INTRUSION DETECTION SYSTEM USING MACHINE LEARNING

BY

Priyanshu Ranjan (Enrolment No. 19/11/EC/041)
Shivam(Enrolment No. 19/11/EC/057)
Rahul Jaiswal (Enrolment No. 19/11/EC/009)
Sagar Arya (Enrolment No. 19/11/EC/061)

under the guidance of

Prof. Dr. Gajendra Pratap Singh, School of Computational and Integrative Science, JNU, Delhi

in the partial fulfilment of the requirements
for the award of the degree of

Bachelor of Technology
(a part of Five-Year Dual Degree Course)

ज.ने.वि.
JNU

School of Engineering
Jawaharlal Nehru University, Delhi
Jan, 2022

JAWAHARLAL NEHRU UNIVERSITY
SCHOOL OF ENGINEERING

DECLARATION

We declare that the project work entitled **“INTRUSION DETECTION SYSTEM USING MACHINE LEARNING”** which is submitted by us in partial fulfillment of the requirement for the award of degree B.Tech. (a part of Dual-Degree Programme) to School of Engineering, Jawaharlal Nehru University, Delhi comprises only our original work and due acknowledgement has been made in the text to all other material used.

PRIYANSHU RANJAN

[illegible]

RAHUL JAISWAL

Rahul Jaiswal

SHIVAM

Shivam

SAGAR ARYA

Sagar

JAWAHARLAL NEHRU UNIVERSITY
SCHOOL OF ENGINEERING

CERTIFICATE

This is to certify that the project work entitled **"INTRUSION DETECTION SYSTEM USING MACHINE LEARNING"** being submitted by **Priyanshu Ranjan**(Enrolment No.- 19/11/EC/041)**, Rahul Jaiswal**(Enrolment No.- 19/11/EC/009)**, Shivam**(Enrolment No.- 19/11/EC/057)**, Sagar Arya** (Enrolment No.- 19/11/EC/061) in fulfilment of the requirements for the award of the **Bachelor of Technology** (part of Five-Year Dual Degree Course) in **Computer Science & Engineering**, will be carried out by him under my supervision.

In my opinion, this work fulfils all the requirements of an Engineering Degree in respective stream as per the regulations of the School of Engineering, Jawaharlal Nehru University, Delhi. This thesis does not contain any work, which has been previously submitted for the award of any other degree.

Dr. Gajendra Pratap Singh
(Supervisor)
Professor
School of Computational and Integrative Science
Jawaharlal Nehru University, Delhi

JAWAHARLAL NEHRU UNIVERSITY
SCHOOL OF ENGINEERING

ACKNOWLEDGMENT

First, we would like to extend our heartfelt appreciation to our academic supervisor Dr. Gajendra Pratap Singh, for his guidance, support, and invaluable feedback throughout this project. His unwavering commitment to the project has been a great source of inspiration and motivation.

We would also like to express our gratitude to the researchers in the field of machine learning and cyber security, who had contributed in these areas.

Finally, we would like to thank our families and friends for their unwavering support and encouragement, which has been crucial to our success in this project.

Thank you all for the contributions, support, and encouragement throughout this project.

(Full Name and Sign of all Group Members)

Priyanshu Ranjan
Rahul Jaiswal
Shivam
Sagar Arya

ABSTRACT

Intrusion detection systems (IDS) using machine learning (ML) have emerged as a promising approach to detecting and responding to security threats in computer networks and systems. ML-based IDSs can analyze large volumes of data and automatically identify patterns or features that are indicative of malicious activity, providing a more accurate and efficient means of threat detection compared to traditional rule-based or signature-based methods.

In ML-based IDSs we are using a variety of ML algorithms, such as decision trees, KNN, random forests, Xgboost, logistic regression, and naive bayes, to distinguish network traffic or system activity as normal or abnormal. These algorithms can learn from historical data and adapt to new threats, providing a more effective means of detecting unknown or previously unseen attacks.

LIST OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Introduction
1.2 Problem Statement
1.3 Aim and Objective
1.4 Significance of the study

CHAPTER 2: LITERATURE SURVEY

2.1 Introduction
2.2 Performance Evaluation
2.3 Limitation and Future Research

CHAPTER 3: PROPOSED WORK AND METHODOLOGY

3.1 Our Approach
3.2 Data Cleaning and Preprocessing
3.3 Model Selection and Training

CHAPTER 4: RESULT DISCUSSION

CHAPTER 5: CONCLUSION AND FUTURE SCOPE
REFERENCES

CHAPTER-1

INTRODUCTION

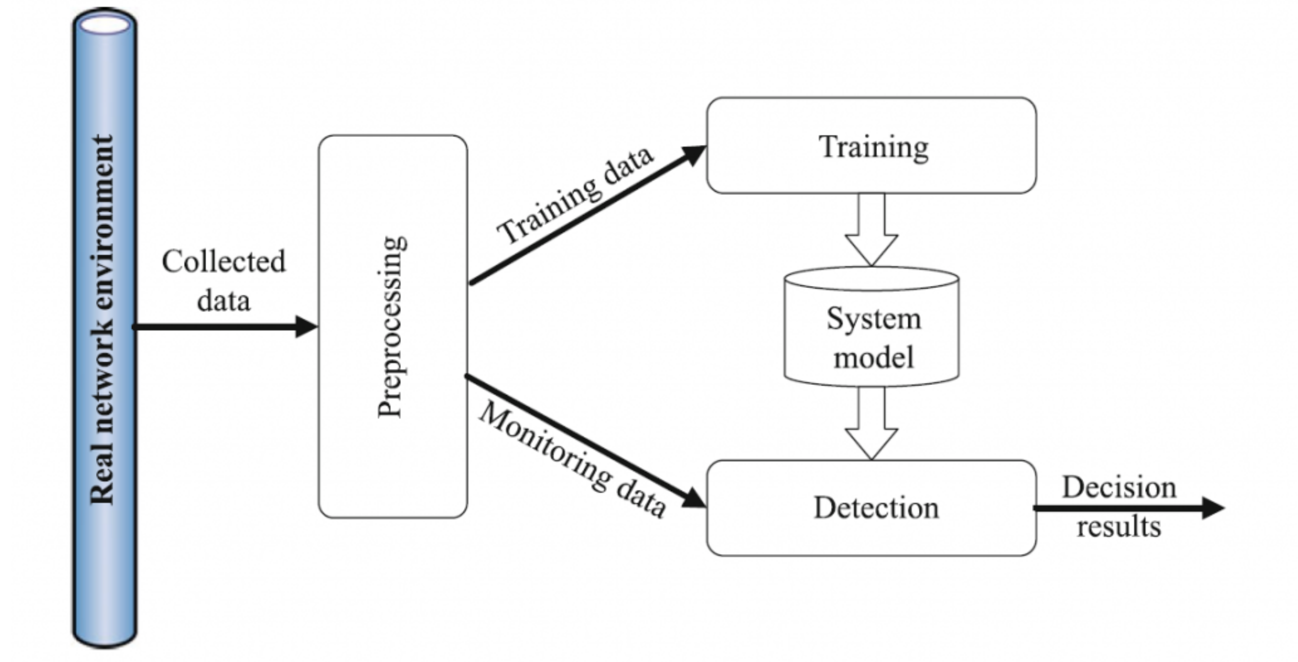
1.1. INTRODUCTION

A network is a collection of linked computers or other devices that may interact and share resources with each other. This can be a smaller network that just exists within a single building or workplace, or a bigger network that connects several sites.

In the context of computer networks, a normal connection refers to a legitimate communication between two devices or systems, where both parties have established a valid connection and are exchanging data or information according to an expected protocol. For example, a typical connection is made between the user's computer and the web server when they view a website. The user's browser then makes HTTP calls to the server, and the server replies with HTML information.

On the other hand, an intrusion refers to an unauthorized attempt to gain access to a computer system or network, where the attacker is attempting to exploit vulnerabilities or weaknesses in the system's defenses. Intrusions can take many forms, such as denial-of-service attacks, port scans, brute-force attacks, and malware infections.

