



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science Engineering and Information Systems (SCORE)

B.Tech (Information Technology)

Course Project Report

INTRUSION DETECTION SYSTEM USING MACHINE ALGORITHM

Submitted for the Course:

ITE3007 Cloud Computing and Virtualization

Offered by:

Dr. M. LawanyaShri
Fall 2023-2024

By

Dhruv Umesh Sompura (20BIT0357)
Ranoj Bhowmik (20BIT0423)

November 2023



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science Engineering and Information Systems B.Tech
(Information Technology)
FALL 2023 – 2024
ITE 3007: Cloud Computing and Virtualization

A Report on the Course Project
Intrusion detection system using Machine Learning

TEAM Name : Cumulus
Dhruv Umesh Sompura; 20BIT0357; 9594675960; dhruvumesh.sompura2020@vitstudent.ac.in Ranoj Bhowmik; 20BIT0423; 7439149199; ranoj.bhowmik2020@vit.ac.in
Intrusion detection system using Machine Learning

1. Problem Statement

1.1 Background (System Study Details in brief)

Network security is a relatively obscure branch of cyber-safety, especially when it comes to the average person. The elderly gentlemen across the street cannot be expected to understand why he got a DDoS attack whilst making his routine payment to the milkman. This is where our Intrusion Detection system/model deployed on cloud, comes into place.

Intrusion Detection System (IDS) is a Device or software application which monitors the network or system activities and finds if there is any malicious activity occur. Outstanding growth and usage of internet raises concerns about how to communicate and protect the digital information safely. In today's world hackers use different types of attacks for getting the valuable information. Many of the intrusion detection techniques, methods and algorithms help to detect those several attacks. The main objective of this project is to detect types of attacks, that are capable to detect and prevent network intrusion

1.2 Problem Statement

The increasing frequency and sophistication of cyber threats pose a significant challenge to the security of digital systems. As technology advances, there is a growing need for a robust Intrusion Detection System (IDS) that can effectively identify and mitigate various forms of malicious activities within networks. The objective of this project is to develop an IDS that not only detects known intrusion patterns but also adapts to evolving cyber threats, ensuring the integrity and security of sensitive data in the face of an ever-changing threat landscape Network intrusion detection using ML algorithms is a critical cybersecurity problem due to the complexity of network data, the need for real-time detection, class imbalance issues, and the threat of adversarial attacks. Addressing these challenges involves efficient data preprocessing, feature engineering, robust model selection, and continuous monitoring to enhance network security.

1.3 Novelty

In the ever-evolving landscape of cybersecurity, our Intrusion Detection System (IDS) stands as a stalwart guardian, leveraging state-of-the-art machine learning algorithms on Amazon SageMaker to fortify network defenses. As internet usage burgeons, so does the imperative to shield digital information from a plethora of malicious activities.

Objective:

The crux of our project is twofold: detect and prevent a spectrum of attacks orchestrated by cunning intruders. Our IDS employs a robust set of techniques, methods, and algorithms to identify and thwart potential threats, ensuring the sanctity of your digital space.

Introduction:

In the realm of network defense, Intrusion Detection assumes a pivotal role, alerting security administrators to potential threats such as intrusions, attacks, and malware. Our IDS, a sophisticated system, diligently monitors for suspicious activities, issuing timely alerts upon detection. Beyond mere alerting, our system can take proactive measures, including blocking traffic from identified suspicious IP addresses.

Incorporating Machine Learning:

To elevate the efficacy of our IDS, we've seamlessly integrated machine learning algorithms on AWS SageMaker. This amalgamation enhances our system's ability to discern anomalous activities with unparalleled accuracy. The machine learning models continuously evolve, learning from patterns and trends, thereby adapting to emerging threats in real-time.

Benefits:

1. *Precision in Detection:* Our IDS, powered by machine learning, refines its understanding of normal and abnormal behaviors, minimizing false positives and ensuring precise threat detection.
2. *Adaptability:* The system's ability to adapt to evolving threats ensures a proactive defense mechanism, staying one step ahead of potential intrusions.
3. *Data-Driven Security:* Organizations can leverage the rich data provided by our IDS to not only bolster their security infrastructure but also to fine-tune network configurations and identify potential vulnerabilities.

In conclusion, our IDS on AWS SageMaker represents a paradigm shift in network intrusion detection, fusing traditional methods with cutting-edge machine learning to provide a robust, adaptive, and highly accurate defense against an ever-expanding array of cyber threats.

1.4 Dataset

We will use the NSL KDD dataset provided by the University of New Brunswick Link. The dataset is given in text format and has to be converted to csv format for processing. Each record has a total of 41 attributes for identifying the various features of the network packets and a label is named to every packet. The 42nd attribute unfolds data about 13 labels of each record. They're categorized as 1 for ordinary class and 4 attack-type classes which are DoS, Probe, R2L, and U2R which are converted into single category as 0.

2. Related Works

2.1 Literature Survey

1. Intrusion Detection System Classification Using Different Machine Learning Algorithms on KDD-99 and NSL-KDD Datasets - A Review Paper: Literature Review and Citation

In the paper titled "Intrusion Detection System Classification Using Different Machine Learning Algorithms on KDD-99 and NSL-KDD Datasets - A Review Paper" by Rama Devi Ravipati and Munther Abualkibash, published in the International Journal of Computer Science & Information Technology (IJCSIT) in June 2019, the authors delve into the critical realm of Intrusion Detection Systems (IDS). The study is grounded in the challenges faced by contemporary anomaly detection systems, emphasizing high false alarm rates and moderate accuracy and detection rates. By conducting an experiment that evaluates diverse machine learning algorithms on the KDD-99 Cup and NSL-KDD datasets, the paper seeks to discern the most effective approach for achieving superior accuracy, detection rates, and maintaining a reasonable false alarm rate.

Citation-Ravipati, R. D., & Abualkibash, M. (2019, June). Intrusion Detection System Classification Using Different Machine Learning Algorithms on KDD-99 and NSL-KDD Datasets - A Review Paper. International Journal of Computer Science & Information Technology (IJCSIT), Vol 11, No 3, June 2019. Available at SSRN: <https://ssrn.com/abstract=3428211> or <http://dx.doi.org/10.2139/ssrn.3428211>.

2. BAT: Deep Learning Methods on Network Intrusion Detection Using NSL-KDD Dataset - Literature Review and Citation

The paper titled "BAT: Deep Learning Methods on Network Intrusion Detection Using NSL-KDD Dataset" introduces a novel traffic anomaly detection model, BAT, which combines Bidirectional Long Short-term Memory (BLSTM) and attention mechanism to address the challenges of low accuracy and feature engineering in intrusion detection. The model, specifically referred to as BAT-MC due to the incorporation of multiple convolutional layers, utilizes a softmax classifier for network traffic classification. Tested on a public benchmark dataset, the experimental results demonstrate superior performance compared to other methods.

Citation: Ravipati, R. D., & Abualkibash, M. (2019, June). Intrusion Detection System Classification Using Different Machine Learning Algorithms on KDD-99 and NSL-KDD Datasets - A Review Paper. International Journal of Computer Science & Information Technology (IJCSIT), Vol 11, No 3, June 2019. Available at SSRN: <https://ssrn.com/abstract=3428211> or <http://dx.doi.org/10.2139/ssrn.3428211>

3. Supervised feature selection techniques in network intrusion detection: A critical review - Literature Review and Citation

The paper titled "Supervised feature selection techniques in network intrusion detection: A critical review" delves into the role of Machine Learning (ML) techniques in network intrusion detection, emphasizing the challenges posed by the vast number of features in data traffic. The review comprehensively evaluates recent datasets, explores various Feature Selection (FS) approaches, including Multi-Objective Evolutionary techniques, and conducts experimental analyses on feature correlation, time complexity, and performance. The comparisons presented offer valuable insights for network/security managers considering the integration of ML concepts into intrusion detection.

Citation: M. Di Mauro, G. Galatro, G. Fortino, A. Liotta, "Supervised feature selection techniques in network intrusion detection: A critical review," Engineering Applications of Artificial Intelligence, Volume 101, 2021, 104216, ISSN 0952-1976, <https://doi.org/10.1016/j.engappai.2021.104216>.

4. Packet Header Classification for Network Intrusion Detection System Based on FPGA - Literature Review and Citation

The paper on "Packet Header Classification for Network Intrusion Detection System Based on FPGA" introduces a novel algorithm utilizing finite state machine (FSM) techniques for packet header classification. Employing field-programmable gate array (FPGA) technology, the algorithm compares each header field against predefined rules stored in the FPGA chip, demonstrating high processing speed and successful packet classification in simulations.

Citation: Y. H. Dakhil, M. E. Özbek, and B. R. Al-Kaseem, "Packet Header Classification for Network Intrusion Detection System Based on FPGA," 2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), Ankara, Turkey, 2022, pp. 1-6, doi: 10.1109/HORA55278.2022.9800025

5. A Review on Feature Selection and Ensemble Techniques for Intrusion Detection System - Literature Review and Citation

The paper titled "A Review on Feature Selection and Ensemble Techniques for Intrusion Detection System" by Majid Torabi, N. Udzir, Mohd Taufik Abdullah, and Razali Yaakob, published in the International Journal of Advanced Computer Science and Applications in 2021, addresses the challenges in developing efficient Intrusion Detection Systems (IDS) capable of handling large amounts of data in real-time circumstances. The review focuses on feature selection and ensemble techniques in anomaly-based IDS research, discussing the importance of selecting relevant features for increased efficiency in terms of accuracy and computational efficiency. The paper provides an analysis of various IDS-based machine learning techniques, highlighting the significance of ensemble methods in improving the performance of anomaly-based IDS models. The review concludes by addressing open issues in the field and suggesting research trends for designing efficient anomaly-based IDS.

Citation: Torabi, Majid, Nur Izura Udzir, Mohd Taufik Abdullah, and Razali Yaakob. "A Review on Feature Selection and Ensemble Techniques for Intrusion Detection System." International Journal of Advanced Computer Science and Applications, 12 (2021): n. pag.

6. Intrusion Detection Using Neural Networks and Support Vector Machines - Literature Review and Citation

The paper on "Intrusion Detection Using Neural Networks and Support Vector Machines" by Srinivas Mukkamala, Guadalupe Janoski, and Andrew H. Sung, affiliated with the New Mexico Institute of Mining and Technology and the University of Southern Mississippi, respectively, explores approaches to intrusion detection using neural networks and support vector machines. The key focus is on discovering useful patterns or features that

describe user behavior and building classifiers based on relevant features to recognize anomalies and known intrusions, potentially in real-time. Using benchmark data from a KDD competition, the paper demonstrates the efficiency and accuracy of classifiers built for intrusion detection. The performance of neural networks and support vector machine-based systems is compared in the context of intrusion detection.
Citation: Mukkamala, Srinivas, Guadalupe Janoski, and Andrew H. Sung. "Intrusion Detection Using Neural Networks and Support Vector Machines." Proceedings of the International Joint Conference on Neural Networks, 2002, pp. 1702-1707, doi: 10.1109/IJCNN.2002.1007774.

7.Outside the Closed World: On Using Machine Learning for Network Intrusion Detection - Literature Review and Citation

The paper titled "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection" by Robin Sommer and Vern Paxson, presented at the IEEE Symposium on Security and Privacy in 2010, critically examines the application of machine learning in the context of network intrusion detection. The authors identify differences between the network intrusion detection problem and other areas where machine learning is more commonly deployed. They argue that the unique challenges of finding attacks make it significantly harder to employ machine learning effectively in intrusion detection. The paper provides insights into the specific challenges of network intrusion detection and offers guidelines to strengthen future research on anomaly detection in this domain.
Citation: Sommer, Robin, and Vern Paxson. "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection." Proceedings - IEEE Symposium on Security and Privacy, 2010, pp. 305-316, doi: 10.1109/SP.2010.25.

8.A detailed analysis of the KDD CUP 99 data set - Literature Review and Citation

The paper titled "A detailed analysis of the KDD CUP 99 data set" by Mahbod Tavallae, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani, presented at the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, addresses issues affecting the performance evaluation of anomaly detection systems using the KDD CUP 99 dataset. The authors conduct a statistical analysis, identifying shortcomings that result in a poor evaluation of anomaly detection approaches. To address these issues, they propose a new dataset, NSL-KDD, which overcomes the shortcomings of the KDD CUP 99 dataset.
Citation: Tavallae, Mahbod, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. "A detailed analysis of the KDD CUP 99 data set." 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, 2009, pp. 1-6, doi: 10.1109/CISDA.2009.5356528.

