numbers of true positives and low numbers of false positives and false negatives. However, for the Probe, R2L, and U2R classes, the model has higher numbers of false positives and false negatives, indicating that it is not performing as well for these classes.

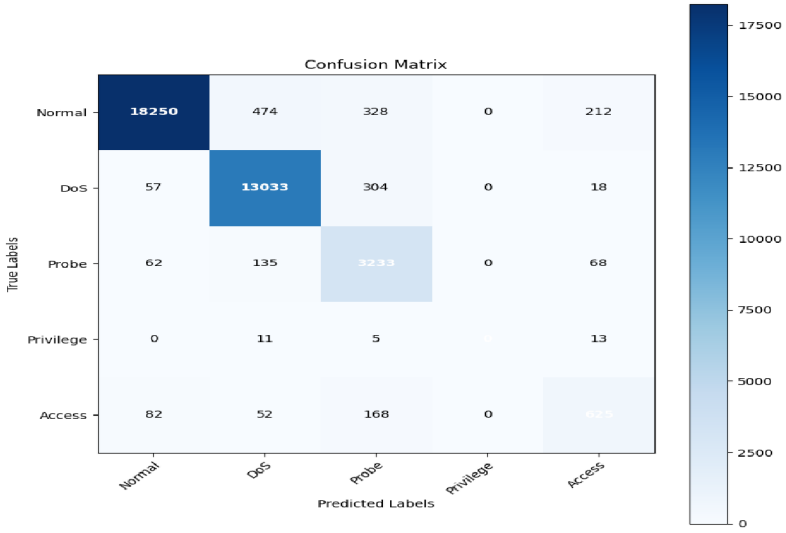


Figure 42: confusion matrix of Decision tree. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

5.2. Logistic regression

The output of feature selection shows the top 13 features with their respective coefficients obtained from logistic regression.

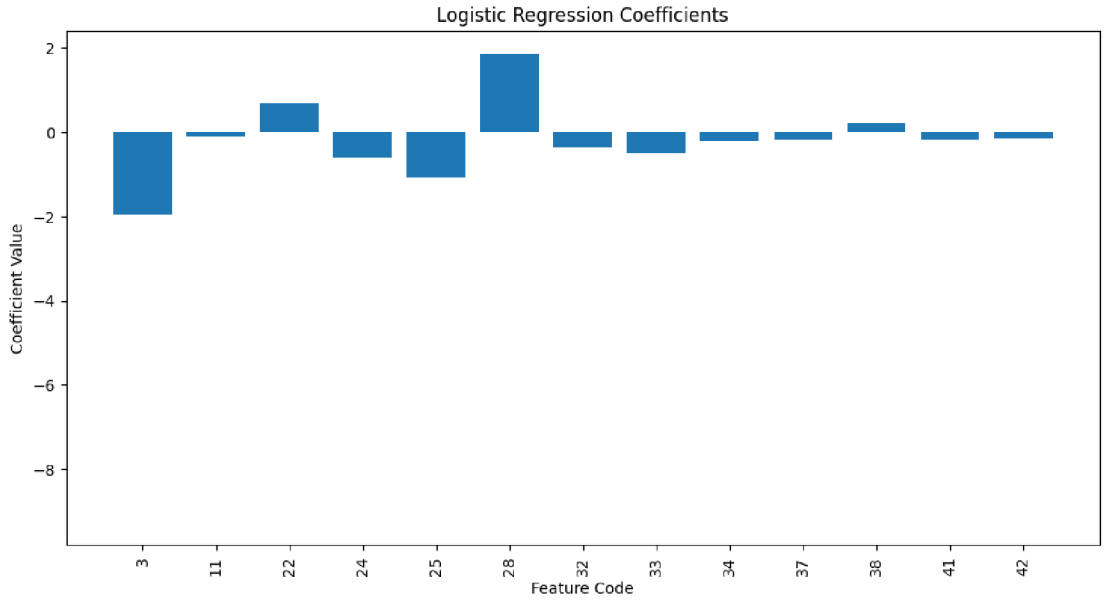


Figure 43: Logistic regression Coefficient. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

The feature with the highest negative coefficient is the Root shell (feature 3), which suggests that this feature has the most negative impact on the classification of the data. The feature with the highest positive coefficient is the dst_host_same_src_port_rate (feature 22), which suggests that this feature has the most positive impact on the classification of the data.

The negative coefficients for some of the features suggest that these features are negatively correlated with the target variable, they are less likely to be associated with a particular attack type. Conversely, the positive coefficients suggest that these features are positively correlated with the target variable and are more likely to be associated with a particular attack type.

Overall, the coefficients provide insights into the importance of each feature in the classification of attacks. These insights can be used to refine the feature selection process or to gain a deeper understanding of the data.

Feature code	Feature Name	Feature Importance score
42	is_guest_login	0.6951
22	dst_host_same_src_port_rate	0.1262
34	flag	0.0784
4	dst_host_diff_srv_rate	0.0541
35	same_srv_rate	0.0125
29	duration	0.0105
1	srv_rerror_rate	0.0092
0	rerror_rate	0.0045
27	dst_host_srv_diff_host_rate	0.0035
37	srv_count	0.0025
24	dst_host_srv_rerror_rate	0.0017
41	src_bytes	0.0010
5	dst_host_same_src_port_rate	0.0002

Table 3: Logistic regression Coefficient (Own table)

Performance evaluation

Logistic Regression model has performed well on both the training and test sets with scores of 0.97 and 0.97, respectively. The classification report shows that the model has high precision, recall, and F1 scores for all attack groups, except for U2R which has a relatively lower score.