Intrusion detection systems (IDSs) are designed to differentiate between normal connections and intrusions by analyzing network traffic and system logs for patterns or features that are indicative of malicious activity. IDSs can use machine learning algorithms to learn from historical data and automatically identify new patterns of malicious behavior. They can also use predefined rules or signatures to identify known attack patterns or anomalous behavior.



1.2. PROBLEM STATEMENT

In this project, we will create a network intrusion detector, a prediction model that can distinguish between "good" or normal connections and "bad" connections, sometimes known as intrusions or attacks. A wide range of intrusions that were modeled in a network environment of the military are included in this dataset. MIT Lincoln Labs developed and supervised the management of the data required to construct the Intrusion detector.

1.3. AIM AND OBJECTIVE

The aim of this project is to develop an effective intrusion detection system using machine learning techniques that can accurately and efficiently detect network and system intrusions in real-time.

The objective of this project are as follow-

- To conduct a comprehensive review of the state-of-the-art machine learning algorithms used in intrusion detection systems, including decision trees, random forests, KNN, etc.

- To analyze & preprocess NSL-KDD dataset, a widely used benchmark dataset for intrusion detection systems, to obtain a representative sample of network traffic data that includes both normal and malicious behavior.

- To implement and evaluate different machine learning algorithms on the preprocessed NSL-KDD dataset, and compare their performance in terms of accuracy, false positive rates, and computational efficiency.

- To investigate the impact of feature selection techniques on the performance of the intrusion detection system and identify the most relevant features for detecting intrusions.

- To design and implement a real-time intrusion detection system using the best performing machine learning algorithm, and deploy it on a network environment to evaluate its effectiveness in detecting network and system intrusions in real-time.

- To evaluate the practical feasibility of the intrusion detection system in terms of its performance, scalability, and reliability, and provide recommendations for further improvements or enhancements.

1.4 SIGNIFICANCE OF THE STUDY

The significance of an intrusion detection system using machine learning project lies in its potential to improve the security of computer networks and systems by detecting and preventing unauthorized access and malicious activities in real-time.

Improving network security: The project can contribute to improving network security by detecting and alerting system administrators to potential intrusions before they can cause significant harm. By using machine learning algorithms, the intrusion detection system can identify new patterns of malicious behavior and adapt to changing threat landscapes.

Reducing false positives: By using machine learning algorithms, the intrusion detection system can learn from historical data and identify complex patterns of normal behavior that may be indicative of an attack. This can reduce false positive rates, which are a common problem in traditional rule-based intrusion detection systems.

Enhancing system performance: By using machine learning algorithms, we can improve the efficiency & scalability of intrusion detection systems. Real-time threat detection and response are made possible by machine learning algorithms' ability to swiftly and reliably recognize patterns in vast amounts of data.
Enabling proactive security measures: By detecting potential threats in real-time, the intrusion detection system can enable proactive security measures such as blocking or quarantining suspicious network traffic, reducing the risk of a successful attack.

Contributing to research: The project can contribute to advancing research in the field of intrusion detection systems using machine learning techniques. By evaluating the effectiveness of different machine learning algorithms and feature selection techniques, the project can provide insights into the best practices for developing effective intrusion detection systems.

CHAPTER-2

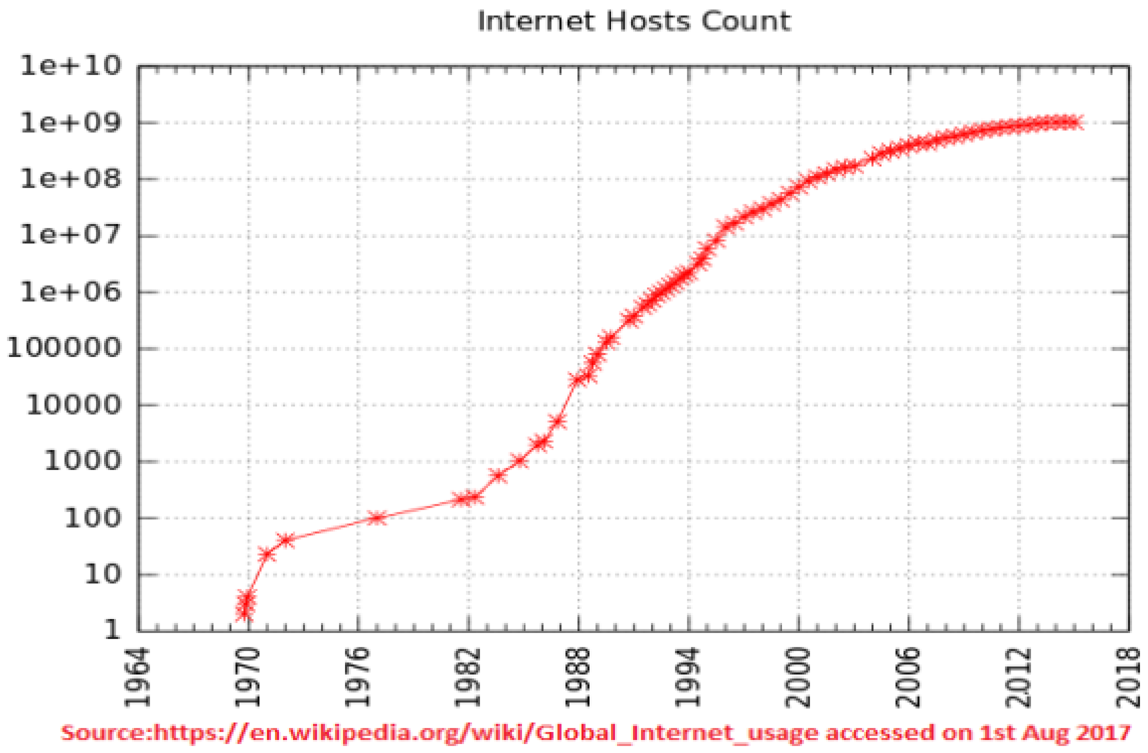
LITERATURE SURVEY

2.1. INTRODUCTION

Internet threats are increasing the risk to information security. The practice of monitoring and analyzing system events and network traffic in order to find any vulnerabilities and intrusions is known as intrusion detection. Currently, one of the most crucial tasks for information security administrators is intrusion detection. A system installed in a network must be protected from attacks because it is vulnerable to various types of attacks.

Although society has benefited greatly from the expansion of the Internet, the growing number of attacks on the IT infrastructure is becoming a critical problem that requires attention. Internet attacks are increasing concurrently with their expansion, as well.

The growth of Internet can be well-understood by the following Fig



2.2 PERFORMANCE EVALUATION

In machine learning, there are several performance metrics used to evaluate the performance of classification models, including AUC, CV score, confusion matrix, precision matrix, and recall matrix.

AUC: AUC (Area Under the Curve) is a metric used to evaluate the performance of binary classification models. It measures the ability of a model to distinguish between positive and negative classes. AUC ranges from 0 to 1, where an AUC of 0.5 indicates a random classifier and an AUC of 1 indicates a perfect classifier.

Confusion matrix: A confusion matrix is a table that shows the true positive, true negative, false positive, and false negative predictions of a model. It is often used to evaluate the performance of a classification model. The rows of the matrix represent the actual class labels, while the columns represent the predicted class labels.

Precision matrix: Precision is a metric that measures the fraction of true positive predictions out of all positive predictions made by the model. It is often used to evaluate the performance of a model on imbalanced datasets, where the number of instances in one class is much larger than the other.

Recall matrix: Recall is a metric that measures the fraction of true positive predictions out of all actual positive instances in the dataset. It is often used to evaluate the performance of a model on datasets where identifying all positive instances is important, such as in medical diagnosis or fraud detection.

F1-score: F1-score is a metric that combines both precision and recall to provide a single score that represents the overall performance of a model. It is the harmonic mean of precision and recall and ranges from 0 to 1, with 1 being the best possible score.

Accuracy: Accuracy is a metric that measures the proportion of correctly classified instances out of all instances in the dataset. While accuracy is a commonly used metric, it may not be suitable for imbalanced datasets, where the number of instances in one class is much larger than the other.