9.Machine Learning Techniques for Intrusion Detection: A Comparative Analysis - Literature Review and Citation

The paper titled "Machine Learning Techniques for Intrusion Detection: A Comparative Analysis" by Yasir Hamid, M. Sugumaran, and Ludovic Journaux, presented at the International Conference on Informatics and Analytics in 2016, evaluates various machine learning algorithms provided by Weka against the standard dataset for intrusion detection, KDD Cup 99. The authors focus on metrics such as False Positive Rate, precision, ROC, and True Positive Rate to assess the performance of different machine learning techniques in intrusion detection.
Citation: Hamid, Yasir, M. Sugumaran, and Ludovic Journaux. "Machine Learning Techniques for Intrusion Detection: A Comparative Analysis." Proceedings of the International Conference on Informatics and Analytics (ICIA-16), 2016, Article 53, 1-6, doi: 10.1145/2980258.2980378.

10.Evaluation of Machine Learning Algorithms for Intrusion Detection System - Literature Review and Citation

The paper titled "Evaluation of Machine Learning Algorithms for Intrusion Detection System" by Mohammad Almseidin, Maen Alzubi, Szilveszter Kovacs, and Mouhammd Alkasassbeh, presented at the 2017 IEEE 15th International Symposium on Intelligent Systems and Informatics, focuses on the assessment of various machine learning classifiers for intrusion detection using the KDD intrusion dataset. The study involves experiments to evaluate the performance of selected classifiers, emphasizing metrics such as false negative and false positive rates to enhance the intrusion detection system's detection rate. The experiments reveal that the decision table classifier achieved the lowest false negative value, while the random forest classifier attained the highest average accuracy rate.
Citation: Almseidin, M., Alzubi, M., Kovacs, S., & Alkasassbeh, M. "Evaluation of machine learning algorithms for intrusion detection system." 2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY), Subotica, Serbia, 2017, pp. 277-282, doi: 10.1109/SISY.2017.8080566.

2.2 Comparative Statement

Paper Title	Authors	Year	Venue/Journal	Key Contributions
1. IDS Classification Using ML	R. D. Ravipati, M. Abualkibash	2019	IJCSIT	Evaluates ML algorithms on KDD-99 and NSL-KDD datasets, focusing on accuracy, detection rates, and false alarm rates.
2. BAT: DL for NID Using NSL-KDD	T. Su, H. Sun, J. Zhu, S. Wang, Y. Li	2020	IEEE Access	Introduces BAT model, combining BLSTM and attention for NID, demonstrating superior performance on NSL-KDD dataset.
3. Supervised FS Techniques in NID	M. Di Mauro, G. Galatro, G. Fortino, A. Liotta	2021	Eng. App. AI	Reviews ML's role in NID, explores FS techniques, and conducts experiments on feature correlation, time complexity, and performance.
4. Packet Header Classification for NIDS	Y. H. Dakhil, M. E. Özbek, B. R. Al-Kaseem	2022	HORA Congress	Introduces FPGA-based algorithm for packet header classification, demonstrating high processing speed in simulations.
5. Feature Selection and Ensemble Techniques Review	M. Torabi, N. Udzir, M. T. Abdullah, R. Yaakob	2021	IJACSA	Reviews feature selection and ensemble techniques in anomaly-based IDS, emphasizing their importance in improving efficiency.
6. IDS Using Neural Networks and SVM	S. Mukkamala, G. Janoski, A. H. Sung	2002	IJCNN	Explores intrusion detection using neural networks and SVM, comparing their efficiency and accuracy in identifying anomalies.
7. On Using ML for Network IDS	R. Sommer, V. Paxson	2010	IEEE S&P	Critically examines ML in network intrusion detection, highlighting challenges and providing guidelines for future research.
8. Detailed Analysis of KDD CUP 99	M. Tavallaee, E. Bagheri, W. Lu, A. A. Ghorbani	2009	IEEE Symposium	Analyzes KDD CUP 99 dataset, identifies issues affecting anomaly detection evaluation, and proposes the NSL-KDD dataset.
9. ML Techniques for IDS: Comparative Analysis	Y. Hamid, M. Sugumaran, L. Journaux	2016	ICIA-16	Evaluates various ML algorithms using Weka against KDD Cup 99, focusing on metrics like False Positive Rate, ROC, and precision.
10. Evaluation of ML Algorithms for IDS	M. Almseidin, M. Alzubi, S. Kovacs, M. Alkasassbeh	2017	IEEE SISY	Assesses ML classifiers for intrusion detection using the KDD intrusion dataset, emphasizing false negative and false positive rates.

2.3 Hardware Requirements

1. Server/Host Machine:

Specification: Multi-core processor (e.g., Intel Core i5 or equivalent), 16 GB RAM, 500 GB HDD/SSD, Gigabit Ethernet interface.
Purpose: Hosting the intrusion detection system, training machine learning models, and processing network traffic.

2. Network TAP or SPAN Port:

Specification: Gigabit or higher network TAP/SPAN port for capturing network traffic.
Purpose: Collecting raw network data for analysis by the intrusion detection system.

3. Storage:

Specification: RAID-configured storage with sufficient capacity for storing historical network traffic data and system logs.
Purpose: Storing and managing captured network data, alerts, and system logs.

4. Backup and Redundancy:

Specification: Regular automated backups, redundant power supplies, and network connectivity.
Purpose: Ensuring data availability and system resilience in case of hardware failures.

2.4 Software Requirements

1. Machine Learning Framework:

Specification: TensorFlow, PyTorch, or scikit-learn.
Purpose: Developing, training, and deploying machine learning models for intrusion detection.

2. Database System:

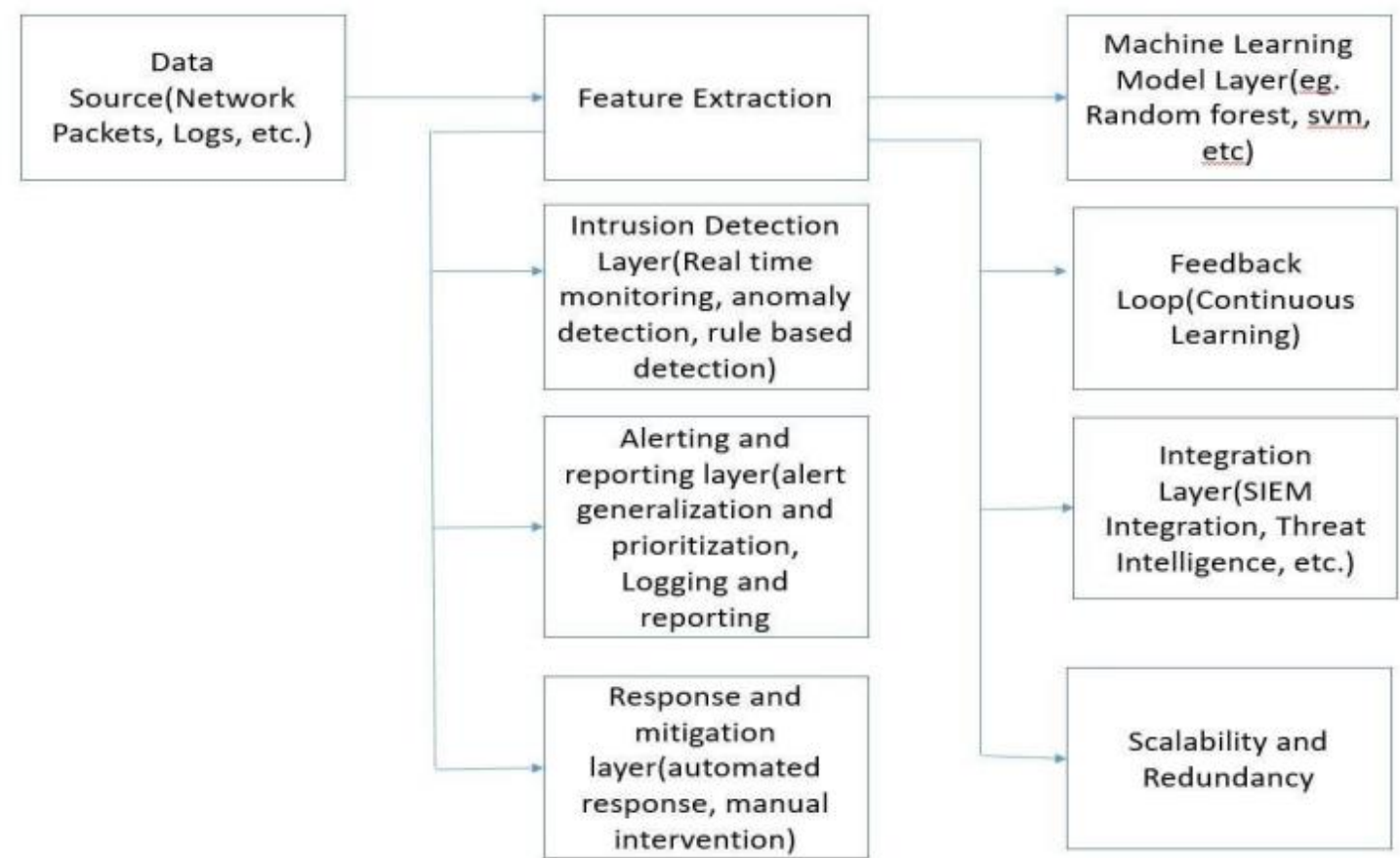
Specification: MySQL, PostgreSQL, or similar relational database.
Purpose: Storing historical network traffic data, alerts, and metadata.

3. Version Control System:

Specification: Git for managing codebase and configuration files.
Purpose: Tracking changes, collaborating, and maintaining version history

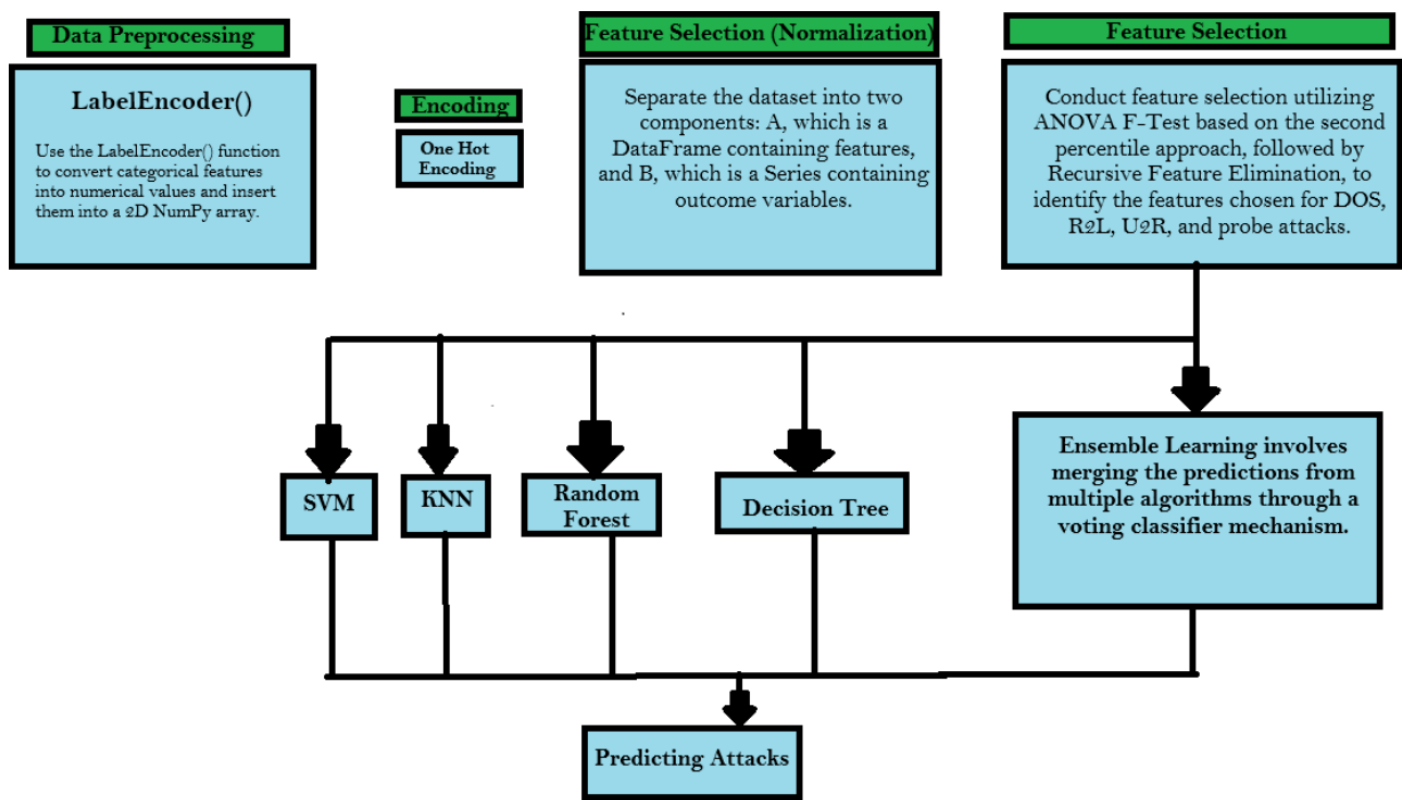
3. System Design

3.1 High-Level Design (Black Box design)



In the high level diagram-once we obtain data from prevalent data sources such as a dataset-we extract certain features using Machine Learning (Ensemble Algorithm consisting of random forest, svm etc.). We then build an intrusion detection layer that detects network anomalies based on predefined standards. We also add a continuous feedback loop that allows the model to keep learning. Further an integration layer (with SIEM integration, Threat intelligence etc.) is built. Finally we deploy alerting, reporting and mitigation layers to do their relevant tasks.