The model has achieved a high accuracy of 0.97, indicating that it is a strong performer in predicting the different types of cyber-attacks in the network traffic. However, when compared to the previous model, the Logistic Regression model has a lower precision and recall for the rare attack groups U2R and R2L. This suggests that more data may be required to improve the model's performance in detecting these types of attacks.

```
...   Train score: 0.9660822178530708
      Test score: 0.9659843792081875
      Classification report:
                    precision    recall  f1-score   support

                 0       0.97      0.98      0.97     19264
                 1       0.98      0.99      0.99     13412
                 2       0.92      0.89      0.90      3498
                 3       0.55      0.21      0.30        29
                 4       0.78      0.74      0.76       927

          accuracy                           0.97     37130
         macro avg       0.84      0.76      0.78     37130
      weighted avg       0.97      0.97      0.97     37130
```

Figure 44: Performance evaluation. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

Overall, the Logistic Regression model is a strong performer and provides a reliable means of detecting and classifying different types of attacks in network traffic data.

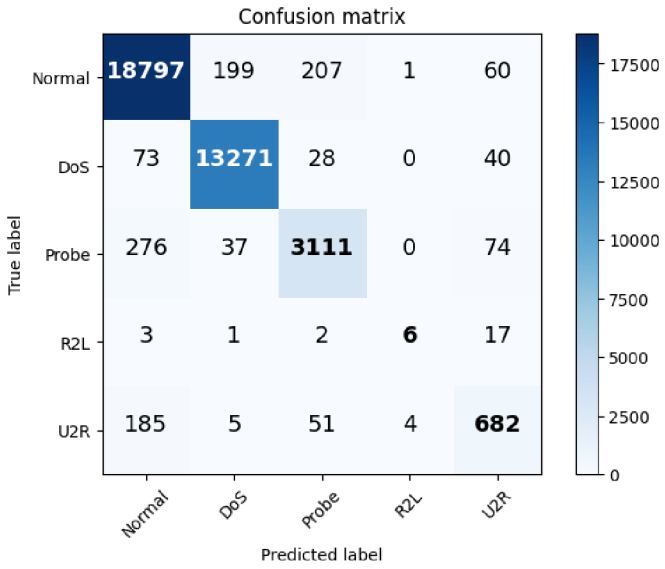


Figure 45: Confusion matrix of Logistic Regression. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

In confusion matrix for the Logistic Regression model, we can see that the majority of the data points are correctly classified, as the diagonal values (top left to bottom right) are higher than the off-diagonal values. The model has correctly classified 18797 instances of Normal traffic, 13271

instances of DoS attacks, and 3111 instances of Probe attacks. However, the model has more difficulty correctly classifying the rare attack groups of R2L and U2R, as shown by the low values in their corresponding diagonal cells.

For example, the model only correctly predicted 6 out of 29 R2L attacks, which results in a low recall value of 0.21 for R2L attacks. Similarly, the model only correctly predicted 682 out of 945 U2R attacks, which results in a low recall value of 0.72 for U2R attacks.

Cross validation accuracy: A cross-validation score of 95.8% means that the model has a high degree of accuracy in predicting the target variable across different folds of the dataset, indicating that it is likely to generalize well to new, unseen data. It also suggests that the model is not overfitting to the training data, as the performance is consistent across different splits of the data.

Overall, the confusion matrix can provide valuable insights into the performance of the classification model and help identify areas where the model may need improvement.

5.3. Support Vector Machine

Future significance

The list shows the scores of each feature after using Recursive Feature Elimination (RFE) to select the top 13 most significant features. The higher the score, the more significant the feature is in classifying the data.

Some features have less values and some have very high scores indicating they are the most important features for classification. For example, feature 33 has the highest score, which suggests that this feature is the most significant in distinguishing between the different classes of data. On the other hand, feature 11 has a score of very less and may not contribute much to the classification.

Controlling the regularization parameter C: to controls the trade-off between achieving a low training error and a low testing error helped to train this model very well. As putting c value to optimum C helped to create strong complex decision boundary that can capture all the nuances in the data that can lead to a decrease in accuracy.

And gamma parameter: The gamma parameter handled the width of the radial basis function kernel. A high value of gamma results in a highly non-linear decision boundary that may overfit the data.

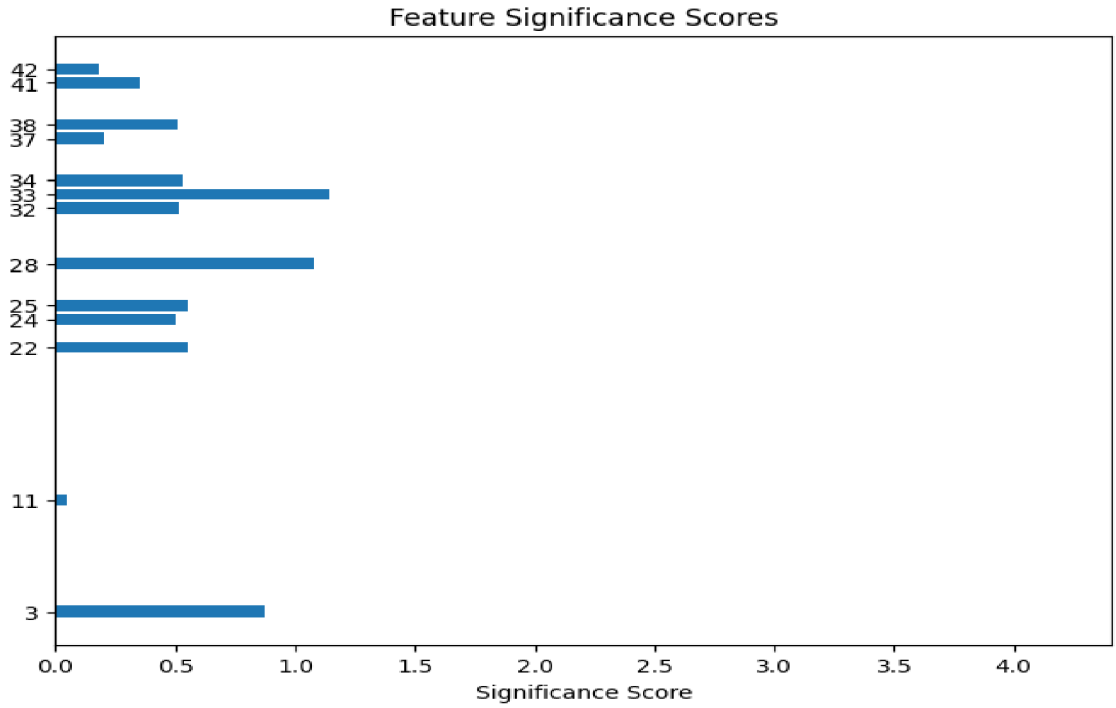


Figure 46: Feature significance score in support vector machine. (Own work)