ROC curve: ROC (Receiver Operating Characteristic) curve is a graphical representation of the trade-off between the true positive rate and false positive rate of a classification model. It is often used to evaluate the performance of a binary classification model and to compare the performance of different models.

These performance metrics provide a variety of ways to evaluate the performance of classification models, and the choice of which metrics to use depends on the specific problem and the goals of the model.

The description of these values is as follows:
True Positive (TP): the number of records correctly classified to the Normal class.
True Negative (TN): the number of records correctly classified to the Attack class.
False Positive (FP): the number of Normal records incorrectly classified to the Attack class.
False Negative (FN): the number of Attack records incorrectly classified to the Normal class.

Based on the above values, the most commonly used Evaluation metrics are given by the following formulas:

$$TP\ rate = \frac{TP}{TP+FN}$$

$$TN\ rate = \frac{TN}{TN+FP}$$

$$FP\ rate = \frac{FP}{FP+TN}$$

$$FN\ rate = \frac{FN}{FN+TP}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

$$F1\ Score = 2 . \frac{Precision\ .Recall}{Precision+Recall}$$

2.3 LIMITATIONS AND FUTURE RESEARCH

Some limitations of intrusion detection system using machine learning include:

1. **Data Availability and Quality**: Machine learning algorithms require large amounts of high-quality data to train effectively. IDS may struggle

to collect enough labeled data for specific types of attacks, making it challenging to train the machine learning models

2. **Limited Scope**: IDS using machine learning techniques may only detect known attack patterns that have been included in the training data. New types of attacks or variants of known attacks may not be detected by the IDS.
3. **False Positives**: Machine learning-based IDS may generate a high number of false positives, which can be time-consuming for security teams to investigate and potentially result in alert fatigue.
4. **Adversarial Attacks**: Attackers can manipulate the input data to fool the machine learning model, which can result in false negatives or false positives.

One of the challenges of ML-based IDSs is the need for high-quality training data. IDSs must be trained on data that is representative of the system being protected and contains examples of both normal and malicious activity. Additionally, IDSs must be designed to handle the dynamic nature of network traffic and system activity, where normal behavior may vary over time.

Despite these challenges, The effectiveness and efficiency of attack detection in computer systems and networks have the potential to be improved by IDSs utilizing ML. As ML algorithms continue to improve and more data becomes available for training, ML-based IDSs are likely to become an increasingly important component of modern security strategies.

Some areas for future research and development include:

1. **Improving Data Quality and Availability**: Developing techniques to generate high-quality, labeled data, and developing methods to process unstructured data.
2. **Hybrid Approaches**: Combining machine learning techniques with traditional rule-based methods to improve accuracy and reduce false positives.
3. **Adversarial Machine Learning**: Developing techniques to detect adversarial attacks and improve the robustness of machine learning models.

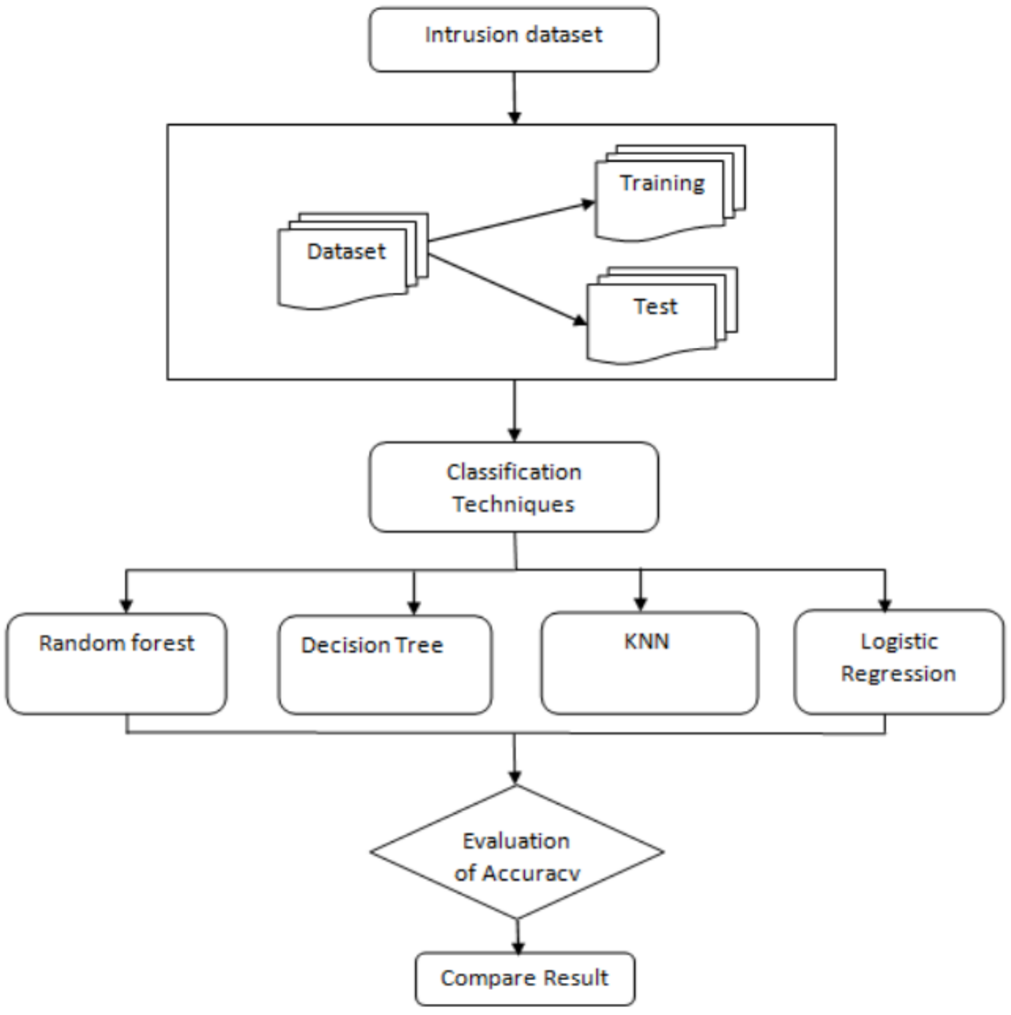
4. **Explaining the Decisions**: Developing techniques to explain the reasoning behind machine learning-based IDS decisions, which can improve trust and facilitate human decision-making.
5. **Real-time Processing**: Developing methods for real-time processing to enable IDS to detect and respond to attacks quickly.

CHAPTER-3

PROPOSED WORK AND METHODOLOGY

OUR APPROACH : We performed cleaning, pre-processing, and EDA, & for feature elimination we used a technique known as recursive feature elimination. We also tested many basic ML models along with a few customised models. We also used AUC as a metric because accuracy is not appropriate when dealing with imbalanced data.

The flowchart of our work is as given below-



We have done our project in a series of the below given work.