3.2 Low-Level Design (Detailed design)



The LabelEncoder() function converts categorial features such as text and labels into numbers and stores them in a 2D array. We use ONE HOT ENCODING to encode the data. We select the features in the dataset by splitting the dataset into two components-A, a DataFrame containing features and B which is a series containing outcome variables. We then select certain features by using ANOVA, F-Test, followed by Recursive function elimination to identify certain features chosen for DOS, R2L, U2R and probe attacks. We then pass the data through an Ensemble model that considers output from a decision tree, Random Forest, KNN and SVM model to produce a final result.

4. System Implementation

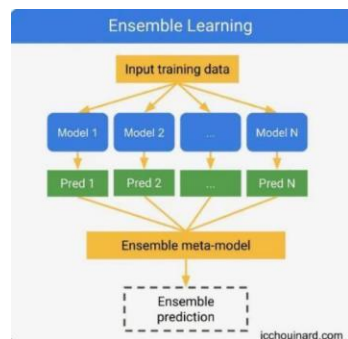
4.1 Algorithms (followed, proposed or altered)

The algorithms we have chosen for this project are as follows:

Ensemble Algorithms:

Ensemble Algorithms combine the predictions of multiple models to make a final decision. It's similar to asking a panel of experts for their advice. Each expert has their opinion on the matter and the algorithm finally combines multiple results and opinions and provides a more accurate result/decision than any single expert could.

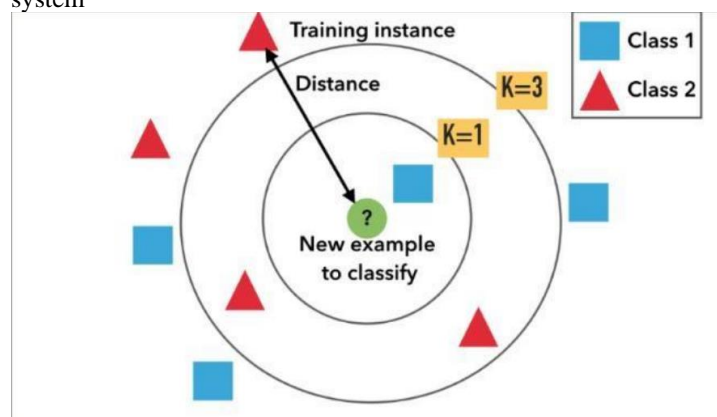
In the context of Network Intrusion Detection Systems, Ensemble Algorithm models like Random Forest are extremely useful. Random Forest creates a 'forest' of decision trees, each trained on different dataset subsets. When a new network packet is encountered, it's passed through each tree and the majority vote from all the trees determines the final classification (normal/intrusion)



K Nearest Neighbours (KNN) Algorithm:

The K Nearest Neighbours (KNN) algorithm operates on the premise of leveraging similarity metrics to classify data points based on their proximity to existing instances within a dataset. In short, KNN identifies the K closest neighbours to a given data point and assigns the category that is most prevalent amongst these neighbours. This technique assumes that similar data tend to share common characteristics and outcomes.

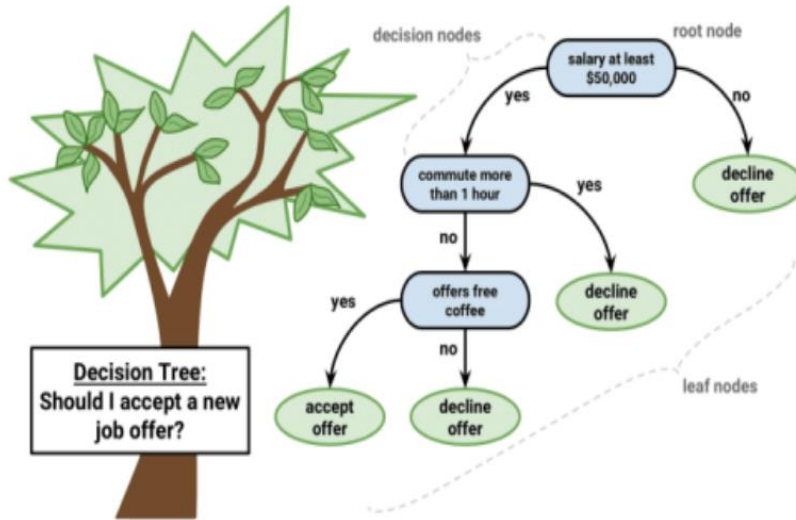
With reference to an IDS (Intrusion Detection System)- consider the scenario where network packets are akin to data points. When a new network packet arrives, the IDS employs KNN to scrutinize historical network packets (neighbours) that are similar to the incoming packet in terms of various attributes. By determining the predominant category among these past packets the IDS determines whether the new packet signifies a regular network activity or a potential intrusion attempt. This methodology gives a data-driven approach to anomaly detection, enhancing the security posture of the system



Decision Tree Algorithm

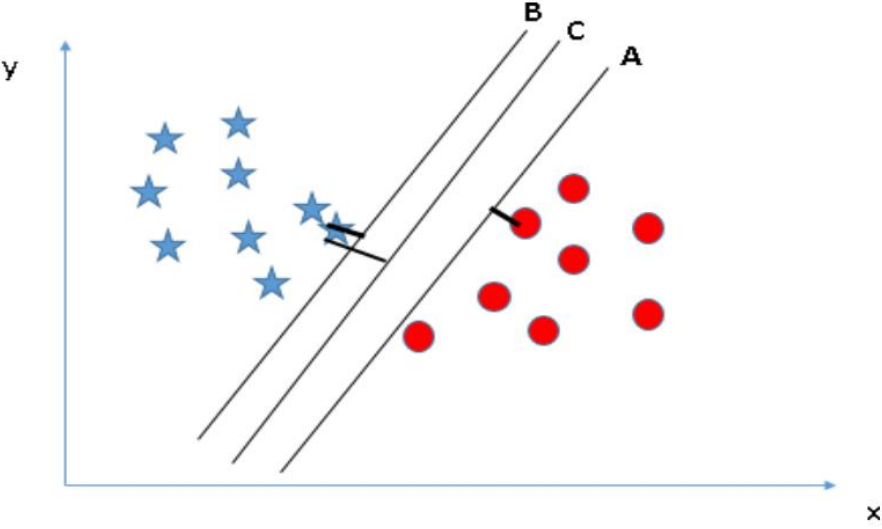
The Decision Tree algorithm is a hierarchical, rule-based model used for both classification and regression tasks. It repeatedly partitions the data based on the most informative features, creating a tree-like structure where decisions occur at each node.

In an IDS scenario, think of network features like IP addresses, packet sizes, and protocols as decision points in the tree. When a new network packet arrives, the Decision Tree algorithm examines these features sequentially, making decisions at each step. For instance, it might first check if the packet's source IP matches a known malicious source. If not, it proceeds to the next relevant feature until it accurately classifies the packet as normal or intrusive. This hierarchical decision-making mirrors the way security experts analyze network data for potential threats.



Support Vector Machine (SVM) Algorithm:

The Support Vector Machine (SVM) algorithm is a sophisticated supervised learning technique used primarily for classification tasks. Given data about various objects and each object has different features like size, weight and color, SVM analyses the data by turning each object into a point in a multi-dimensional space. The number of dimensions depends on how many features each object has. SVM thereby finds the best possible line (or hyperplane) that really separates the objects into distinct categories. SVM’s goal is to position this barrier in such a way that it creates the maximum gap between objects of different categories. These crucial points that are right on the edge of this gap are called Support Vectors. In the context of IDS-an SVM algorithm can work by drawing a clear line between usual and unusual network behaviours. SVM draws an optimal barrier that maximizes the distance between typical and unusual network behaviours.



4.2 Mathematical Model

Since our model uses ensemble algorithm primarily-it does not follow a fixed mathematical model, instead it combines multiple models to make a good prediction. A general overview of the models/algos are as follows:

Ensemble Algorithm:

Ensemble methods combine predictions from multiple machine learning models to improve overall performance and robustness. Common ensemble techniques include Bagging (used in Random Forest), Boosting, and Stacking. The mathematical foundation involves combining the outputs of base models, often through a voting or averaging mechanism.

Support Vector Machines (SVM):

SVM is a supervised learning algorithm for classification or regression tasks. The mathematical model of SVM involves finding a hyperplane that best separates different classes in the feature space. It aims to maximize the margin between classes while minimizing classification errors.

Random Forest (RF):

Random Forest is an ensemble learning method based on constructing a multitude of decision trees during training. The mathematical model of a Random Forest involves training multiple decision trees independently and combining their outputs through voting (classification) or averaging (regression).

k-Nearest Neighbors (KNN):

KNN is a non-parametric, lazy learning algorithm used for classification and regression. The mathematical model is simple: to classify a data point, it looks at the class labels of its k nearest neighbors (points with similar features).

4.3 Module Development –Code

```
import pandas as pd
import numpy as np
import sys
import sklearn
import io
import random

train_url = 'Training.csv'
test_url = 'Testing.csv'

col_names = ["duration","protocol_type","service","flag","src_bytes",
"dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
"logged_in","num_compromised","root_shell","su_attempted","num_root",
"num_file_creations","num_shells","num_access_files","num_outbound_cmds",
"is_host_login","is_guest_login","count","srv_count","serror_rate",
"srv_serror_rate","error_rate","srv_rerror_rate","same_srv_rate",
"diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
"dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
"dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
"dst_host_rerror_rate","dst_host_srv_rerror_rate","label"]

df = pd.read_csv(train_url,header=None, names = col_names)

df_test = pd.read_csv(test_url, header=None, names = col_names)

print('Dimensions of the Training set:',df.shape)
print('Dimensions of the Test set:',df_test.shape)
df.head(5)
print('Label distribution Training set:')
print(df['label'].value_counts())
print()
print('Label distribution Test set:')
print(df_test['label'].value_counts())
*Step 1: Data preprocessing:*

One-Hot-Encoding

print('Training set:')
for col_name in df.columns:
    if df[col_name].dtypes == 'object' :
        unique_cat = len(df[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_cat=unique_cat))

print()
print('Distribution of categories in service:')
print(df['service'].value_counts().sort_values(ascending=False).head())
# Test set
print('Test set:')
for col_name in df_test.columns:
    if df_test[col_name].dtypes == 'object' :
        unique_cat = len(df_test[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_cat=unique_cat))

*LabelEncoder*

*Insert categorical features into a 2D numpy array*
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
categorical_columns=['protocol_type', 'service', 'flag']
```

```

df_categorical_values = df[categorical_columns]
testdf_categorical_values = df_test[categorical_columns]

df_categorical_values.head()
# protocol type
unique_protocol=sorted(df.protocol_type.unique())
string1 = 'Protocol_type_'
unique_protocol2=[string1 + x for x in unique_protocol]
print(unique_protocol2)

# service
unique_service=sorted(df.service.unique())
string2 = 'service_'
unique_service2=[string2 + x for x in unique_service]
print(unique_service2)

# flag
unique_flag=sorted(df.flag.unique())
string3 = 'flag_'
unique_flag2=[string3 + x for x in unique_flag]
print(unique_flag2)

# put together
dumcols=unique_protocol2 + unique_service2 + unique_flag2

#do it for test set
unique_service_test=sorted(df_test.service.unique())
unique_service2_test=[string2 + x for x in unique_service_test]
testdumcols=unique_protocol2 + unique_service2_test + unique_flag2

*Transform categorical features into numbers using LabelEncoder()*
df_categorical_values_enc=df_categorical_values.apply(LabelEncoder().fit_transform)

print(df_categorical_values.head())
print('-----')
print(df_categorical_values_enc.head())

# test set
testdf_categorical_values_enc=testdf_categorical_values.apply(LabelEncoder().fit_transform)
*One-Hot-Encoding*
enc = OneHotEncoder(categories='auto')
df_categorical_values_encenc = enc.fit_transform(df_categorical_values_enc)
df_cat_data = pd.DataFrame(df_categorical_values_encenc.toarray(),columns=dumcols)

# test set
testdf_categorical_values_encenc = enc.fit_transform(testdf_categorical_values_enc)
testdf_cat_data = pd.DataFrame(testdf_categorical_values_encenc.toarray(),columns=testdumcols)

df_cat_data.head()

trainservice=df['service'].tolist()
testservice= df_test['service'].tolist()
difference=list(set(trainservice) - set(testservice))
string = 'service_'
difference=[string + x for x in difference]
difference
for col in difference:
    testdf_cat_data[col] = 0

print(df_cat_data.shape)
print(testdf_cat_data.shape)

newdf=df.join(df_cat_data)
newdf.drop('flag', axis=1, inplace=True)
newdf.drop('protocol_type', axis=1, inplace=True)
newdf.drop('service', axis=1, inplace=True)

# test data
newdf_test=df_test.join(testdf_cat_data)
newdf_test.drop('flag', axis=1, inplace=True)
newdf_test.drop('protocol_type', axis=1, inplace=True)
newdf_test.drop('service', axis=1, inplace=True)

print(newdf.shape)
print(newdf_test.shape)