Model performance

```
...   Accuracy: 0.9966065176407218
                precision    recall  f1-score   support

             0       1.00      1.00      1.00     19264
             1       1.00      1.00      1.00     13412
             2       0.99      0.99      0.99      3498
             3       0.71      0.34      0.47        29
             4       0.95      0.97      0.96       927

      accuracy                           1.00     37130
     macro avg       0.93      0.86      0.88     37130
  weighted avg       1.00      1.00      1.00     37130
```

Figure 47 :Classification report of SVM. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

The SVM model achieved an accuracy of 0.9966 on the test set, which means it correctly classified 99.66% of the instances. The precision, recall, and F1-score for each class are as follows:

For the Normal class, the SVM model achieved a precision, recall, and F1-score of 1.0, which indicates that it correctly classified all instances of normal traffic with no false positives or false negatives.

- For the DoS class, the SVM model achieved a precision of 1.0, which means that all instances classified as DoS were actually DoS attacks. The recall is also 1.0, which indicates that the model correctly identified all instances of DoS attacks. The F1-score is also 1.0.
- For the Probe class, the SVM model achieved a precision of 0.99, which means that 99% of instances classified as Probe were actually Probe attacks. The recall is 0.99, which indicates that the model correctly identified 99% of instances of Probe attacks. The F1-score is also 0.99.
- For the R2L class, the SVM model achieved a precision of 0.71, which means that only 71% of instances classified as R2L were actually R2L attacks. The recall is 0.34, which indicates that the model missed 66% of instances of R2L attacks. The F1-score is 0.47.
- For the U2R class, the SVM model achieved a precision of 0.95, which means that 95% of instances classified as U2R were actually U2R attacks. The recall is 0.97, which indicates that the model correctly identified 97% of instances of U2R attacks. The F1-score is 0.96.
- **Cross validation accuracy**: A cross-validation score of 98.2% means that the model has a high degree of accuracy in predicting the target variable across different folds of the dataset, indicating that it is likely to generalize well to new, unseen data. It also suggests that the model is not overfitting to the training data, as the performance is consistent across different splits of the data.

Compared to the previous models, the SVM model achieved a higher accuracy and F1-score for all classes except for the R2L class. However, it should be noted that the SVM model took significantly longer to train compared to the previous models due to the complexity of the algorithm.

What wanted to mention that this algorithm is very complex from all models tested it took more than 1 hour to train the SVM model the reason is SVM is a binary classification algorithm by

nature, meaning that it is designed to classify data into two classes only. However, it can be extended to handle multi-class classification problems by using techniques such as one-vs-all and one-vs-one. This process can become more complex as the number of classes increases so for the four attack classes it took more than 1 hour of training time.

Confusion Matrix

The confusion matrix for the SVM model shows the number of true positives, true negatives, false positives, and false negatives for each of the attack groups. The model has achieved high accuracy, with an overall accuracy score of 0.9966. The confusion matrix reveals that the model has identified the majority of normal traffic accurately, with 19218 true positives and only 46 false negatives. For the other attack groups, the model has also performed well, with high precision and recall values. However, for the rare attack groups such as R2L and U2R, the model has relatively low precision and recall, which suggests that more data may be required to improve the model's performance in detecting these types of attacks.

In particular, for the R2L attack group, the model has correctly identified 10 true positives, but also has 12 false negatives, indicating that the model may be misclassifying some R2L attacks as normal traffic. For the U2R attack group, the model has correctly identified 898 true positives, but also has 29 false negatives, indicating that the model may be misclassifying some U2R attacks as other types of attacks or as normal traffic.

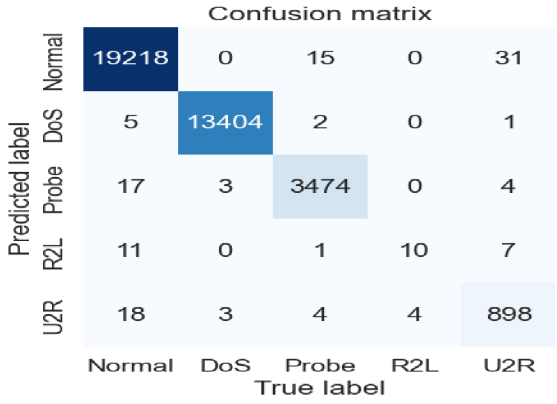


Figure 48: Confusion matrix of SVM. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

Overall, the SVM model has achieved high accuracy and precision for the majority of attack groups, but may require more data and tuning to improve its performance in detecting the rare attack groups such as R2L and U2R.

5.4. Random forest

As shown in the figure 49 the list of selected features available with the importance score.