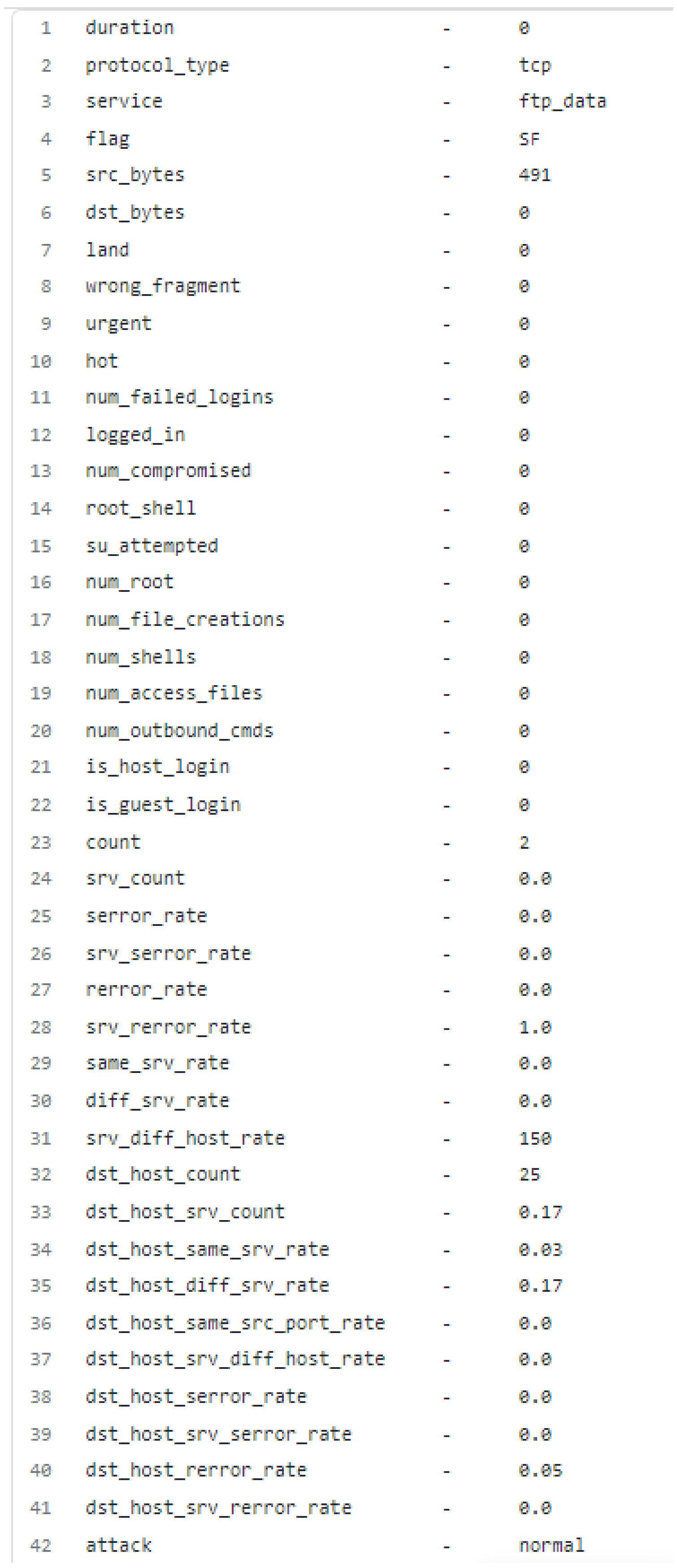
DATA COLLECTION AND PREPROCESSING : The first step is to collect the network traffic data, either from a simulated environment or from a real-world network.

- The data set used in this project is taken from NSL KDD (a new version of kdd-cup99).
- We may easily obtain the data set by clicking on this link. : https://www.unb.ca/cic/datasets/nsl.html

We have 2 data set in this project and they are as follow-

- Train data has 125973 data points and 42 features
- Test data has 22544 data points and 42 features

Example data points-



1	duration	-	0
2	protocol_type	-	tcp
3	service	-	ftp_data
4	flag	-	SF
5	src_bytes	-	491
6	dst_bytes	-	0
7	land	-	0
8	wrong_fragment	-	0
9	urgent	-	0
10	hot	-	0
11	num_failed_logins	-	0
12	logged_in	-	0
13	num_compromised	-	0
14	root_shell	-	0
15	su_attempted	-	0
16	num_root	-	0
17	num_file_creations	-	0
18	num_shells	-	0
19	num_access_files	-	0
20	num_outbound_cmds	-	0
21	is_host_login	-	0
22	is_guest_login	-	0
23	count	-	2
24	srv_count	-	0.0
25	serror_rate	-	0.0
26	srv_serror_rate	-	0.0
27	rerror_rate	-	0.0
28	srv_rerror_rate	-	1.0
29	same_srv_rate	-	0.0
30	diff_srv_rate	-	0.0
31	srv_diff_host_rate	-	150
32	dst_host_count	-	25
33	dst_host_srv_count	-	0.17
34	dst_host_same_srv_rate	-	0.03
35	dst_host_diff_srv_rate	-	0.17
36	dst_host_same_src_port_rate	-	0.0
37	dst_host_srv_diff_host_rate	-	0.0
38	dst_host_serror_rate	-	0.0
39	dst_host_srv_serror_rate	-	0.0
40	dst_host_rerror_rate	-	0.05
41	dst_host_srv_rerror_rate	-	0.0
42	attack	-	normal

- After reading the dataset got to know that these dataset have not the feature name in their appropriate column so columns name were given
    - The shape of training data (datapoints : 125973, features : 42)

    - The shape of test data (datapoints : 22544, features : 42)
- The task is to identify whether a given connection is normal or attack , for that we created a column "label" and gave all the attacks which are named normal as class 0 and all other attacks as class 1.
- By checking the distribution of the dataset with respect to the class label ,We found that the dataset is a little bit imbalanced(53.5% normal and 46.5% attack).
- Checked for duplicate and null value : there were not null and duplicate values.
- Checked for distribution with respect to different attacks in train and test dataset :
  Train data set is not uniformly distributed (by looking at different attacks). There are many attacks with few data points, while some of them, such as normal and neptune, having 85% data points out of 100% data points. There are 16 attacks out of 23 where the data points represent less than 1% of the test data. Look at the test dataset, we got a bunch of new assaults that weren't in the test data * regular and neptune attacks have more data points than others
- To analyze the feature we thought to analyse the categorical feature and numerical feature separately.
- Univariate analysis on categorical feature
    - we have 3 categorical feature : protocol_type , service and flag
    - to analysis these categorical feature we have gone through 4 things
- Number of category present in the dataset :
  - In protocol_feature : 3 categories present tcp,udp and icmp where majority of the points are from tcp and udp .

  - In service feature : 70 unique category present
  - In flag feature : 11 unique category present

- Distribution of the categorical feature :

   - In protocol_feature : tcp has both normal and attack class data points reasonable, where udp has more class0 (normal) points than class1(attack) and icmp has more class 1 - In service feature : The distribution is skewed, with few services occurring more frequently and the majority of services occurring less frequently. - In flag feature : It is also a skewed distribution

- featurization of the categorical feature :

  - All of them have been featurize using one hot encoding

- How good is this protocol_type feature in predicting y_i ?

  - building a simple model (Decision tree classifier) for each categorical feature and know the important feature. - In protocol_feature : we get some feature importance where udp is the most important feature in predicting yi where icmp is less

  - In service feature : model got test auc value of 87 by only using this feature this means this categorical feature will be useful when we build actual model.There are few feature which are important not all.

  - In flag feature : this feature is also useful as it has 78 test auc score, the model might overfit a little bit as train and test auc has a gap.

    - There are some categories where the model thinks that those features are not at all important , we can do some feature engineering by removing those unimportant features.
- Univariate analysis on continuous feature
    - Duration : majority of points have value of zero of this feature , from 98 percentile onward values are changing.
    - src_bytes : values of this feature are increasing in a slow rate upto 98 - 99.9 , but there is sudden change in the 100 percentile with a quite large value(1.25 gb) which mean 1.25 gb of data goes from source to destination, this might be an outlier.
    - dst_bytes : this is the same as the src_byte , in the 100 percentile there is quite a big number(1.21 gb of data from destination to source) this could be an outlier or maybe these values are a sign of attack.
    - wrong_fragment : in the the violin plot class 1 has more variance than class 1 while class 0 has value around 0
- Bivariate Analysis using pair plot

    - By looking at the pair plot we can say that there are some overlap between class1 and class0, but not fully.
    - The PDF's of each feature has some information like one pdf(class 0) has more value than another(class1) and vice versa. With this information i looked at the the violin plot and the pdf separately of some feature , where some of them are not fully overlap so we can say these feature may helpful in distinguish class 0 and class 1
- Multivariate analysis using tSNE
    - Here i have taken 32 feature and plotted using tsne with 7k data points , there result i get is quite brilliant
    - classes are separable
    - less overlapping points.

MODEL SELECTION AND TRAINING: The next step is to select the appropriate machine learning algorithm for intrusion detection. Machine learning models are widely used in IDS to detect such anomalies by analyzing the patterns in the data.This can be done by comparing the performance of different algorithms.