```

```

DoS :

Probe :

R2L :

U2R :
labeldf=newdf['label']
labeldf_test=newdf_test['label']

# change the label column
newlabeldf=labeldf.replace({ 'normal' : 0, 'neptune' : 1, 'back': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop':
1, 'mailbomb': 1, 'apache2': 1, 'processtable': 1, 'udpstorm': 1, 'worm': 1,
                        'ipsweep' : 2, 'nmap' : 2, 'portsweep' : 2, 'satan' : 2, 'mscan' : 2, 'saint' : 2
                        , 'ftp_write': 3, 'guess_passwd': 3, 'imap': 3, 'multihop': 3, 'phf': 3, 'spy': 3, 'warezclient':
3, 'warezmaster': 3, 'sendmail': 3, 'named': 3, 'snmpgetattack': 3, 'snmpguess': 3, 'xlock': 3, 'xsnoop': 3, 'httptunnel': 3,
                        'buffer_overflow': 4, 'loadmodule': 4, 'perl': 4, 'rootkit': 4, 'ps': 4, 'sqlattack': 4, 'xterm': 4})
newlabeldf_test=labeldf_test.replace({ 'normal' : 0, 'neptune' : 1, 'back': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop':
1, 'mailbomb': 1, 'apache2': 1, 'processtable': 1, 'udpstorm': 1, 'worm': 1,
                        'ipsweep' : 2, 'nmap' : 2, 'portsweep' : 2, 'satan' : 2, 'mscan' : 2, 'saint' : 2
                        , 'ftp_write': 3, 'guess_passwd': 3, 'imap': 3, 'multihop': 3, 'phf': 3, 'spy': 3, 'warezclient':
3, 'warezmaster': 3, 'sendmail': 3, 'named': 3, 'snmpgetattack': 3, 'snmpguess': 3, 'xlock': 3, 'xsnoop': 3, 'httptunnel': 3,
                        'buffer_overflow': 4, 'loadmodule': 4, 'perl': 4, 'rootkit': 4, 'ps': 4, 'sqlattack': 4, 'xterm': 4})

# put the new label column back
newdf['label'] = newlabeldf
newdf_test['label'] = newlabeldf_test
to_drop_DoS = [0,1]
to_drop_Probe = [0,2]
to_drop_R2L = [0,3]
to_drop_U2R = [0,4]

DoS_df=newdf[newdf['label'].isin(to_drop_DoS)];
Probe_df=newdf[newdf['label'].isin(to_drop_Probe)];
R2L_df=newdf[newdf['label'].isin(to_drop_R2L)];
U2R_df=newdf[newdf['label'].isin(to_drop_U2R)];

#test
DoS_df_test=newdf_test[newdf_test['label'].isin(to_drop_DoS)];
Probe_df_test=newdf_test[newdf_test['label'].isin(to_drop_Probe)];
R2L_df_test=newdf_test[newdf_test['label'].isin(to_drop_R2L)];
U2R_df_test=newdf_test[newdf_test['label'].isin(to_drop_U2R)];

print('Train:')
print('Dimensions of DoS:' ,DoS_df.shape)
print('Dimensions of Probe:' ,Probe_df.shape)
print('Dimensions of R2L:' ,R2L_df.shape)
print('Dimensions of U2R:' ,U2R_df.shape)
print()
print('Test:')
print('Dimensions of DoS:' ,DoS_df_test.shape)
print('Dimensions of Probe:' ,Probe_df_test.shape)
print('Dimensions of R2L:' ,R2L_df_test.shape)
print('Dimensions of U2R:' ,U2R_df_test.shape)
*Step 2: Feature Scaling*

X_DoS = DoS_df.drop('label', axis=1)
Y_DoS = DoS_df['label']

X_Probe = Probe_df.drop('label', axis=1)
Y_Probe = Probe_df['label']

X_R2L = R2L_df.drop('label', axis=1)
Y_R2L = R2L_df['label']

X_U2R = U2R_df.drop('label', axis=1)
Y_U2R = U2R_df['label']

```

```

# test set
X_DoS_test = DoS_df_test.drop('label', axis=1)
Y_DoS_test = DoS_df_test['label']

X_Probe_test = Probe_df_test.drop('label', axis=1)
Y_Probe_test = Probe_df_test['label']

X_R2L_test = R2L_df_test.drop('label', axis=1)
Y_R2L_test = R2L_df_test['label']

X_U2R_test = U2R_df_test.drop('label', axis=1)
Y_U2R_test = U2R_df_test['label']

colNames=list(X_DoS)
colNames_test=list(X_DoS_test)
from sklearn import preprocessing

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)

# test data
scaler5 = preprocessing.StandardScaler().fit(X_DoS_test)
X_DoS_test=scaler5.transform(X_DoS_test)

scaler6 = preprocessing.StandardScaler().fit(X_Probe_test)
X_Probe_test=scaler6.transform(X_Probe_test)

scaler7 = preprocessing.StandardScaler().fit(X_R2L_test)
X_R2L_test=scaler7.transform(X_R2L_test)

scaler8 = preprocessing.StandardScaler().fit(X_U2R_test)
X_U2R_test=scaler8.transform(X_U2R_test)
*Step 3: Feature Selection:*

---

# Random Forest
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=10, n_jobs=2, class_weight='balanced')
rfe = RFE(estimator=clf, n_features_to_select=13, step=1)

rfe.fit(X_DoS, Y_DoS.astype(int))
X_rfeDoS = rfe.transform(X_DoS)
true = rfe.support_
rfecolindex_DoS = [i for i, x in enumerate(true) if x]
rfecolname_DoS = [colNames[i] for i in rfecolindex_DoS]

rfe.fit(X_Probe, Y_Probe.astype(int))
X_rfeProbe=rfe.transform(X_Probe)
true=rfe.support_
rfecolindex_Probe=[i for i, x in enumerate(true) if x]
rfecolname_Probe=list(colNames[i] for i in rfecolindex_Probe)

rfe.fit(X_R2L, Y_R2L.astype(int))
X_rfeR2L=rfe.transform(X_R2L)
true=rfe.support_
rfecolindex_R2L=[i for i, x in enumerate(true) if x]
rfecolname_R2L=list(colNames[i] for i in rfecolindex_R2L)
rfe.fit(X_U2R, Y_U2R.astype(int))
X_rfeU2R=rfe.transform(X_U2R)
true=rfe.support_
rfecolindex_U2R=[i for i, x in enumerate(true) if x]
rfecolname_U2R=list(colNames[i] for i in rfecolindex_U2R)
*Summary of features selected by RFE*
print('Features selected for DoS:',rfecolname_DoS)

```



```

print()
print('Features selected for Probe:',rfecolname_Probe)
print()
print('Features selected for R2L:',rfecolname_R2L)
print()
print('Features selected for U2R:',rfecolname_U2R)

```

```

print(X_rfeDoS.shape)
print(X_rfeProbe.shape)
print(X_rfeR2L.shape)
print(X_rfeU2R.shape)

```

Step 4: Build the model:

Classifier is trained for all features and for reduced features, for later comparison.

The classifier model itself is stored in the clf variable.

```

# all features
clf_DoS=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_Probe=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_R2L=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_U2R=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_DoS.fit(X_DoS, Y_DoS.astype(int))
clf_Probe.fit(X_Probe, Y_Probe.astype(int))
clf_R2L.fit(X_R2L, Y_R2L.astype(int))
clf_U2R.fit(X_U2R, Y_U2R.astype(int))
# selected features
clf_rfeDoS=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_rfeProbe=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_rfeR2L=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_rfeU2R=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_rfeDoS.fit(X_rfeDoS, Y_DoS.astype(int))
clf_rfeProbe.fit(X_rfeProbe, Y_Probe.astype(int))
clf_rfeR2L.fit(X_rfeR2L, Y_R2L.astype(int))
clf_rfeU2R.fit(X_rfeU2R, Y_U2R.astype(int))
*Step 5: Prediction & Evaluation (validation):*

```

Using all Features for each category

Confusion Matrices

```

DoS¶
# Apply the classifier we trained to the test data
clf_DoS.predict(X_DoS_test)

# View the predicted probabilities of the first 10 observations
clf_DoS.predict_proba(X_DoS_test)[0:10]
Y_DoS_pred=clf_DoS.predict(X_DoS_test)

# Create confusion matrix
pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
*Probe*
Y_Probe_pred=clf_Probe.predict(X_Probe_test)
# Create confusion matrix

pd.crosstab(Y_Probe_test, Y_Probe_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
*R2L*
Y_R2L_pred=clf_R2L.predict(X_R2L_test)
# Create confusion matrix
pd.crosstab(Y_R2L_test, Y_R2L_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
*U2R*
Y_U2R_pred=clf_U2R.predict(X_U2R_test)
# Create confusion matrix
pd.crosstab(Y_U2R_test, Y_U2R_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
*Cross Validation: Accuracy, Precision, Recall, F-measure*
*DoS*
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')

```

```

print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

*Probe*
accuracy = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

*U2R*
accuracy = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

*R2L*
accuracy = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

*Using 13 Features for each category*

```

Confusion Matrices

```

DoS
# reduce test dataset to 13 features, use only features described in rfecolname_DoS etc.
X_DoS_test2=X_DoS_test[:,rfecolindex_DoS]
X_Probe_test2=X_Probe_test[:,rfecolindex_Probe]
X_R2L_test2=X_R2L_test[:,rfecolindex_R2L]
X_U2R_test2=X_U2R_test[:,rfecolindex_U2R]
X_U2R_test2.shape
Y_DoS_pred2=clf_rfeDoS.predict(X_DoS_test2)
# Create confusion matrix
pd.crosstab(Y_DoS_test, Y_DoS_pred2, rownames=['Actual attacks'], colnames=['Predicted attacks'])

*Probe*
Y_Probe_pred2=clf_rfeProbe.predict(X_Probe_test2)
# Create confusion matrix
pd.crosstab(Y_Probe_test, Y_Probe_pred2, rownames=['Actual attacks'], colnames=['Predicted attacks'])

*R2L*
Y_R2L_pred2=clf_rfeR2L.predict(X_R2L_test2)
# Create confusion matrix
pd.crosstab(Y_R2L_test, Y_R2L_pred2, rownames=['Actual attacks'], colnames=['Predicted attacks'])

*U2R*
Y_U2R_pred2=clf_rfeU2R.predict(X_U2R_test2)
# Create confusion matrix
pd.crosstab(Y_U2R_test, Y_U2R_pred2, rownames=['Actual attacks'], colnames=['Predicted attacks'])
*Cross Validation: Accuracy, Precision, Recall, F-measure*

*DoS*
accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

*Probe*
accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='accuracy')

```

```

print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

```

```

*R2L*
accuracy = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

```

```

*U2R*
accuracy = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

```

```

# KNeighbors

```

```

from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier

```

```

clf_KNN_DoS = KNeighborsClassifier()
clf_KNN_Probe = KNeighborsClassifier()
clf_KNN_R2L = KNeighborsClassifier()
clf_KNN_U2R = KNeighborsClassifier()

```

```

# Fit the KNeighborsClassifier instances before using them in the CalibratedClassifierCV

```

```

clf_KNN_DoS.fit(X_DoS, Y_DoS.astype(int))
clf_KNN_Probe.fit(X_Probe, Y_Probe.astype(int))
clf_KNN_R2L.fit(X_R2L, Y_R2L.astype(int))
clf_KNN_U2R.fit(X_U2R, Y_U2R.astype(int))