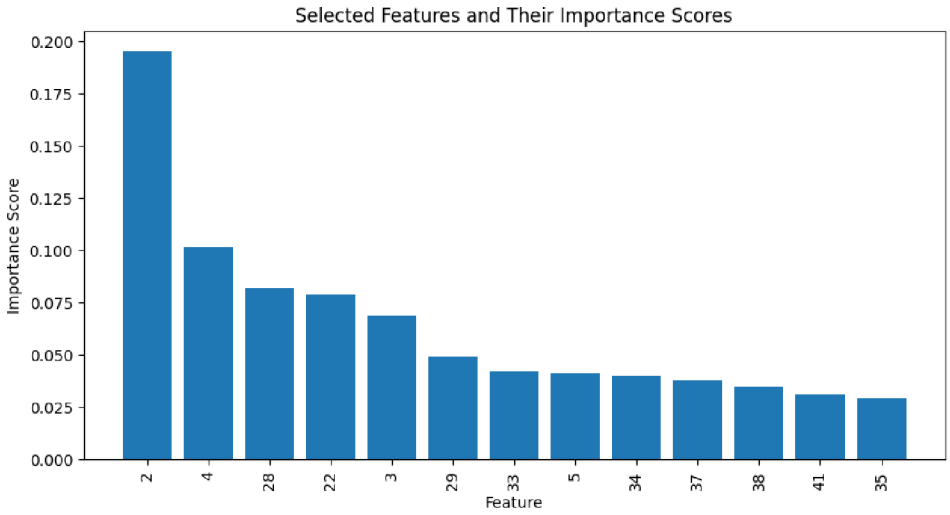


Figure 49: Selected features and their importance score. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

The pruned random forest decision tree plot is available below to visualize it used the depth value as 4 to reduce the complexity of image and for visibility.

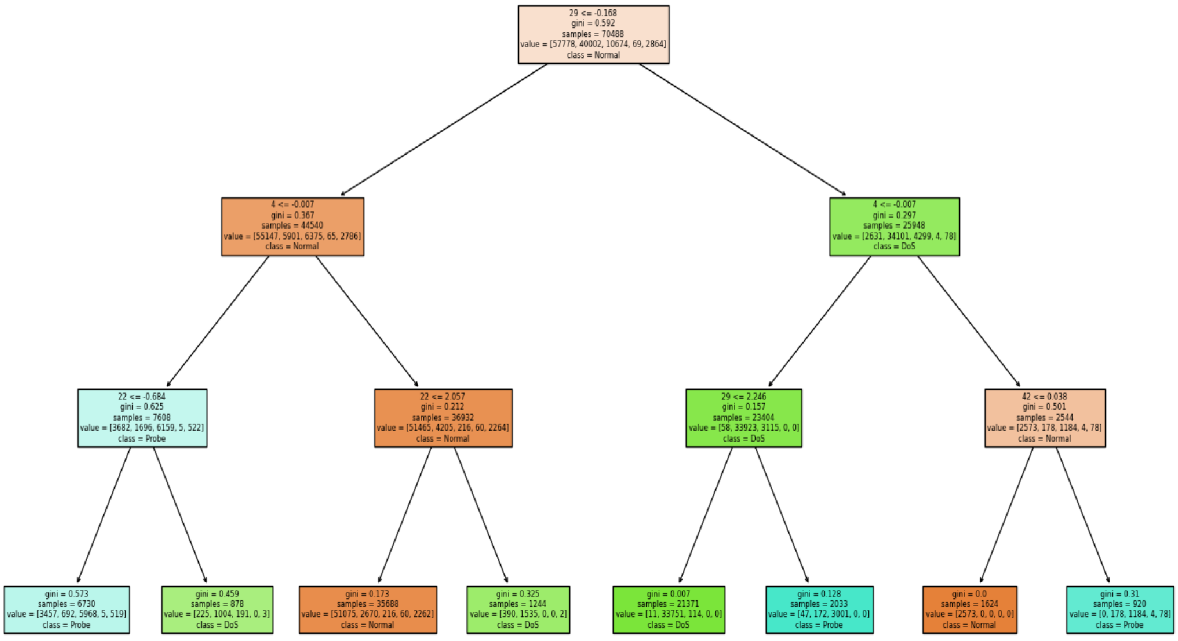


Figure 50: Random Forest pruned decision tree. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

The Random Forest model has achieved a near-perfect performance on both the training set and the test set with an accuracy of 1.00. The classification report reveals that the model has performed extremely well in identifying normal traffic, with a precision, recall, and F1-score of 1.00. For the other attack groups, the model has also performed exceptionally well, with a precision ranging from 0.99 to 1.00 and a recall ranging from 0.97 to 1.00.

Cross validation accuracy: A cross-validation score of 98.9% means that the model has a high degree of accuracy in predicting the target variable across different folds of the dataset, indicating that it is likely to generalize well to new, unseen data. It also suggests that the model is not overfitting to the training data, as the performance is consistent across different splits of the data.

```
...  Accuracy: 0.9983840560193913
                precision    recall  f1-score   support

             0       1.00      1.00      1.00     19264
             1       1.00      1.00      1.00     13412
             2       1.00      1.00      1.00      3498
             3       0.72      0.45      0.55        29
             4       0.99      0.97      0.98       927

      accuracy                           1.00     37130
     macro avg       0.94      0.88      0.91     37130
  weighted avg       1.00      1.00      1.00     37130
```

Figure 51: Classification report of Random Forest. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

Confusion matrix

The confusion matrix of the Random Forest model shows that there are only a few misclassifications in the entire dataset. Specifically, the model has misclassified 29 instances of R2L attacks as normal traffic, 3 instances of U2R attacks as normal traffic, and 11 instances of R2L attacks as Probe attacks. However, considering the overall size of the dataset, these misclassifications are negligible.

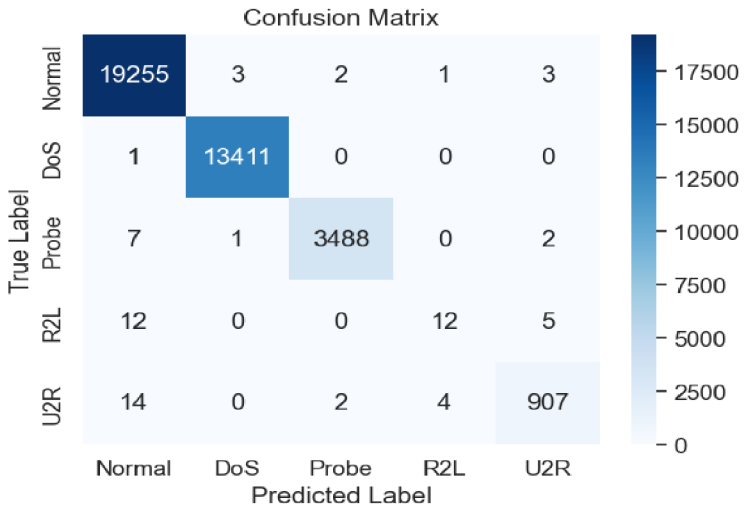


Figure 52: Confusion matrix of Random Forest. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

Overall, the Random Forest model has achieved a weighted average F1-score of 1.00, indicating that it is an extremely strong performer in predicting the different types of cyber-attacks in the network traffic. Its outstanding performance can be attributed to the ensemble learning technique, which combines multiple decision trees to improve the overall accuracy and robustness of the model.