Here are some of the commonly used machine learning models for intrusion detection:

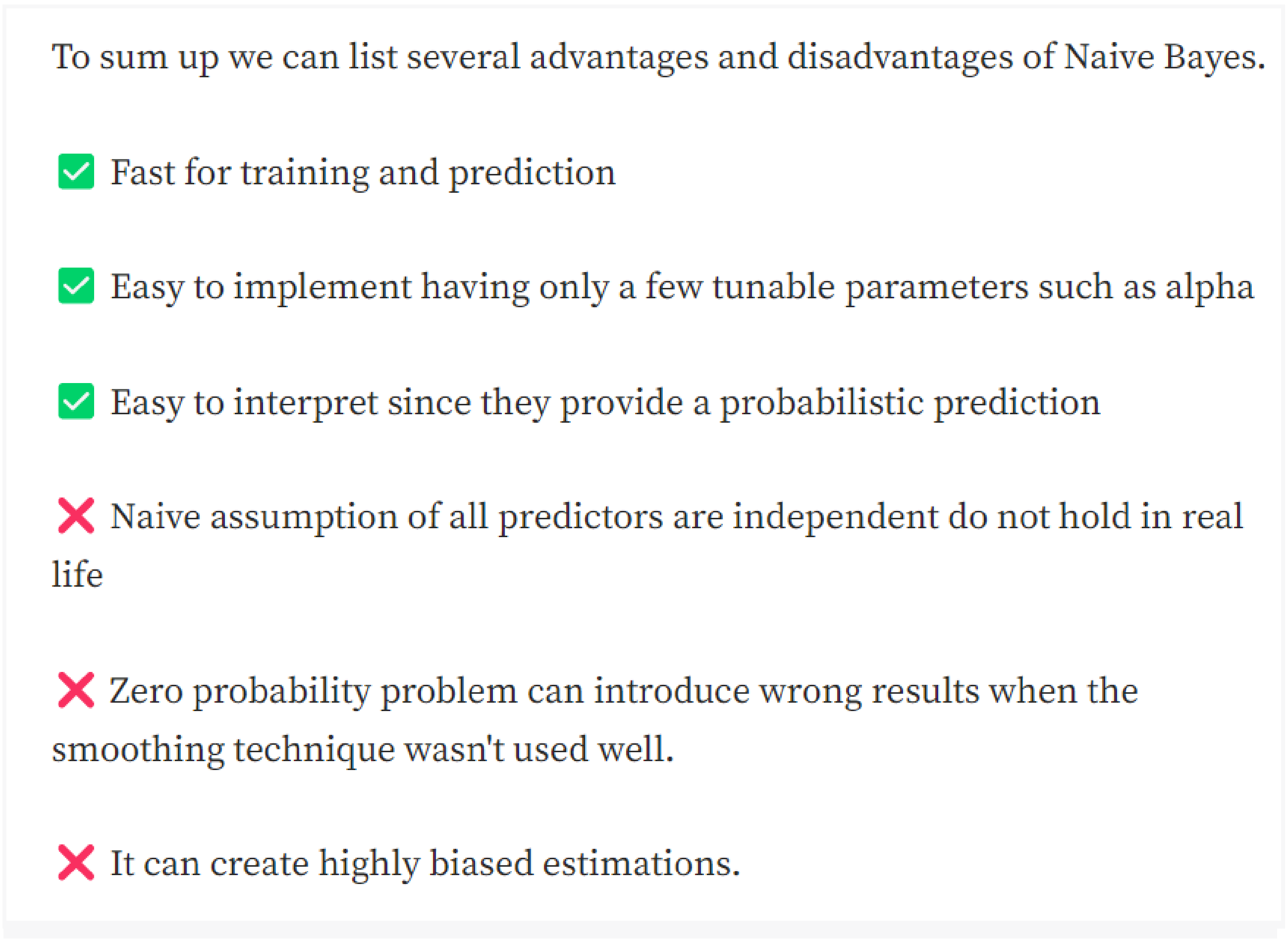
NAIVE BAYES CLASSIFICATION

Naive Bayes is a popular supervised algorithm used as a baseline model in intrusion detection systems. Naive Bayes is a probabilistic algorithm that is used to classify instances based on their features. In the context of intrusion detection, the features are typically network traffic attributes such as source IP address, destination IP address, port number, protocol, etc.

The Naive Bayes algorithm makes the assumption that, given the class variable, the features be conditionally independent. The computation of a posterior probability of class given the features is made simpler by this assumption. Assigning the instance to a class with a highest probability is what the algorithm does after calculating the probability of every class given the features.

In intrusion detection, Naive Bayes is often used to classify network traffic into two classes - normal traffic and malicious traffic. The algorithm is trained on a dataset of labeled instances, where each instance is labeled as normal or malicious. The algorithm estimates the probability distributions of the features for each class using the training data. During testing, the algorithm uses these probability distributions to calculate the posterior probability of each class given the features of the instance.

Naive Bayes is a fast and efficient algorithm that can handle high-dimensional data with many features. It is often used as a baseline model for intrusion detection because it is simple to implement and provides a good starting point for more advanced algorithms. However, it has some limitations, such as the assumption of conditional independence, which may not hold in some cases. Additionally, the algorithm may not perform well when there are rare events or when there is a class imbalance in the data.



To sum up we can list several advantages and disadvantages of Naive Bayes.

✅ Fast for training and prediction

✅ Easy to implement having only a few tunable parameters such as alpha

✅ Easy to interpret since they provide a probabilistic prediction

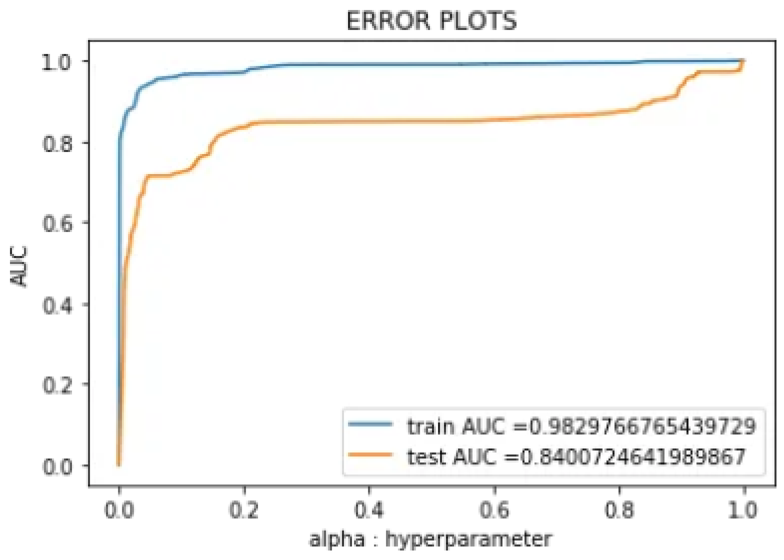
❌ Naive assumption of all predictors are independent do not hold in real life

❌ Zero probability problem can introduce wrong results when the smoothing technique wasn't used well.

❌ It can create highly biased estimations.