```

```

calibrated_clf_KNN_DoS = CalibratedClassifierCV(clf_KNN_DoS, method='sigmoid', cv='prefit')
calibrated_clf_KNN_DoS.fit(X_DoS, Y_DoS.astype(int))

```

```

*DoS*
Y_DoS_pred=clf_KNN_DoS.predict(X_DoS_test)

```

```

# Create confusion matrix
pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
*Probe*
Y_Probe_pred=clf_KNN_Probe.predict(X_Probe_test)
# Create confusion matrix

```

```

pd.crosstab(Y_Probe_test, Y_Probe_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])

```

```

*R2L*
Y_R2L_pred=clf_KNN_R2L.predict(X_R2L_test)

```

```

# Create confusion matrix
pd.crosstab(Y_R2L_test, Y_R2L_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])

```

```

*U2R*
Y_U2R_pred=clf_KNN_U2R.predict(X_U2R_test)

```

```

# Create confusion matrix
pd.crosstab(Y_U2R_test, Y_U2R_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])

```

```

*Cross Validation: Accuracy, Precision, Recall, F-measure*

```

```

*DoS*
from sklearn.model_selection import cross_val_score
from sklearn import metrics

```

```

accuracy = cross_val_score(clf_KNN_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

```

```

*Probe*
accuracy = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')

```

```

print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
*R2L*
accuracy = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
*U2R*
accuracy = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

# SVM

from sklearn.svm import SVC

clf_SVM_DoS=SVC(kernel='linear', C=1.0, random_state=0)
clf_SVM_Probe=SVC(kernel='linear', C=1.0, random_state=0)
clf_SVM_R2L=SVC(kernel='linear', C=1.0, random_state=0)
clf_SVM_U2R=SVC(kernel='linear', C=1.0, random_state=0)

clf_SVM_DoS.fit(X_DoS, Y_DoS.astype(int))
clf_SVM_Probe.fit(X_Probe, Y_Probe.astype(int))
clf_SVM_R2L.fit(X_R2L, Y_R2L.astype(int))
clf_SVM_U2R.fit(X_U2R, Y_U2R.astype(int))
*DoS*
Y_DoS_pred=clf_SVM_DoS.predict(X_DoS_test)

# Create confusion matrix
pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
Y_Probe_pred=clf_SVM_Probe.predict(X_Probe_test)
# Create confusion matrix

pd.crosstab(Y_Probe_test, Y_Probe_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
Y_R2L_pred=clf_SVM_R2L.predict(X_R2L_test)
# Create confusion matrix
pd.crosstab(Y_R2L_test, Y_R2L_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
Y_U2R_pred=clf_SVM_U2R.predict(X_U2R_test)
# Create confusion matrix
pd.crosstab(Y_U2R_test, Y_U2R_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
*DoS*
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_SVM_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_SVM_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_SVM_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_SVM_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
*Probe*
accuracy = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
*R2L*
accuracy = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))

```

```

f = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
*U2R*
accuracy = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
# Ensemble Learning
from sklearn.ensemble import VotingClassifier

clf_voting_DoS = VotingClassifier(estimators=[('rf', clf_DoS), ('knn', clf_KNN_DoS), ('svm', clf_SVM_DoS)], voting='hard')
clf_voting_Probe = VotingClassifier(estimators=[('rf', clf_Probe), ('knn', clf_KNN_Probe), ('svm', clf_SVM_Probe)],
voting='hard')
clf_voting_R2L = VotingClassifier(estimators=[('rf', clf_R2L), ('knn', clf_KNN_R2L), ('svm', clf_SVM_R2L)], voting='hard')
clf_voting_U2R = VotingClassifier(estimators=[('rf', clf_U2R), ('knn', clf_KNN_U2R), ('svm', clf_SVM_U2R)], voting='hard')

clf_voting_DoS.fit(X_DoS, Y_DoS.astype(int))
clf_voting_Probe.fit(X_Probe, Y_Probe.astype(int))
clf_voting_R2L.fit(X_R2L, Y_R2L.astype(int))
clf_voting_U2R.fit(X_U2R, Y_U2R.astype(int))

*DoS*
Y_DoS_pred=clf_voting_DoS.predict(X_DoS_test)

# Create confusion matrix
pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
*Probe*
Y_Probe_pred=clf_voting_Probe.predict(X_Probe_test)

# Create confusion matrix
pd.crosstab(Y_Probe_test, Y_Probe_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
*R2L*
Y_R2L_pred=clf_voting_R2L.predict(X_R2L_test)

# Create confusion matrix
pd.crosstab(Y_R2L_test, Y_R2L_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
*U2R*
Y_U2R_pred=clf_voting_U2R.predict(X_U2R_test)

# Create confusion matrix
pd.crosstab(Y_U2R_test, Y_U2R_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
*DoS*
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
*Probe*
accuracy = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
*R2L*
accuracy = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
*U2R*
accuracy = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))

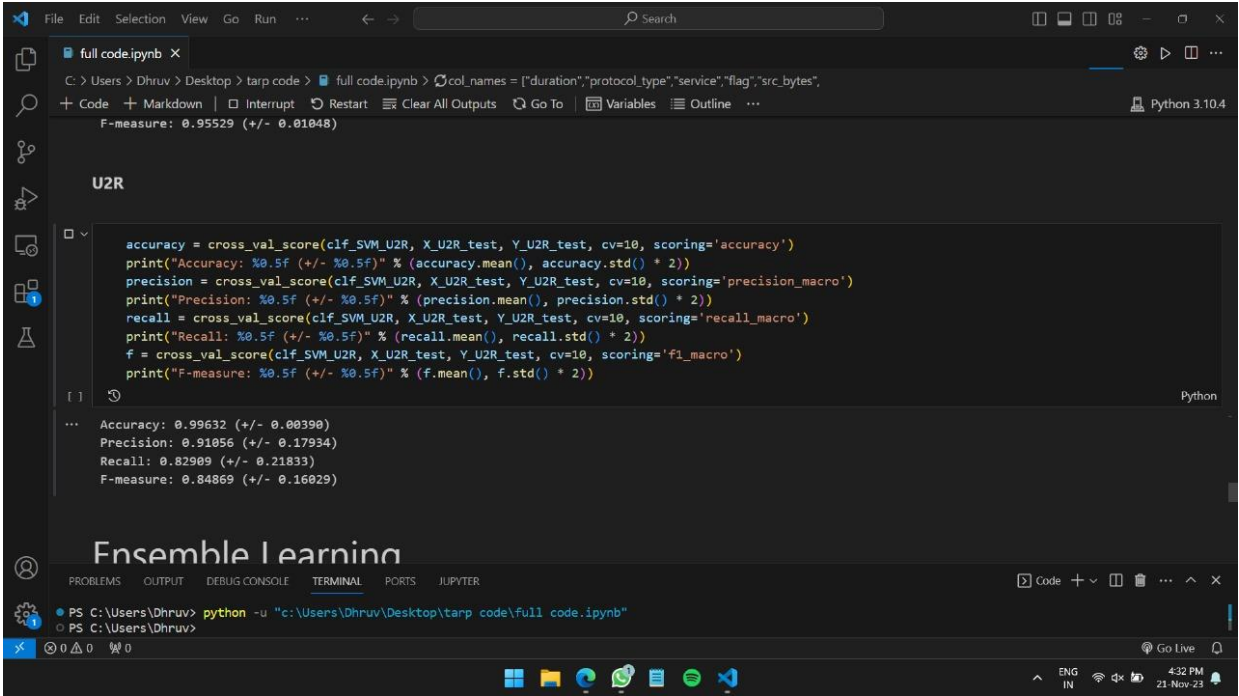
```

```
recall = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

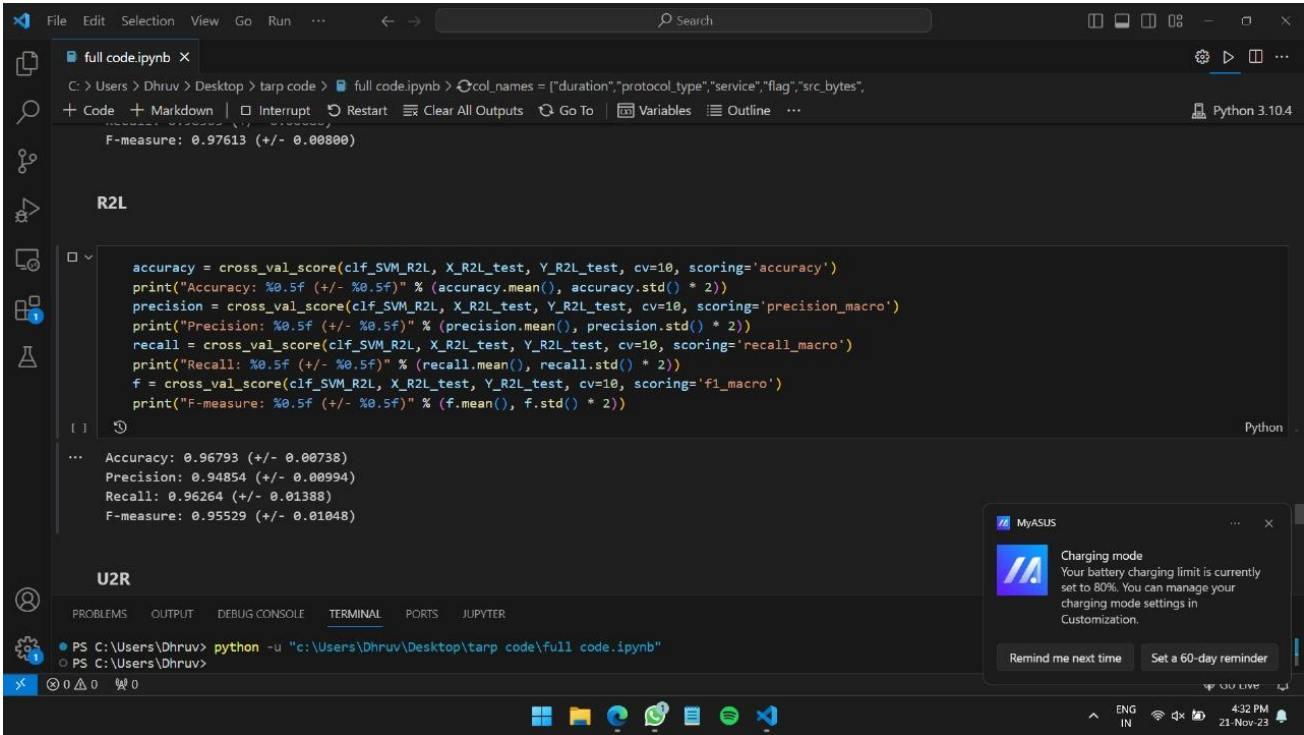
4.4 Test Cases

Support Vector Machine Algo tested against

a. U2R



b. R2L



c. DoS

full code.ipynb X

C:\Users\Dhruv\Desktop\tarp code> full code.ipynb > col_names = ["duration","protocol_type","service","flag","src_bytes",

+ Code + Markdown | Interrupt Restart Clear All Outputs Go To Variables Outline ... Python 3.10.4

DoS

```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_SVM_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_SVM_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_SVM_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_SVM_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

Accuracy: 0.99371 (+/- 0.00375)
Precision: 0.99107 (+/- 0.00785)
Recall: 0.99450 (+/- 0.00388)
F-measure: 0.99278 (+/- 0.00428)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

PS C:\Users\Dhruv> python -u "c:\Users\Dhruv\Desktop\tarp code\full code.ipynb"
PS C:\Users\Dhruv>

Go Live 4:32 PM 21-Nov-23

d. Probe

full code.ipynb X

C:\Users\Dhruv\Desktop\tarp code> full code.ipynb > col_names = ["duration","protocol_type","service","flag","src_bytes",

+ Code + Markdown | Interrupt Restart Clear All Outputs Go To Variables Outline ... Python 3.10.4