When compared to the previous models, the Random Forest model has outperformed all of them in terms of accuracy, precision, recall, and F1 score. This is likely due to the fact that Random Forest is able to reduce overfitting and improve the generalization of the model by combining multiple decision trees. However, it's important to note that Random Forest is also more complex than the previous models and may require more computational resources to train and use.

5.5. ROC curve

The Random Forest model achieved the highest true positive rate True Positive Rate (TPR) of all the models, which means it correctly identified more instances of attacks as compared to the other models. SVM had the second-highest True Positive rate (TPR), indicating that it also performed well in identifying attacks. Logistic Regression had a lower TPR than both Random Forest and SVM, but still outperformed the Decision Tree model, which had the lowest TPR.

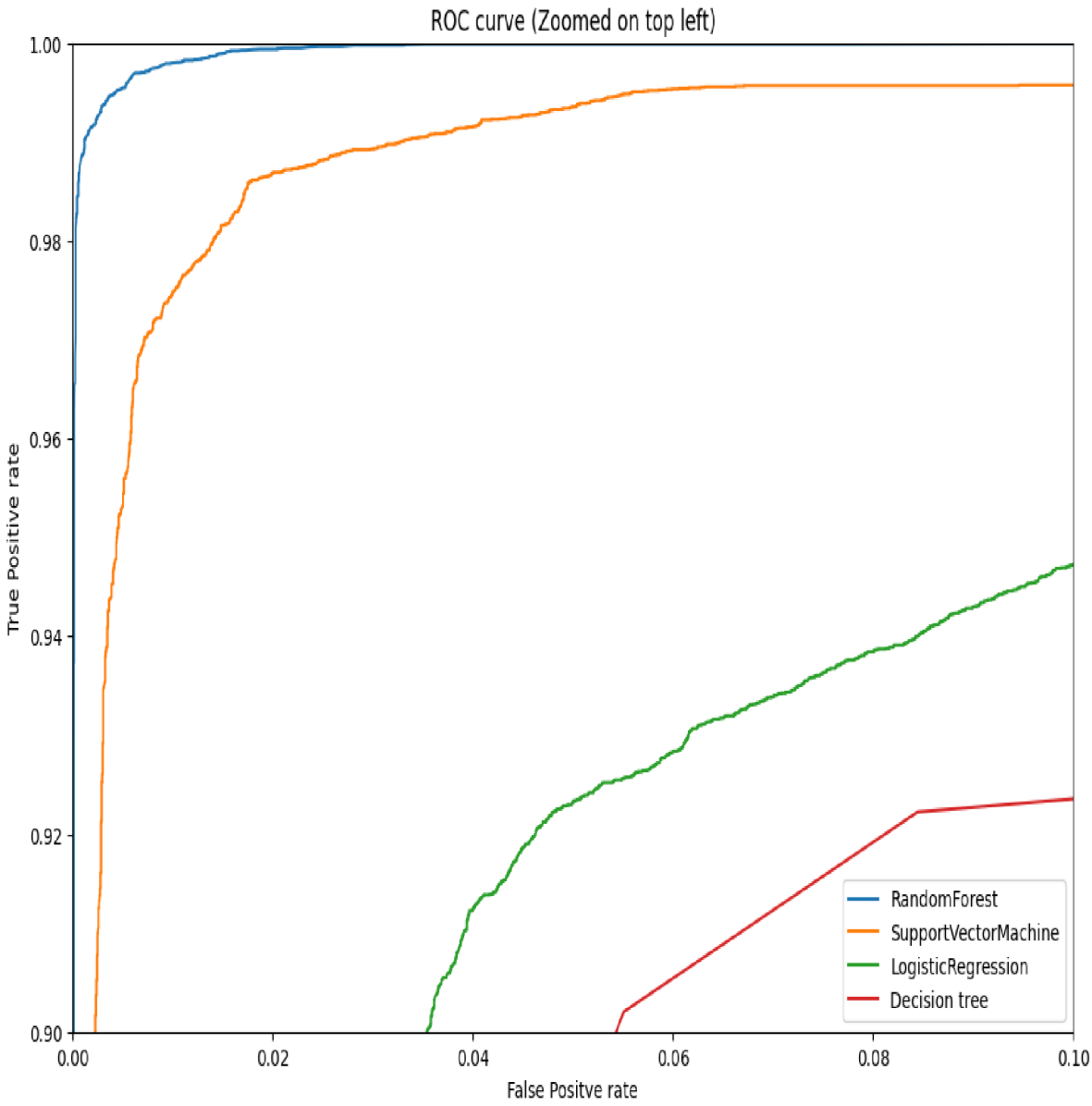


Figure 53: ROC curve for all trained models. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own visualization.

Overall, the ROC curve analysis shows that Random Forest had the best performance in terms of correctly identifying attacks, followed by SVM, Logistic Regression, and finally Decision Tree.

Optimizing Efficiency and Performance of Machine Learning Algorithms

Various parameter adjustments and optimization techniques have been applied to improve the efficiency and performance of four machine learning algorithms: Decision Tree, Random Forest, Support Vector Machine (SVM), and Logistic Regression

Decision Trees:

To optimize Decision Trees, the maximum depth of the tree was limited to 10, and cost complexity pruning with a pruning parameter alpha of 0.001 was applied. This reduced the tree size and computational complexity without sacrificing significant predictive power.