HYPER PARAMETER TUNING

The best hyper-parameter we found is **alpha=0.01**

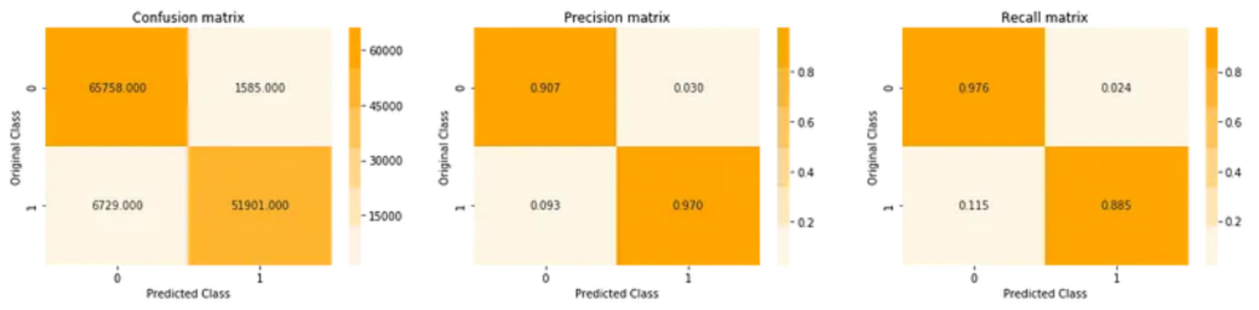


training data f1 score is 0.9258446608869385

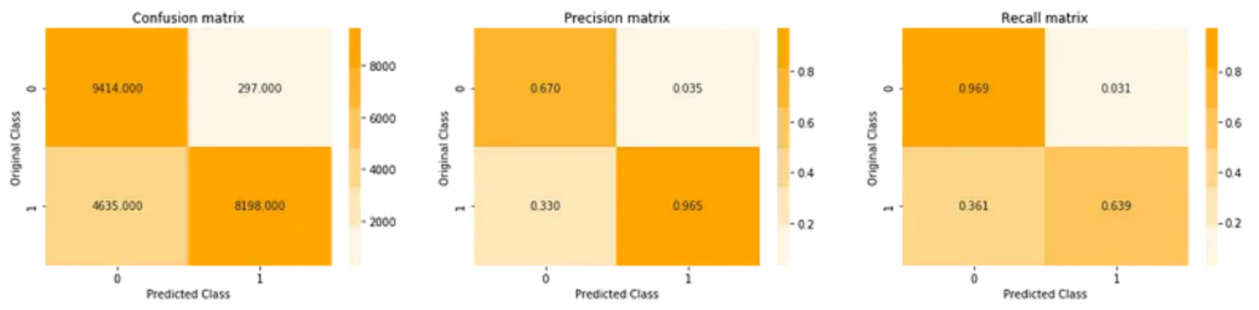
test data f1 score is 0.7687546886721681

training data recall score / detection rate is 0.8852294047415998

test recall score / detection rate is 0.6388217875788982



Train confusion matrix



Test confusion matrix

Observations:

- There is a difference between the train & test AUC values, indicating that the model is overfitting.
- Let us perform some feature selection to minimize overfitting.
- We will pick features using recursive feature elimination.

FEATURE SELECTION BY RECURSIVE FEATURE ELIMINATION

The purpose of recursive feature elimination (RFE) is just to pick features by iteratively considering smaller and smaller feature sets given an external estimator which assigns weight to features (for example, the coefficients of a linear model). The estimator is first trained on the original set of features, and also the importance of the each feature is determined using either a coef_ attribute or a feature importances_ attribute. The least significant features are then removed from the present list of features. This technique is continued recursively on the pruned set until the appropriate number of features to select is reached.

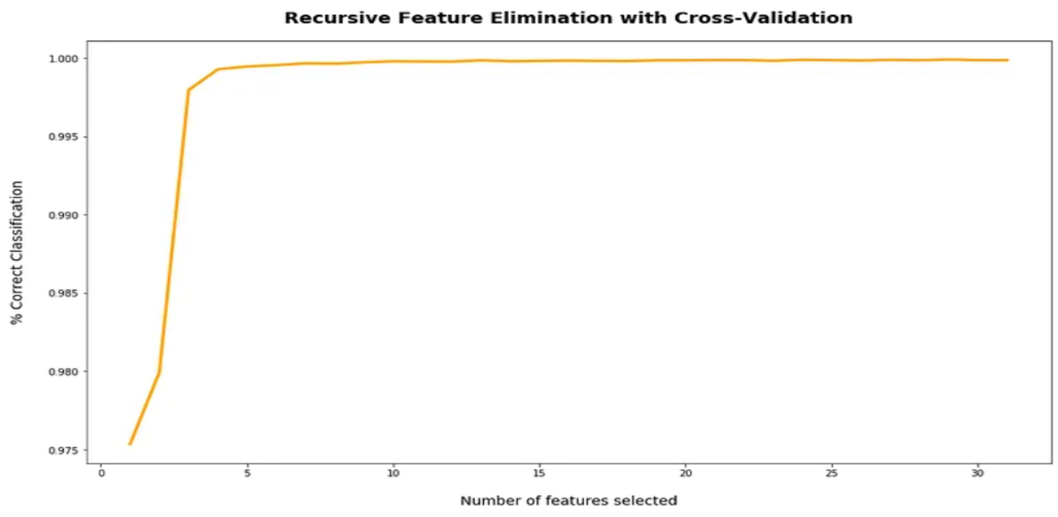
Refer :
https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html

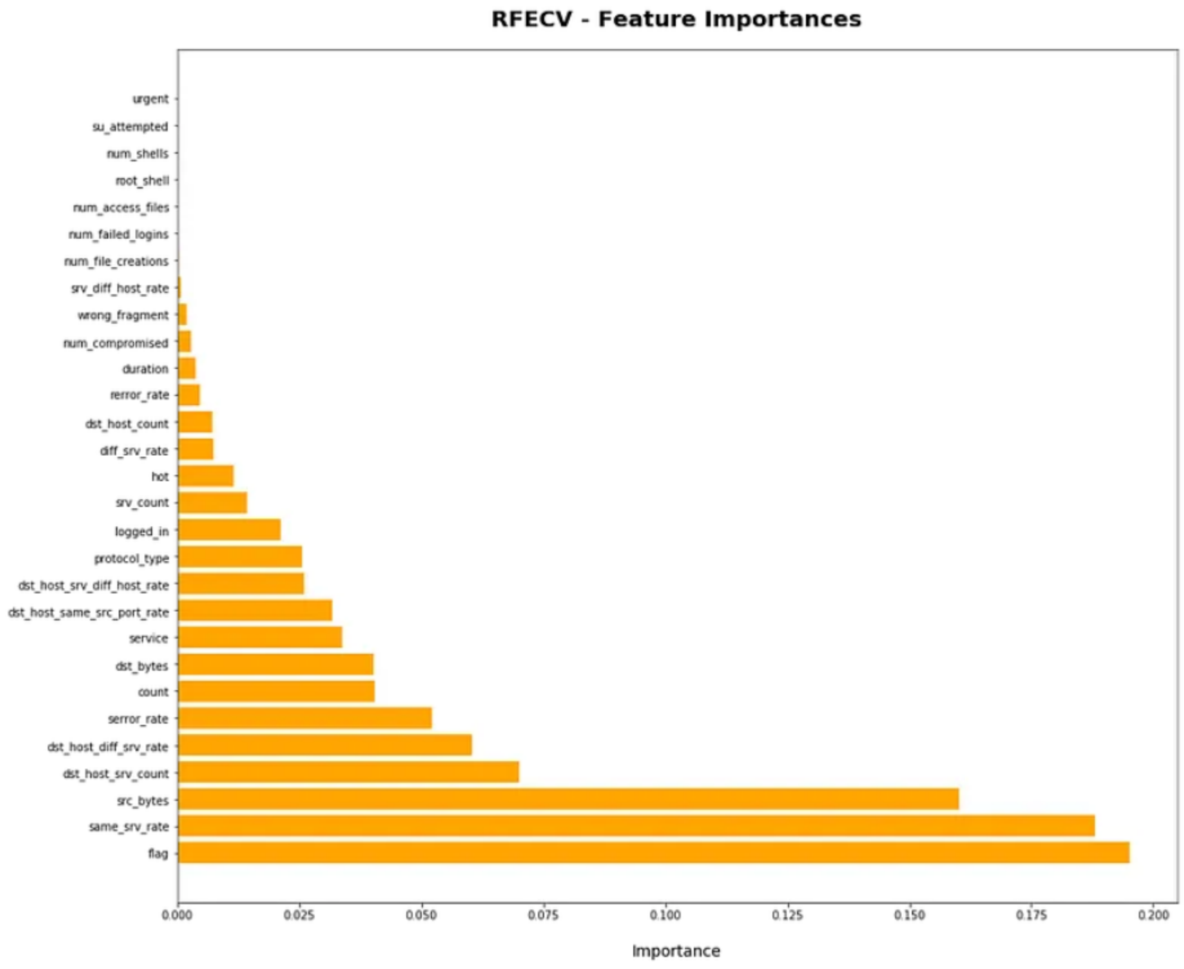
Procedure to get optimal or useful feature