Probe

```
accuracy = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

Accuracy: 0.98450 (+/- 0.00526)
Precision: 0.96907 (+/- 0.01031)
Recall: 0.98365 (+/- 0.00686)
F-measure: 0.97613 (+/- 0.00800)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

PS C:\Users\Dhruv> python -u "c:\Users\Dhruv\Desktop\tarp code\full code.ipynb"
PS C:\Users\Dhruv>

Go Live 4:32 PM 21-Nov-23

ENSEMBLE LEARNING ALGO tested against:

a. U2R

full code.ipynb X

C:\Users> Dhruv > Desktop > tarp code > full code.ipynb > col_names = ["duration","protocol_type","service","flag","src_bytes",

+ Code + Markdown | Interrupt Restart Clear All Outputs Go To Variables Outline ... Python 3.10.4

Recall: 0.96220 (+/- 0.01435)
F-measure: 0.96133 (+/- 0.00777)

U2R

(variable) X_U2R_test: ndarray | spmatrix

accuracy = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Python

... Accuracy: 0.99734 (+/- 0.00306)
Precision: 0.94328 (+/- 0.12453)
Recall: 0.87345 (+/- 0.17282)
F-measure: 0.90677 (+/- 0.09408)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

PS C:\Users\Dhruv> python -u "c:\Users\Dhruv\Desktop\tarp code\full code.ipynb"
PS C:\Users\Dhruv>

Go Live 4:33 PM 21-Nov-23

b. R2L

full code.ipynb X

C:\Users> Dhruv > Desktop > tarp code > full code.ipynb > col_names = ["duration","protocol_type","service","flag","src_bytes",

+ Code + Markdown | Interrupt Restart Clear All Outputs Go To Variables Outline ... Python 3.10.4

Precision: 0.98781 (+/- 0.00739)
Recall: 0.98953 (+/- 0.00732)
F-measure: 0.98854 (+/- 0.00571)

R2L

+ Code + Markdown

accuracy = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Python

... Accuracy: 0.97213 (+/- 0.00664)
Precision: 0.95774 (+/- 0.00919)
Recall: 0.96220 (+/- 0.01435)
F-measure: 0.96133 (+/- 0.00777)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

PS C:\Users\Dhruv> python -u "c:\Users\Dhruv\Desktop\tarp code\full code.ipynb"
PS C:\Users\Dhruv>

Go Live 4:33 PM 21-Nov-23

c. DoS

File Edit Selection View Go Run ... Search

full code.ipynb X

C:\Users> Dhruv > Desktop > tarp code > full code.ipynb > col_names = ["duration","protocol_type","service","flag","src_bytes",

+ Code + Markdown | Interrupt Restart Clear All Outputs Go To Variables Outline ... Python 3.10.4

DoS

```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

(variable) Y_DoS_test: Series

Accuracy: 0.99802 (+/- 0.00182)
Precision: 0.99839 (+/- 0.00313)
Recall: 0.99692 (+/- 0.00318)
F-measure: 0.99772 (+/- 0.00249)

Probe

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

PS C:\Users\Dhruv> python -u "c:\Users\Dhruv\Desktop\tarp code\full code.ipynb"

PS C:\Users\Dhruv>

Go Live

ENG IN 4:33 PM 21-Nov-23

d. PROBE

File Edit Selection View Go Run ... Search

full code.ipynb X

C:\Users> Dhruv > Desktop > tarp code > full code.ipynb > col_names = ["duration","protocol_type","service","flag","src_bytes",

+ Code + Markdown | Interrupt Restart Clear All Outputs Go To Variables Outline ... Python 3.10.4

Probe

```
accuracy = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

Accuracy: 0.99291 (+/- 0.00355)
Precision: 0.98781 (+/- 0.00739)
Recall: 0.98953 (+/- 0.00732)
F-measure: 0.98854 (+/- 0.00571)

R2L

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

PS C:\Users\Dhruv> python -u "c:\Users\Dhruv\Desktop\tarp code\full code.ipynb"

PS C:\Users\Dhruv>

Go Live

ENG IN 4:33 PM 21-Nov-23

KNN ALGO tested against:

a. U2R

File Edit Selection View Go Run ...

Search

Python 3.10.4

full code.ipynb X

C:\Users\Dhruv\Desktop> tarp code > full code.ipynb > col_names = ["duration","protocol_type","service","flag","src_bytes",

+ Code + Markdown | Interrupt Restart Clear All Outputs Go To Variables Outline ...

Precision: 0.95312 (+/- 0.01230)
Recall: 0.95454 (+/- 0.01363)
F-measure: 0.95376 (+/- 0.01052)

U2R

accuracy = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

[56] ✓ 12s Python

... Accuracy: 0.99703 (+/- 0.00281)
Precision: 0.93143 (+/- 0.14679)
Recall: 0.85073 (+/- 0.17639)
F-measure: 0.87831 (+/- 0.11390)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

PS C:\Users\Dhruv> python -u "c:\Users\Dhruv\Desktop\tarp code\full code.ipynb"
PS C:\Users\Dhruv>

Go Live

ENG IN 4:34 PM 21-Nov-23

b. R2L

File Edit Selection View Go Run ...

Search

Python 3.10.4

full code.ipynb X

C:\Users\Dhruv\Desktop> tarp code > full code.ipynb > col_names = ["duration","protocol_type","service","flag","src_bytes",

+ Code + Markdown | Interrupt Restart Clear All Outputs Go To Variables Outline ...

R2L

accuracy = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

[55] ✓ 4.8s Python

... Accuracy: 0.96729 (+/- 0.00741)
Precision: 0.95312 (+/- 0.01230)
Recall: 0.95454 (+/- 0.01363)
F-measure: 0.95376 (+/- 0.01052)

U2R

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

PS C:\Users\Dhruv> python -u "c:\Users\Dhruv\Desktop\tarp code\full code.ipynb"
PS C:\Users\Dhruv>

Go Live

ENG IN 4:34 PM 21-Nov-23

c. DoS

File Edit Selection View Go Run ... Search

full code.ipynb X

C:\Users> Dhruv> Desktop> tarp code> full code.ipynb col_names = ["duration","protocol_type","service","flag","src_bytes",

+ Code + Markdown Interrupt Restart Clear All Outputs Go To Variables Outline Python 3.10.4

DoS

```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_KNN_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

[54] ✓ 8.1s Python

... Accuracy: 0.99715 (+/- 0.00278)
Precision: 0.99678 (+/- 0.00383)
Recall: 0.99665 (+/- 0.00344)
F-measure: 0.99672 (+/- 0.00320)

Probe

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

PS C:\Users\Dhruv> python -u "c:\Users\Dhruv\Desktop\tarp code\full code.ipynb"
PS C:\Users\Dhruv>

Go Live 4:34 PM 21 Nov 23

d. PROBE

File Edit Selection View Go Run ... Search

full code.ipynb X

C:\Users> Dhruv> Desktop> tarp code> full code.ipynb col_names = ["duration","protocol_type","service","flag","src_bytes",

+ Code + Markdown Interrupt Restart Clear All Outputs Go To Variables Outline Python 3.10.4