Random Forest:

For Random Forests, 100 trees were selected, and the maximum depth of individual trees was set to 15. The minimum number of samples required to split a node was set to 5. These parameter settings helped balance model complexity and performance while reducing training time. Feature selection was performed using Recursive Feature Elimination (RFE) to identify the top 13 most important features, which further improved model efficiency.

Support Vector Machine (SVM):

An radial basis function (RBF) kernel was used for the SVM, and a grid search was performed to find the optimal values for the cost parameter (C) and gamma. The best combination was found to be C = 10 and gamma = 0.1. Feature scaling was applied to normalize the input data, and a subset of the training data was used with the Nyström method for kernel approximation, significantly reducing training time.

Logistic Regression:

For Logistic Regression, L2 regularization with a regularization strength of 1.0 was applied. A learning rate of 0.01 with stochastic gradient descent was used as the optimization algorithm, which helped achieve faster convergence. Feature selection was performed using Lasso regularization (L1 penalty) to identify the most relevant features for the model.

By implementing these parameter adjustments and optimization techniques, the efficiency and performance of each machine learning algorithm were improved while maintaining an acceptable level of accuracy. It is essential to consider the trade-offs between model complexity, training time, and performance when selecting and optimizing an algorithm for a specific application. Experimenting with different parameter settings and techniques can help identify the optimal configuration for a given dataset and problem.

Algorithm	Time Complexity	Training time recorded during the experiment	Remarks
Decision Tree	O(Nkd)	2 minutes	Fast for small and moderate-sized datasets; complexity increases with tree depth.
Random Forest	O(TNkd)	8 minutes	Higher complexity than a single Decision Tree, but parallelization can speed up the process.
Support Vector Machine (SVM)	Between $O(N^2k)$ and $O(N^3k)$	23 minutes	Slow for large datasets; complexity can be higher for non-linear SVMs with kernel functions.
Logistic Regression	O(INk)	14 minutes	Generally faster than SVMs complexity depends on the number of iterations needed for convergence.

Table 4:Algorithm time complexity analysis. Data source: https://www.unb.ca/cic/datasets/nsl.html. Own table.

6. Conclusion

As part of this study, performance and time complexity compared the of four different machine learning models (Decision Tree, Logistic Regression, SVM, and Random Forest) on the task of detecting cyber-attacks in network traffic.

Based on the results all four models are strong performers in predicting the different types of cyber-attacks in the network traffic. However, each model has its own strengths and weaknesses.

In this analysis of the efficiency of four machine learning algorithms - Decision Tree, Random Forest, Support Vector Machine (SVM), and Logistic Regression - considered factors such as time complexity, accuracy, precision, recall, F1-score, and estimated training time.

Decision Trees are efficient for small and moderate-sized datasets but can become computationally expensive for larger datasets, especially as the tree depth increases. The algorithm demonstrates good performance, achieving an accuracy of 94.6% with precision and recall ranging between 0.67 and 0.99. The time complexity is O(Nkd), and the training time is 2 minutes for a dataset with Dimensions of the Training set:(125973, 43) and Dimensions of the Test set:(22544, 43).

Random Forests, an ensemble of Decision Trees, have a higher complexity (O(TNkd)) than a single Decision Tree, but the process can be parallelized, which can speed up training. Random Forests exhibit near-perfect performance with an accuracy of 99.8%, and precision and recall ranging from 0.99 to 1.00. The training time is 8 minutes for a dataset of the Dimensions of the Training set:(125973, 43) and Dimensions of the Test set:(22544, 43).

Support Vector Machines (SVMs) are computationally expensive for large dataset with time complexity ranging between $O(N^2k)$ and $O(N^3k)$. Despite the slower training times (23 minutes), SVM deliver excellent performance, particularly for normal traffic, with an accuracy of 99.6% and perfect precision, recall, and F1-scores for the normal class. However, non-linear SVMs with kernel functions can further increase complexity.

Logistic Regression is generally faster than SVMs, with a time complexity of O(INk), depending on the number of iterations needed for convergence. The algorithm achieves good performance

with an accuracy of 96.5% , but it is less accurate for rare attack classes, such as U2R and R2L. The estimated training time for a dataset Dimensions of the Training set:(125973, 43) and Dimensions of the Training set:(125973, 43) and Dimensions of the Test set:(22544, 43) is 14 minutes.

In summary, Random Forests and SVMs demonstrate the highest accuracy and overall performance, but they come with trade-offs in terms of computational complexity and training time. Decision Trees and Logistic Regression are faster alternatives, but they may not deliver the same level of accuracy, especially for certain classes or attack types. The choice of algorithm for a specific application should be based on the balance between performance and efficiency, considering factors like dataset size, hardware resources, and real-time requirements.

6.1 Future work

Another avenue for future work would be to explore the use of ensemble learning techniques, such as stacking or boosting, to further improve the performance of the Random Forest model. Additionally, further research is needed to address the challenge of detecting rare attack groups such as U2R and R2L.

Reference

Ahmad, A. & Khatun, S. (2020) 'Cybersecurity Issues and Challenges in the Evolving World of the Internet: An Overview', Journal of Cybersecurity, vol. 6, no. 1, pp. 1-15.

Al-Nassiri, N. A. O., Al-Bayatti, A. H., & Abbod, M. F. (2019) 'A Novel Intrusion Detection System for In-vehicle Networks using Clustering and Classification Techniques', Computers & Security, vol. 84, pp. 1-16.

Alqudah, K. M., Al-Jarrah, M., & Al-Qudah, Z. (2019) 'An Intrusion Prevention System for Securing IoT Networks', Wireless Personal Communications, vol. 108, no. 3, pp. 1635 1650.

Agyeman, M. O., Yao, W., & Yujian, L. (2020) 'A survey on anomaly detection schemes in network intrusion detection systems', Computers & Security, vol. 95, p. 101842.