- Take an object type feature (which signifies categorical) and encode it with a label encoder.
- combine encoder category and numerical features
- generate the correlation matrix with the data frame method corr(), which uses the Pearson correlation coefficient by default
- Iterate over two for loops, removing features with values greater than .8 in the correlation matrix
- Now, using recursive feature elimination (using the random forest model; we may use whatever model we want, but the RF gives strong feature importance), it offered 29 characteristics that are beneficial in forecasting the model (called optimal features)
- Then, by looking at the most significant features, we can remove ones that have a very low value.

```
print('Optimal number of feature: {}'.format(rfecv.n_features_))
```

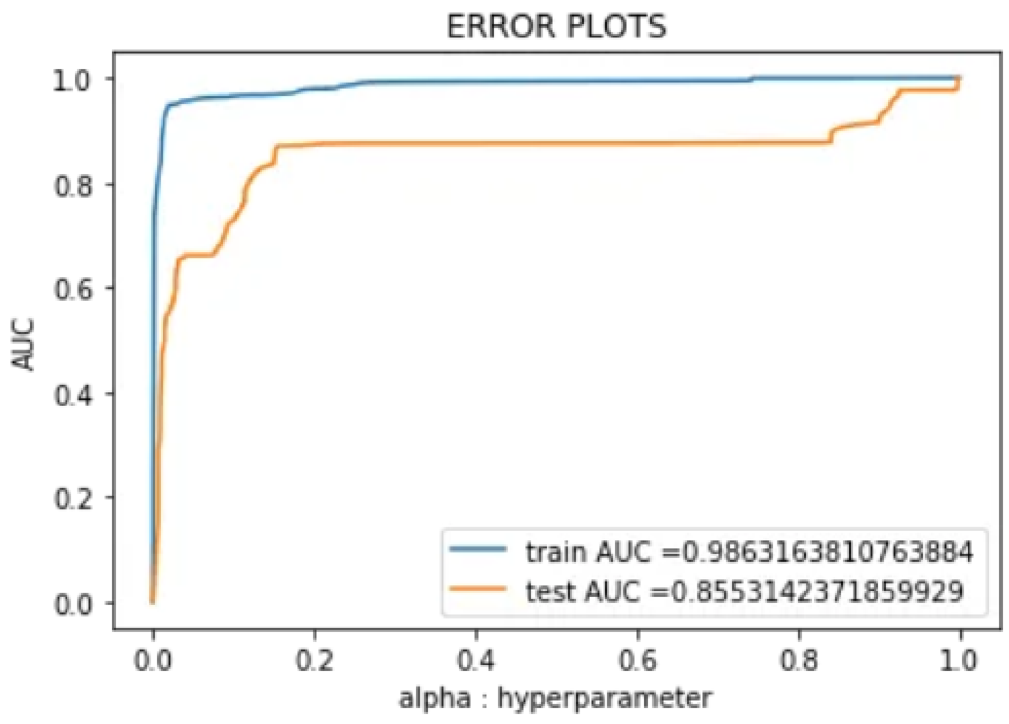
Output : Optimal number of features: 29





NAIVE BAYES WITH HYPER PARAMETER TUNING

- Here the best hyper parameter : alpha = 10

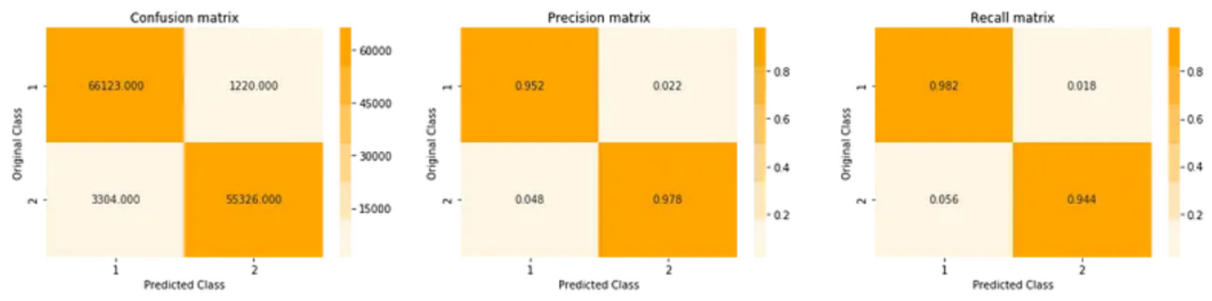


Training data f1 score is 0.9607209835382371

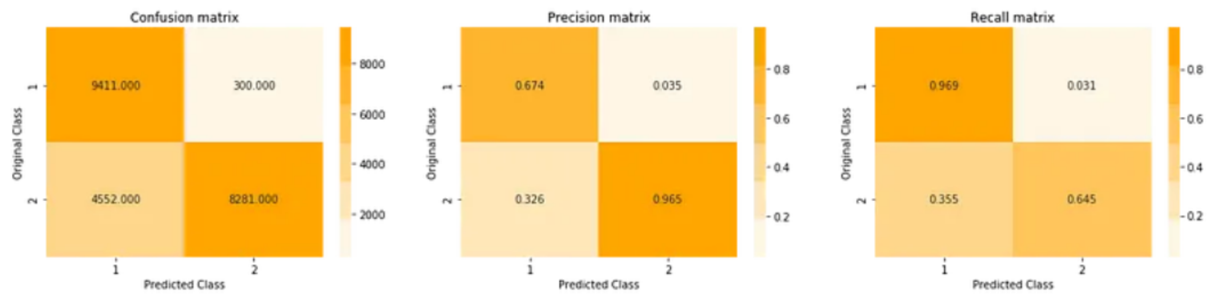
Test data f1 score is 0.773419258429065

training data recall score / detection rate is 0.9436465973051339

test data recall score / detection rate is 0.6452894880386504'



Train confusion matrix



Test confusion matrix

Observation :

1 -> class 0 & 2 -> class 1

- After deleting certain extraneous features, the f1 score & recall both improve slightly.
- There appears to be some uncertainty regarding recall: 64% of all actual points is predicted to be a class 2 and about 36% are expected to be class 0.

KNN HYPER PARAMETER TUNING

The non-parametric machine learning technique K-Nearest Neighbors (KNN) can be applied to classification and regression issues. In order to forecast the class (or value) of a query instance, KNN locates the k-nearest neighbors of the query instance in a training data and uses their class label (or values). A hyperparameter that may be adjusted to improve the algorithm's performance is the value of k.

Hyperparameter tuning involves selecting the optimal hyperparameters for a model to achieve the best performance on a validation dataset.The tuning of hyperparameters can be done using a variety of techniques, such as random search, grid search, and Bayesian optimization. Here, we will focus on grid search, which is a simple and commonly used method.

Grid search involves evaluating the performance of a model with different hyperparameter combinations by training a model on training dataset and evaluating it’s performance on the validation dataset. The hyperparameter combinations are selected from a predefined range of values for each hyperparameter.

To tune the k hyperparameter in KNN, we can use grid search to evaluate the performance of the model with different values of k. The steps involved in hyperparameter tuning for KNN are:

Split the dataset into training, validation, and test sets.

Define a range of values for the k hyperparameter.

Train a KNN model for each value of k on the corresponding training data, and then assess the model's performance on validation data using the performance metric like accuracy or F1-score.

Choose the k value that performs the best on validation data.

Train a KNN model using the chosen value of k on training and the validation data, and then assess its performance using the test data.

The math behind KNN hyperparameter tuning involves calculating the distance between the query instance and its k-nearest neighbors in the training data. The

Euclidean distance is the most commonly used distance metric in KNN, and it is determined as follows:

$$d(x, y) = \sqrt{(x1 - y1)^2 + (x2 - y2)^2 + ... + (xn - yn)^2}$$

where x and y are dataset instances, n is number of features, and xi & yi are the ith feature values in the instances x and y respectively.