Probe

```
accuracy = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

[54] ✓ 4.3s Python

... Accuracy: 0.99077 (+/- 0.00403)
Precision: 0.98606 (+/- 0.00675)
Recall: 0.98508 (+/- 0.01137)
F-measure: 0.98553 (+/- 0.00645)

R2L

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

PS C:\Users\Dhruv> python -u "c:\Users\Dhruv\Desktop\tarp code\full code.ipynb"
PS C:\Users\Dhruv>

Go Live 4:34 PM 21 Nov 23

4.5 CLOUD DEPLOYMENT

The model has been deployed on AWS:

Sign in

How it works | Amazon SageMaker

full code.ipynb - JupyterLab

(23) Ten Minute Tutorial for Res...

https://d-tr4goups7x5m.studio.ap-south-1.sagemaker.aws/jupyter/default/lab/tree/full%20code.ipynb

Amazon SageMaker Studio

File Edit View Run Kernel Git Tabs Settings Help

Home Launcher full code.ipynb

Cluster Data Science 3.0 Python 3 2 vCPU + 4 GiB Share

Filter files by name

Name

full code.ipynb

Testing.csv

Training.csv

Untitled.ipynb

untitled.py

Full code.ipynb

DoS

44: accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99720 (+/- 0.00329)
Precision: 0.99853 (+/- 0.00368)
Recall: 0.99598 (+/- 0.00634)
F-measure: 0.99705 (+/- 0.00295)

Probe

45: accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99464 (+/- 0.00419)
Precision: 0.99145 (+/- 0.00713)
Recall: 0.98880 (+/- 0.01026)
F-measure: 0.99041 (+/- 0.00823)

Simple 0 2 Python 3 (Data Science 3.0) | Idle Kernel: Idle | Instance MEM Mode: Command Ln 1, Col 1 full code.ipynb 0 11:40 PM 21-Nov-23

Sign in

How it works | Amazon SageMaker

full code.ipynb - JupyterLab

(23) Ten Minute Tutorial for Res...

https://d-tr4goups7x5m.studio.ap-south-1.sagemaker.aws/jupyter/default/lab/tree/full%20code.ipynb

Amazon SageMaker Studio

File Edit View Run Kernel Git Tabs Settings Help

Home Launcher full code.ipynb

Cluster Data Science 3.0 Python 3 2 vCPU + 4 GiB Share

Filter files by name

Name

full code.ipynb

Testing.csv

Training.csv

Untitled.ipynb

untitled.py

Full code.ipynb

R2L

46: accuracy = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.97626 (+/- 0.00440)
Precision: 0.96625 (+/- 0.01321)
Recall: 0.96515 (+/- 0.01290)
F-measure: 0.96613 (+/- 0.00859)

U2R

47: accuracy = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99540 (+/- 0.00246)
Precision: 0.95679 (+/- 0.17149)
Recall: 0.76294 (+/- 0.15737)
F-measure: 0.79332 (+/- 0.19349)

KNeighbors

Simple 0 2 Python 3 (Data Science 3.0) | Idle Kernel: Idle | Instance MEM Mode: Command Ln 1, Col 1 full code.ipynb 0 11:40 PM 21-Nov-23

5. Results and Discussion

5.1 Implementation Results

After training and testing our IDS model against a large dataset, we have achieved the following results w.r.t the different algorithms used:

Support Vector Machine Algorithm:

This algorithm produced the following results when tasked with identifying the following types of attacks:

1. DoS (Denial of Service):

The algorithm produced an accuracy of 99.3% with a precision of 99.1% and a recall of 99.4%. The F-Measure for the algorithm is: 99.2%.

2. U2R (User to Root):

The algorithm produced an accuracy of 99.6% with a precision of 91.0% and a recall of 82.9%. The F-Measure for the algorithm is: 84%.

3. R2L (Remote to Local):

The algorithm produced an accuracy of 96.7% with a precision of 94.8% and a recall of 96.2%. The F-Measure for the algorithm is: 95.5%.

4. PROBE:

The algorithm produced an accuracy of 98.5% with a precision of 97% and a recall of 98.3%. The F-Measure for the algorithm is: 97%.

Ensemble Algorithm:

This algorithm produced the following results when tasked with identifying the following types of attacks:

1. DoS (Denial of Service):

The algorithm produced an accuracy of 99.8% with a precision of 99.8% and a recall of 99.6%. The F-Measure for the algorithm is: 99.7%.

2. U2R (User to Root):

The algorithm produced an accuracy of 99.7% with a precision of 94.3% and a recall of 87.3%. The F-Measure for the algorithm is: 91%.

3. R2L (Remote to Local):

The algorithm produced an accuracy of 97.2% with a precision of 95.7% and a recall of 96.2%. The F-Measure for the algorithm is: 96.1%.

4. PROBE:

The algorithm produced an accuracy of 99.2% with a precision of 98.7% and a recall of 98.9%. The F-Measure for the algorithm is: 98.8%.

K Nearest Neighbours:

This algorithm produced the following results when tasked with identifying the following types of attacks:

1. DoS (Denial of Service):

The algorithm produced an accuracy of 99.7% with a precision of 99.6% and a recall of 99.6%. The F-Measure for the algorithm is: 99.6%.

2. U2R (User to Root):

The algorithm produced an accuracy of 99.7% with a precision of 93.1% and a recall of 85.1%. The F-Measure for the algorithm is: 87%.

3. R2L (Remote to Local):

The algorithm produced an accuracy of 96.7% with a precision of 95.3% and a recall of 95.4%. The F-Measure for the algorithm is: 95.4%.

4. PROBE:

The algorithm produced an accuracy of 99.1% with a precision of 98.6% and a recall of 98.5%. The F-Measure for the algorithm is: 98.5%.

In evaluating the intrusion detection system using different algorithms, it is evident that each algorithm performs well in identifying specific types of attacks. The Support Vector Machine (SVM) algorithm demonstrates high accuracy across all attack categories, particularly excelling in DoS detection. The Ensemble algorithm showcases a balanced performance, achieving notable accuracy and F-measure values for each attack type. K Nearest Neighbours (KNN) also exhibits strong accuracy, especially in DoS detection, although its precision and recall in U2R and R2L are comparatively lower. The choice of the most suitable algorithm depends on the specific priorities of the intrusion detection system, such as emphasizing precision, recall, or a balanced F-measure. Overall, these results highlight the importance of considering the nature of attacks and the corresponding algorithm strengths when designing an effective intrusion detection system.

5.2 Metrics

We evaluate the performance of the IDS based on the following metrics:

- Accuracy: Defined as the percentage of correctly classified records over the total number of records.
- Precision (P): Defined as the % ratio of the number of true positives (TP) records divided by the number of true positives (TP) and false positives (FP) classified records

$$P = \frac{TP}{(TP + FP)} \times 100\%$$

- Recall (R): Defined as the % ratio of number of true positives records divided by the number of true positives and false negatives (FN) classified records. $R = \frac{TP}{TP + FN} \times 100\%$ (5)

$$R = \frac{TP}{(TP + FN)} \times 100\%$$

- F-Measure (F): Defined as the harmonic mean of precision and recall and represents a balance between them. $F = \frac{2 \cdot P \cdot R}{(P + R)}$

$$F = \frac{2 \cdot P \cdot R}{(P + R)}$$

5.3 Results in table form

	U2R	R2L	DOS	PROBE
SVM	99.6%	96.7%	99.3%	98.5%
Ensemble	99.7%	97.2%	99.8%	99.2%
KNN	99.7%	96.7%	99.7%	99.1%

Precision Scores

	U2R	R2L	DOS	PROBE
SVM	91%	94.8%	99.1%	97%
Ensemble	94.3%	95.7%	99.8%	98.7%
KNN	93.1%	95.3%	99.6%	98.6%

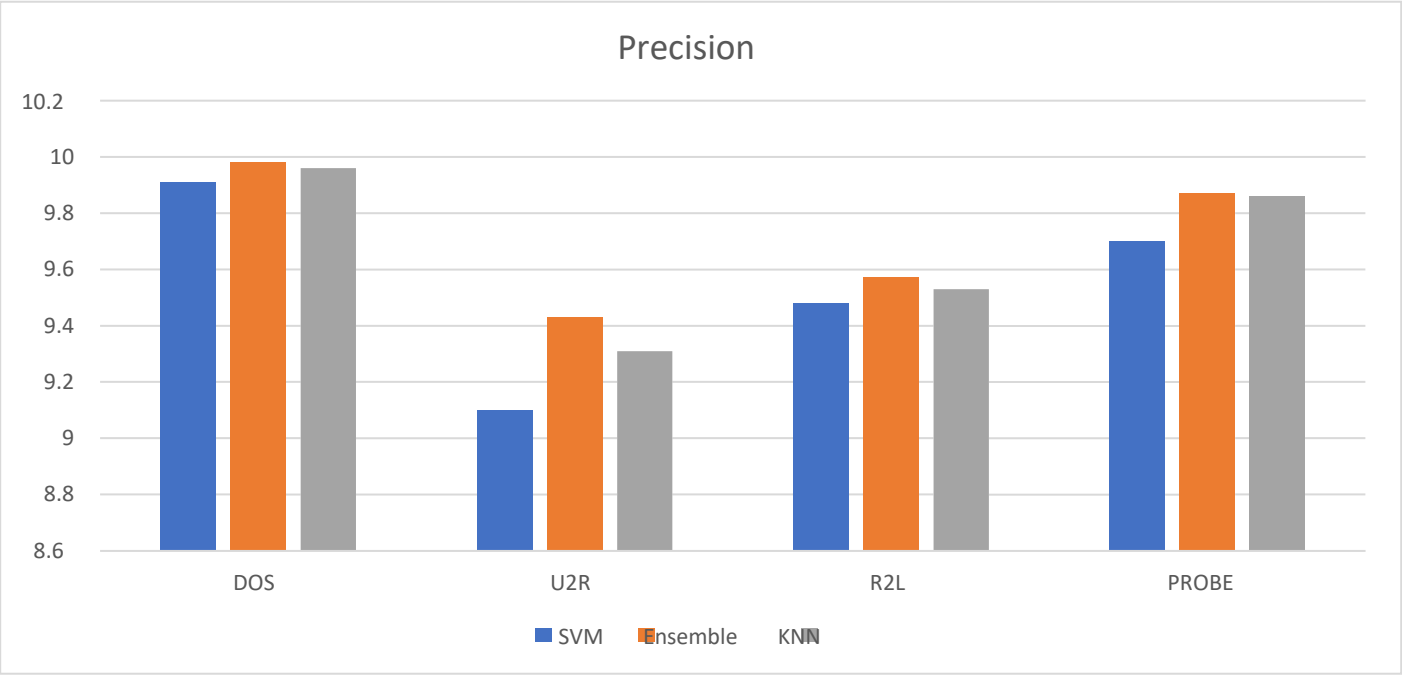
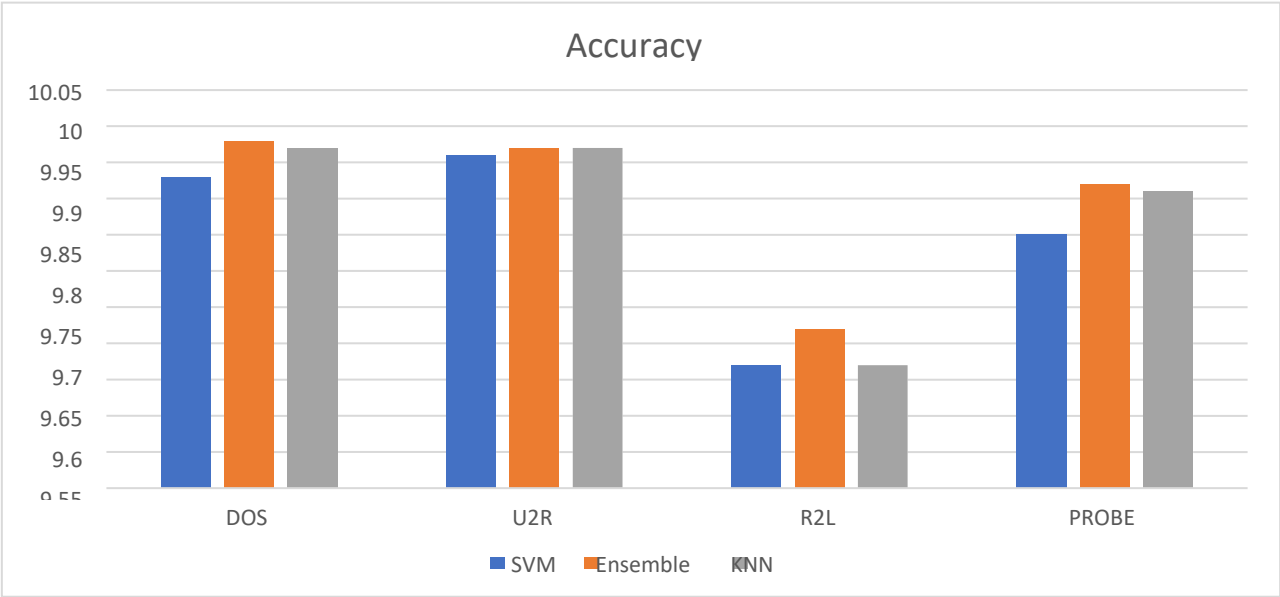
Recall Scores

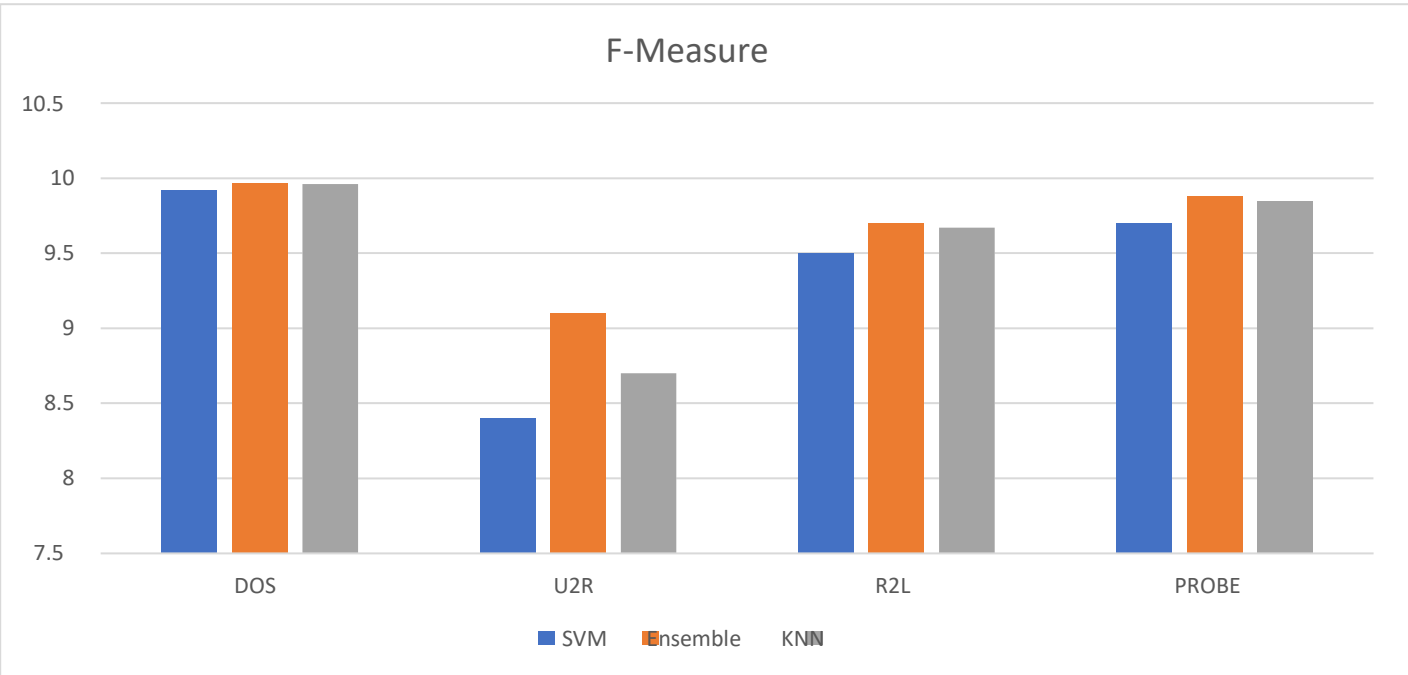
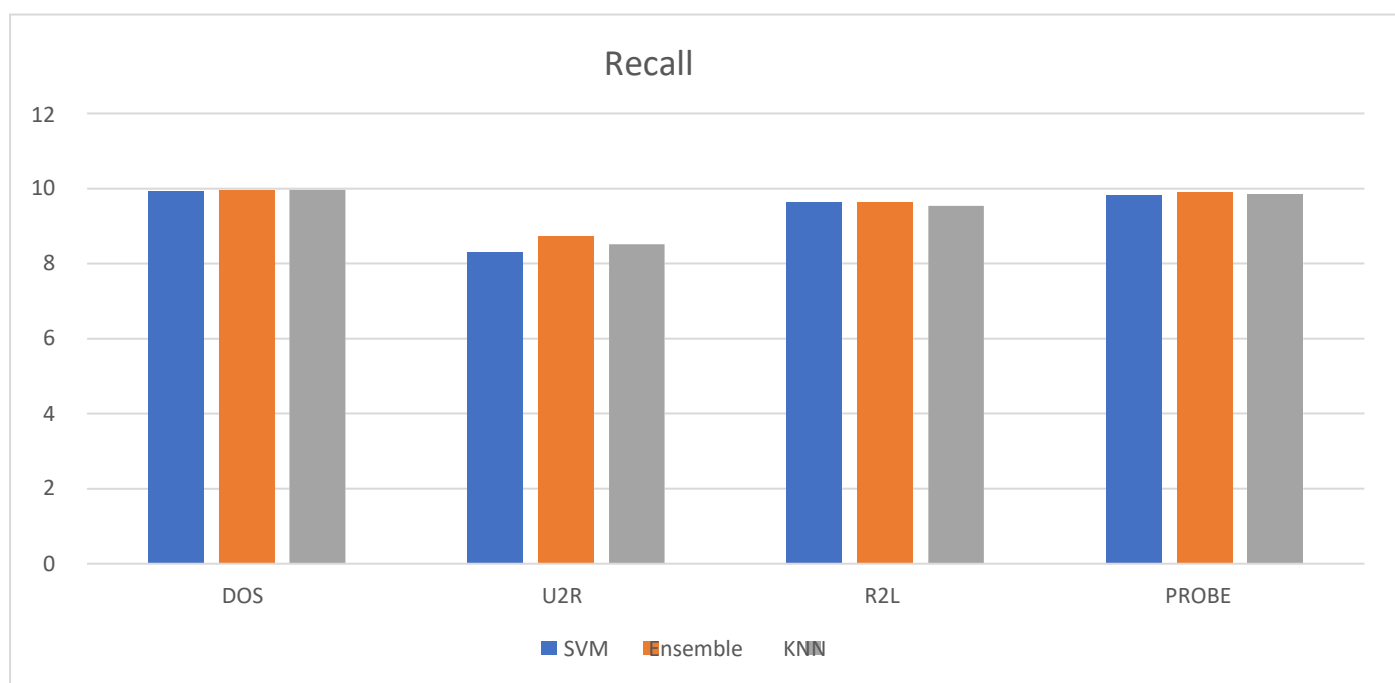
	U2R	R2L	DOS	PROBE
SVM	82.9%	96.2%	99.4%	98.3%
Ensemble	87.3%	96.2%	99.6%	98.9%
KNN	85.1%	95.4%	99.6%	98.5%

F-Measure Scores

	U2R	R2L	DOS	PROBE
SVM	84%	95.5%	99.2%	97%
Ensemble	91%	96.1%	99.7%	98.8%
KNN	87%	95.4%	99.6%	98.5%

5.4 Graphs (measured out of 10)





5.5 Mapping the results with problem statement

Whilst developing our IDS-we had to keep in mind our original goal of satisfying our problem statement and meeting societal needs. Our project addresses critical societal needs by deploying an Intrusion Detection System (IDS) aimed at enhancing cybersecurity universally, transcending age and gender barriers. It safeguards digital assets, particularly benefiting IT and finance sectors, by thwarting network attacks. Furthermore, it lays the groundwork for secure public services, envisioning a future where individuals can connect to potentially insecure networks without compromising data security. The IDS contributes to privacy protection by monitoring and identifying network threats, making cybersecurity more accessible for the average user. It plays a pivotal role in advancing Research and Development (R&D) efforts, providing proactive defense and valuable data for studying cyber threats. Additionally, our project facilitates tailored cybersecurity insurance policies, enhances digital infrastructure

resilience, promotes educational outreach on cybersecurity, encourages collaboration, ensures adaptability to future threats, and serves as a prototype for ongoing improvements, making it a comprehensive and impactful cybersecurity solution

5.6 Comparison with existing system

In order to find areas to improve our model-we need to find an existing paper/system. For this purpose, we have chosen the model used in the following research paper:

A Deep Learning Approach for Network Intrusion Detection System
by Quamar Niyaz, Weiqing Sun, Ahmad Y Javaid, and Mansoor Alam

In this paper, an IDS model is developed using **Self-taught learning** algorithm. Self-taught learning is a deep learning approach with two stages for classification. In the first stage, it learns a good feature representation from a large collection of unlabeled data (Unsupervised Feature Learning). This is typically done using techniques like Sparse Autoencoder, Restricted Boltzmann Machine (RBM), K-Means Clustering, or Gaussian Mixtures. The second stage involves applying this learned representation to labeled data for the actual classification task. In this project, a sparse autoencoder is used for feature learning. It's a neural network with input, hidden, and output layers, optimizing weight matrices and bias vectors using back-propagation. Soft-max regression is then employed for classification.

The following table tabulates the differences between the ensemble learning algorithm used in our model and the self-taught learning algorithm

Aspect	Feature Representation	Model Combination
Objective	Learn feature representation from unlabeled data.	Combine multiple models for improved performance.
Learning Approach	Deep learning, often involving autoencoders.	Combining multiple base models or learners.