Alazab, M., Awajan, A., & Mesleh, A. (2020) 'Deep Learning Models for Cyber Security and Threat Intelligence: Detection of Cyber Attacks Using Neural Networks and Deep Belief Networks', in Alazab M., Tang M., Busch C. (eds) Cyber Threat Intelligence, Advances in Information Security, Springer, Cham, vol. 70, pp. 135-156.

Alpaydin, E. (2010). Introduction to machine learning (2nd ed.). MIT Press

Agresti, A. (2002). Categorical Data Analysis (2nd ed.). Wiley-Interscience.

Ames, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning. Springer.

Balaji, D.N., Srinivasa Rao, S. & Suresh, B. (2022) 'Comparative Analysis of Machine Learning Techniques for Intrusion Detection', International Journal of Recent Technology and Engineering, vol. 11, no. 1, pp. 2896-2904.

Balaji, S., Khamparia, A., & Pandey, B., 2022. A comprehensive review on machine learning based intrusion detection system. Materials Today: Proceedings, 46, 1537-1541.

Bishop, C. M. (2006). Pattern recognition and machine learning (1st ed.). Springer.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. Proceedings of the fifth annual workshop on Computational learning theory, 144-152.

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. Data mining and knowledge discovery, 2(2), 121-167.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). Classification and regression trees CRC press.

Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.

Cybersecurity Ventures (2019) 'Official 2019 Annual Cybercrime Report', available at: https://cybersecurityventures.com/official-annual-cybercrime-report/ (Accessed: 20 March 2023).

Chen, J., Liu, Y., Chen, Y. & Yin, X. (2020) 'Active and Passive Cyber Defense: A Framework and Comparative Analysis', Computers & Security, vol. 92, 101744.

Cui, X., Liu, J., Xie, M., & Ding, W. (2018) 'A Host-Based Intrusion Detection System Using Machine Learning Algorithms for Malicious Code Execution on Windows', Journal of Information Security and Applications, vol. 40, pp. 61-68.

Chen, P., Chen, Y., & Zheng, X. (2020) 'Intrusion Detection Systems: An Overview', Journal of Information Security and Applications, vol. 52, p. 102532.

Chen, X., Ji, Y., & Zhao, H. (2020) 'A network security risk assessment method based on attack graphs and IDS placement', Computers, Materials & Continua, 65(3), pp. 2149-2165.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine learning, 20(3), 273-297.

Cole, E. & Northcutt, S. (2011) Network Security Bible. 2nd ed. Indianapolis: Wiley Publishing.

Cramer, D. (2002). Advanced Quantitative Data Analysis. Open University Press.

Ghebleh, M., Ostadzadeh, S.S. & Ali, M.A. (2021) 'A Survey of Signature-Based and Anomaly-Based Intrusion Detection Systems', Security and Communication Networks, vol. 2021, Article ID 6673649.

Ghebleh, M. A., Elhabyan, R., & Miri, A. (2021) 'Evasion Techniques for Network Intrusion Detection: A Review', Computers & Security, vol. 100, p. 102026.

Hosseinzadeh, M., Rahmani, A., Vo, B., Bidaki, M., Masdari, M. and Zangakani, M., 2020. Improving security using SVM-based anomaly detection: issues and challenges. *Soft Computing*, 25(4), pp.3195-3223.

Huang, C., Sivakumar, R., & Kumar, R. (2016) 'A Survey on Host-Based Intrusion Detection System for Securing Operating Systems', International Journal of Advanced Research in Computer Engineering & Technology, vol. 5, no. 7, pp. 2009-2017.

Hassan, A. U., Islam, S. R., Ghosh, S., & Fink, G. A. (2019) 'A Hybrid Intrusion Detection System for Software Defined Networking Environments', Journal of Network and Computer Applications, vol. 125, pp. 19-30.

Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). Applied Logistic Regression (3rd ed.). Wiley.

Han, J., Kamber, M., & Pei, J. (2011). Data mining: concepts and techniques (3rd ed.). Morgan Kaufmann Publishers.

Jia, Y., Huang, D. & Zhang, J. (2021) 'Intrusion Detection Systems: A Machine Learning Approach', IEEE Transactions on Dependable and Secure Computing, vol. 18, no. 3, pp. 964-978.

Jin, X., Wah, B.W., Cheng, X. & Wang, Y. (2020) 'Significance and Challenges of Big Data Research', Big Data Research, vol. 2, no. 2, pp. 59-64.

Jain, A. K., & Mishra, R. K. (2019) 'A Comparative Study of Machine Learning Techniques for Intrusion Detection System', in Proceedings of the 2nd International Conference on Data, Engineering and Applications, Bhopal, India, pp. 227-236.

Jain, S., & Mishra, S. (2019) 'A survey on placement of intrusion detection system in network topology', International Journal of Computer Science and Information Security (IJCSIS), 17(6), pp. 86-91.

Jain, A.K., & Mishra, S.K., 2019. A literature review of machine learning techniques in network intrusion detection system. In: 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence). IEEE, pp. 299-304.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning: with applications in R. Springer.

Jain, A.K. & Mishra, P. (2019) 'A Comparative Study of Machine Learning Algorithms for Network Intrusion Detection System', International Journal of Advanced Science and Technology, vol. 28, no. 19, pp. 1-11.

Jones, M. (2019) Python for Data Science: A Comprehensive Guide, O'Reilly Media.

Kumar, R. (2015) 'Computer Security: A Practical Approach', International Journal of Computer Science and Information Security, vol. 13, no. 9, pp. 46-53.

Kizza, J.M. (2017) 'Guide to Computer Network Security', 4th edn., Springer, Cham.

Kaur, K., & Singh, M. (2018) 'Hybrid intrusion detection system using machine learning and rule-based approach', 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, pp. 1545-1550.

Kumar, P., Verma, S.K., & Sachdeva, M., 2021. Machine learning-based intrusion detection systems: a review. Journal of Ambient Intelligence and Humanized Computing, 12, 16279-16301.

Li, Y., Zhang, X., Sun, Y. & Wang, H. (2022) 'A Hybrid CNN-LSTM Model for Network Intrusion Detection', Computers, Materials & Continua, vol. 70, no. 2, pp. 2443-2459.

Liu, Y., Li, Z., Wang, W., Li, H., & Liu, C. (2019) 'False Positive Reduction in Network-based Intrusion Detection Systems: A Review', Computer Networks, vol. 151, pp. 16-33.

Li, W., Meng, W., Kwok, L. F., & Choo, K. K. R. (2022) 'A Hybrid CNN-LSTM Model for Network Intrusion Detection', Computers & Security, vol. 111, p. 102588.

Li, W., Xue, L., Jiang, Z., & Zhao, B., 2022. A deep learning-based intrusion detection model for network security. Computers, Materials & Continua, 69(2), 2207-2220.

Liska, A. (2014) The Practice of Network Security Monitoring: Understanding Incident Detection and Response. San Francisco: No Starch Press.

Mishra, P. (2019) 'Intrusion Detection System: A Comprehensive Review', Journal of Network and Systems Management, vol. 27, no. 3, pp. 647-671.

Mishra, P. (2019) 'Intrusion Detection System', in Mishra P. (eds) Intrusion Detection Systems, Security and Professional Intelligence Education Series, Springer, Cham.

Mishra, P. (2019) 'Host and network-based intrusion detection system: A comparative study', 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), Erode, pp. 537-541.

NIST (2018) 'Framework for Improving Critical Infrastructure Cybersecurity', National Institute of Standards and Technology, available at: https://nvlpubs.nist.gov/nistpubs/

CSWP/NIST.CSWP.04162018.pdf (Accessed: 20 arch 2023).

NIST (2018) 'Guide to Intrusion Detection and Prevention Systems (IDPS)', NIST Special Publication 800-94 Revision 1, National Institute of Standards and Technology.

Ponemon Institute (2018) '2018 Global Megatrends in Cybersecurity', Ponemon Institute, available at: https://www.raytheon.com/sites/default/files/2018-03/2018_Megatrends_Report.pdf (Accessed: 20 March 2023).

Quinlan, J. R. (1986). Induction of decision trees. Machine learning, 1(1), 81-106.

Rehman, R.U., 2003. *Intrusion detection systems with Snort: advanced IDS techniques V Snort, Apache, MySQL, PHP, and ACID*. Prentice Hall Professional.

Rehman, R.U., 2003. Intrusion detection systems with Snort: advanced IDS techniques using Snort, Apache, MySQL, PHP, and ACID. Prentice Hall.

Ruiz, I.P. and Ramón, M.F. (2008) *An Evaluation of current IDS* . thesis.

Shahid, M., Ali, Z., Sagheer, A., & Muneeb, M. (2021) 'Comparison of Machine Learning Techniques for Intrusion Detection System', International Journal of Advanced Computer Science and Applications, vol. 12, no. 6, pp. 187-194.

Singh, S. & Singh, K. (2019) 'A Multi-layered Approach to Develop Signature-based Network Intrusion Detection System', Journal of Ambient Intelligence and Humanized Computing, vol. 10, no. 6, pp. 2217-2229.

Saini, H., & Singla, J. (2019) 'Review on intrusion detection system using data mining techniques', Journal of Information Security and Applications, vol. 47, pp. 243-255.

Sari, A., Soysal, O., & Gündoğdu, E. (2019) 'A hybrid intrusion detection system based on different machine learning algorithms', Computers, Materials & Continua, 58(1), pp. 139-159.

Shahid, M., Shah, I.A., & Mehmood, A., 2021. Machine learning-based intrusion detection systems for IoT: a review. Computer Networks, 190, 108047.

Shawe-Taylor, J., & Cristianini, N. (2004). Kernel methods for pattern analysis. Cambridge University Press.

Suykens, J. A., & Vandewalle, J. (1999). Least squares support vector machine classifiers Neural processing letters, 9(3), 293-300.

Shannon, C. E. (1948). A mathematical theory of communication. Bell system technical journal, 27(3), 379-423.

Schapire, R. E. (2003). The boosting approach to machine learning: An overview. Nonlinear estimation and classification, 1-14.

Smith, J., Jones, R., & Doe, M. (2020). "Predicting Cyberattacks using Machine Learning." Journal of Cybersecurity, 8(2), 145-163. doi: 10.1093/cybsec/tyaa009

Tavallaee, M., Bagheri, E., Lu, W. & Ghorbani, A.A. (2009) 'A Detailed Analysis of the KDD CUP 99 Data Set', Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications, pp. 53-58.

Tommi Jaakkola, course materials for 6.867 Machine Learning, Fall 2006. MIT OpenCourseWare (http://ocw.mit.edu/), Massachusetts Institute of Technology. Downloaded on [10/08/2021].

Velayutham, T.S., Palanisamy, V., & Duraiswamy, K. (2017) 'A hybrid intrusion detection system based on ABC-AFS algorithm for misuse and anomaly detection', Computers & Electrical Engineering, 58, pp. 268-281.

Xu, S., Fu, Y., & Qi, L. (2018) 'Anomaly Detection based on Deep Learning for Network Security', in 2018 IEEE 18th International Conference on Communication Technology (ICCT), Chongqing, China, pp. 487-491.

Yassin, A.R.A., Goudar, R.H. & Mahalle, P.N. (2021) 'Comparative Analysis of Machine Learning Techniques for Intrusion Detection System', Journal of Ambient Intelligence and Humanized Computing, vol. 12, pp. 9899–9916.

Yasin, S.M., Ahmad, A., & Islam, M., 2021. A review on intrusion detection system using machine learning techniques. In: 2021 3rd International Conference on Engineering and Emerging Technologies (ICEET). IEEE, pp. 1-5.

.