Training Data	Large collection of unlabeled data (xu).	Labeled data for individual base learners.
Stages	Two stages: Unsupervised Feature Learning (UFL) and Classification.	Single or multiple stages depending on the ensemble type (e.g., boosting, bagging).
Data Distribution	Unlabeled and labeled data may come from different distributions, but there must be relevance among them.	Labeled data may come from the same distribution.
Feature Learning Method	Sparse Autoencoder is commonly used.	Various base learners, e.g., decision trees, SVMs, neural networks.
Training Process	Learns optimal feature representation from unlabeled data through back-propagation.	Trains multiple models independently or sequentially.
Model Diversity	Focuses on learning diverse and meaningful features from the data.	Relies on diverse base learners to contribute unique insights.
Output Layer Activation	Typically involves a softmax regression layer for classification.	Depends on the base learner; can be different activation functions.
Performance on Novelty	Well-suited for detecting unknown and new attacks.	Depends on the diversity of base models; can handle novel patterns.
Challenge	Proper feature selection from the network traffic dataset for anomaly detection is difficult. Unavailability of labeled traffic dataset.	Selection and combination of diverse models, dealing with overfitting, and ensuring ensemble diversity.
Accuracy (2-class)	>98% for all types of classification.	2-class classification accuracy: DoS (99.8%), U2R (99.7%), R2L (97.2%), PROBE (99.2%).

Precision (2-class)	Better values compared to SMR; e.g., 98.84% for f-measure.	Precision: DoS (99.8%), U2R (94.3%), R2L (95.7%), PROBE (98.7%).
Recall (2-class)	Better values compared to SMR.	Recall: DoS (99.6%), U2R (87.3%), R2L (96.2%), PROBE (98.9%).
F-Measure (2-class)	Achieved 98.84% for f-measure.	F-Measure: DoS (99.7%), U2R (91.0%), R2L (96.1%), PROBE (98.8%).
Conclusion	Comparable performance to the best results in previous work.	High accuracy and performance in identifying specific attack types.

STL, a deep learning method, excels in learning feature representations from large amounts of unlabeled data, making it well-suited for detecting unknown and new attacks. With a two-stage process involving Unsupervised Feature Learning (UFL) and Classification, STL, particularly using Sparse Autoencoder, achieves accuracy rates exceeding 98% for various classification scenarios. It demonstrates superior precision, recall, and f-measure values in 2-class classification, showcasing its effectiveness in anomaly detection. However, STL faces challenges in proper feature selection and relies on the availability of substantial labeled traffic datasets.

On the other hand, Ensemble Algorithms prove formidable in combining the strengths of multiple base models, enhancing overall performance. These algorithms, employing diverse base learners such as decision trees, SVMs, and neural networks, showcase notable accuracy, precision, recall, and f-measure metrics for specific attack types. The ensemble approach is particularly effective in identifying DoS, U2R, R2L, and PROBE attacks with high precision and recall rates. Despite potential challenges related to model selection and diversity, Ensemble Algorithms provide a robust solution for NIDS, leveraging the strengths of various learners to improve accuracy and performance, especially in recognizing specific attack patterns. Ultimately, the choice between STL and Ensemble Algorithms depends on the specific requirements of the NIDS and the nature of the threats it aims to detect

5.5 Discussions

In the development of our intrusion detection system (IDS), we employed a comprehensive set of methodologies to ensure the robustness and effectiveness of our machine learning model. The initial phase involved data preprocessing, where we loaded datasets using Pandas, ensuring the necessary format for training and testing. To handle categorical data, we utilized One Hot Encoding and Label Encoding, converting non-numeric features into suitable numerical formats.

Addressing the challenge of class imbalance, we categorized attack types into broader classes (DoS, Probe, R2L, U2R). Feature scaling was then applied using StandardScaler to standardize numerical features, preventing dominance by a single feature. In the feature engineering phase, Random Forest with Recursive Feature Elimination (RFE) was employed for feature selection, optimizing model efficiency.

For model training, we utilized Random Forest, K Nearest Neighbors (KNN), and Support Vector Machines (SVM). The ensemble approach, combining predictions from SVM, RF, and KNN, enhanced overall model performance and robustness.

To evaluate model performance, confusion matrices were generated, providing a detailed breakdown of classifier effectiveness for each attack category. Cross-validation matrices were employed to assess generalization to independent datasets, considering metrics such as accuracy, precision, recall, and F-measure.

Calibration of probabilities, a crucial aspect for different classifiers, was achieved using 'CalibratedClassifierCV' to ensure consistency and reliability in probability estimates. These methodologies collectively contribute to the development of a sophisticated IDS capable of effectively detecting and mitigating various forms of malicious activities within networks.


6. Conclusion and Future Developments

In conclusion, our Intrusion Detection System (IDS) built on ensemble algorithms represents a formidable stride in bolstering cybersecurity defenses. By amalgamating the strengths of diverse machine learning models such as Random Forests, KNN and SVM, our IDS exhibits a robust and versatile defense mechanism against an evolving spectrum of cyber threats. The ensemble approach harnesses the collective intelligence of these models, enhancing the overall accuracy and reliability of intrusion detection. Through meticulous feature engineering, extensive dataset analysis, and iterative model refinement, our IDS achieves a balance between precision and recall, crucial for minimizing false positives and negatives. The ensemble methodology not only fortifies the system against known attack vectors but also demonstrates a capacity to adapt to novel and sophisticated threats, showcasing its resilience in real-world scenarios.

In the ever-evolving landscape of cybersecurity, our IDS stands as a testament to innovation, collaboration, and a proactive approach, poised to make significant strides in advancing the collective security posture of individuals and organizations alike.

7. References

- Ravipati, R. D., & Abualkibash, M. (2019, June). Intrusion Detection System Classification Using Different Machine Learning Algorithms on KDD-99 and NSL-KDD Datasets - A Review Paper. *International Journal of Computer Science & Information Technology (IJCSIT)*, Vol 11, No 3, June 2019. Available at SSRN: <https://ssrn.com/abstract=3428211> or <http://dx.doi.org/10.2139/ssrn.3428211>
- Ravipati, R. D., & Abualkibash, M. (2019, June). Intrusion Detection System Classification Using Different Machine Learning Algorithms on KDD-99 and NSL-KDD Datasets - A Review Paper. *International Journal of Computer Science & Information Technology (IJCSIT)*, Vol 11, No 3, June 2019. Available at SSRN: <https://ssrn.com/abstract=3428211> or <http://dx.doi.org/10.2139/ssrn.3428211>
- M. Di Mauro, G. Galatro, G. Fortino, A. Liotta, "Supervised feature selection techniques in network intrusion detection: A critical review," *Engineering Applications of Artificial Intelligence*, Volume 101, 2021, 104216, ISSN 0952-1976, <https://doi.org/10.1016/j.engappai.2021.104216>.
- Y. H. Dakhil, M. E. Özbek, and B. R. Al-Kaseem, "Packet Header Classification for Network Intrusion Detection System Based on FPGA," 2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), Ankara, Turkey, 2022, pp. 1-6, doi: 10.1109/HORA55278.2022.9800025
- Torabi, Majid, Nur Izura Udzir, Mohd Taufik Abdullah, and Razali Yaakob. "A Review on Feature Selection and Ensemble Techniques for Intrusion Detection System." *International Journal of Advanced Computer Science and Applications*, 12 (2021): n. pag
- Mukkamala, Srinivas, Guadalupe Janoski, and Andrew H. Sung. "Intrusion Detection Using Neural Networks and Support Vector Machines." *Proceedings of the International Joint Conference on Neural Networks*, 2002, pp. 1702-1707, doi: 10.1109/IJCNN.2002.1007774.
- Sommer, Robin, and Vern Paxson. "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection." *Proceedings - IEEE Symposium on Security and Privacy*, 2010, pp. 305-316, doi: 10.1109/SP.2010.25
- Tavallaee, Mahbod, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. "A detailed analysis of the KDD CUP 99 data set." 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, 2009, pp. 1-6, doi: 10.1109/CISDA.2009.5356528.
- Hamid, Yasir, M. Sugumaran, and Ludovic Journaux. "Machine Learning Techniques for Intrusion Detection: A Comparative Analysis." *Proceedings of the International Conference on Informatics and Analytics (ICIA-16)*, 2016, Article 53, 1-6, doi: 10.1145/2980258.2980378
- Almseidin, M., Alzubi, M., Kovacs, S., & Alkasassbeh, M. "Evaluation of machine learning algorithms for intrusion detection system." 2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY), Subotica, Serbia, 2017, pp. 277-282, doi: 10.1109/SISY.2017.8080566.

 <div style="display: inline-block; vertical-align: middle;"> <h1 style="margin: 0;">VIT[®]</h1> <p style="margin: 0;">Vellore Institute of Technology</p> <p style="font-size: small; margin: 0;">(Deemed to be University under section 3 of UGC Act, 1956)</p> </div>	<p>B.Tech (Information Technology) FALL 2023 – 2024 ITE 3007 :Cloud Computing and Virtualization 20.11.2023 – 24.11.2023</p>			
Title: _____				
Team Name _____				
Project Team				
S.No	Register Number	Student Name	Signature	Guided By
				Dr. M.Lawanyashri
Team Member(s) Contribution and Performance Assessment				
Components		Student 1	Student 2	Student 3
Implementation & Results				
-(20)				
Contributed fair share to the team project				
-(05)				
Documentation without Plagiarism-				
(20)				
Q & A				
- (05)				
Student Feedback <small>(Student Experience in this Course Project)</small>			Evaluator Comments	
Name & Signature of the Evaluator (Dr. M. Lawanyashri)				