Once the distances between the query instance and all instances in the training data have been calculated, the k instances with the shortest distances are selected as the k-nearest neighbors. The class (or value) of the query instance is then predicted as the majority class (or average value) among the k-nearest neighbors.

For values of **best alpha = 99** .The **train auc score** is: 0.9997159094312437

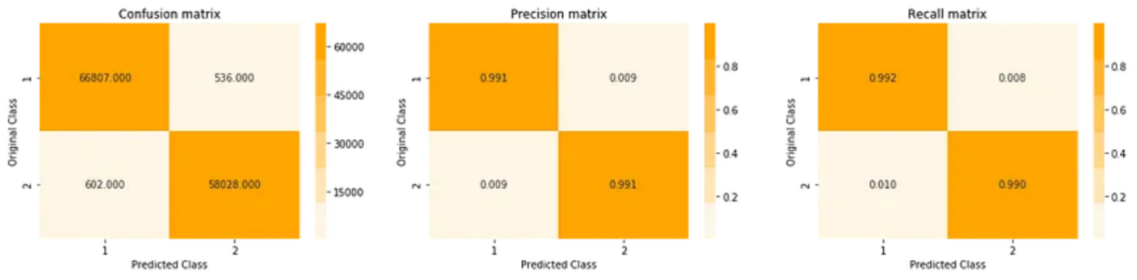
For values of **best alpha = 99 .**The test **auc score** is: 0.8918894763568558

Training data f1 score is 0.9902896052698944

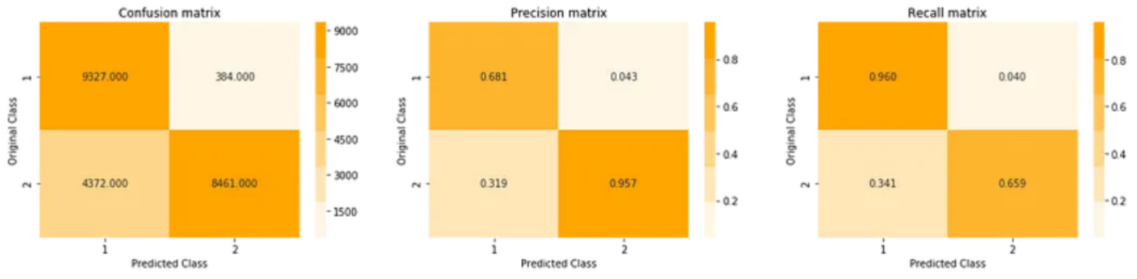
Test data f1 score is 0.7806070670726082

Training data recall score / detection rate is 0.9897322190005117

Test data recall score / detection rate is 0.6593158263851009



Train confusion matrix



Test confusion matrix

Observation :

1 -> class 0 & 2 -> class 1

- This simple knn model gives us greater AUC, f1, and recall value, which is a good indicator.
- Nonetheless, there is still considerable misunderstanding in the recall on test results between classes 2 and 1.
- This could be due to a disparity in class.

LOGISTIC REGRESSION HYPERPARAMETER TUNING

Logistic Regression is a popular machine learning algorithm used for binary classification problems. It models the relationship between a binary dependent variable and one or more independent variables using a logistic function. In logistic regression, hyperparameters are parameters that cannot be directly learned from the training data and must be set before the training process. Examples of hyperparameters include the learning rate, regularization parameter, and the type of penalty used in regularization.

Hyperparameter tuning in logistic regression involves selecting the optimal hyperparameters to improve the performance of the model on the validation dataset. The tuning of hyperparameters can be done using a variety of techniques, such as random search, grid search, and Bayesian optimization. Here, we will focus on grid search, which is a simple and commonly used method.

The following are the steps involved in hyperparameter tuning for logistic regression:

Split the dataset into training, validation, and test sets.
Define a range of values for the hyperparameters to be tuned, such as the learning rate, regularization parameter, and penalty type.
Train a logistic regression model on a training data for every combination of hyperparameters, and assess its performance on the validation data using the performance metric like F1-score, accuracy, or area under the ROC curve (AUC).
Select the combination of hyperparameters that gives the best performance on the validation data.
Train a logistic regression model on the combined training and validation data with the selected hyperparameters and evaluate its performance on the test data.
The math behind logistic regression Hyperparameter tuning involves the loss function used to train the model. The loss function represents the difference between predicted output of the model and the actual output of the data. The two most commonly used loss functions for logistic regression are the log loss function and the hinge loss function.

The log loss function is defined as:

$L(y, h(x)) = -[y*\log(h(x)) + (1-y)*\log(1-h(x))]$

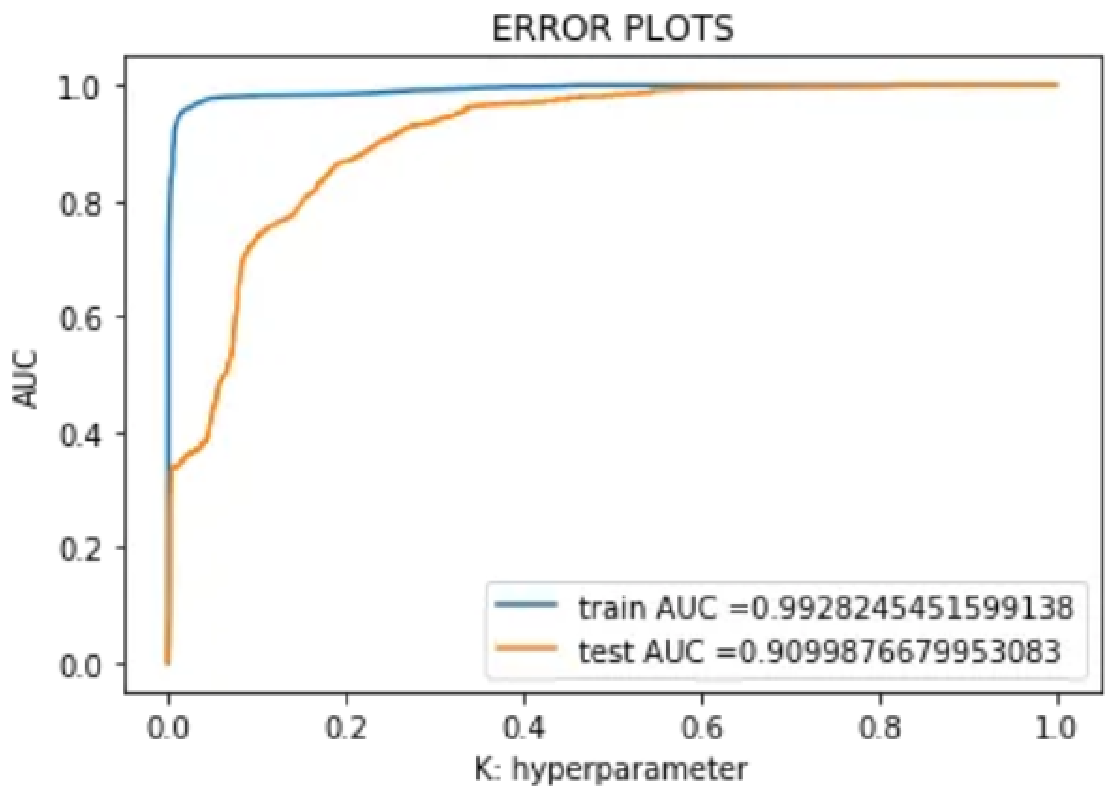
where y is the actual output (0 or 1), h(x) is the predicted output of the model for input x, and log is the natural logarithm.

The hinge loss function is defined as:

L(y, h(x)) = max(0, 1-y*h(x))

where y is the actual output (1 or -1), h(x) is the predicted output of the model for input x, and max is the maximum function.

By inserting a penalty term to the loss function, regularization is frequently employed in logistic regression to prevent overfitting. The two most common types of regularization are L1 regularization (Lasso) and L2 regularization (Ridge). The regularization parameter controls the strength of the penalty term.



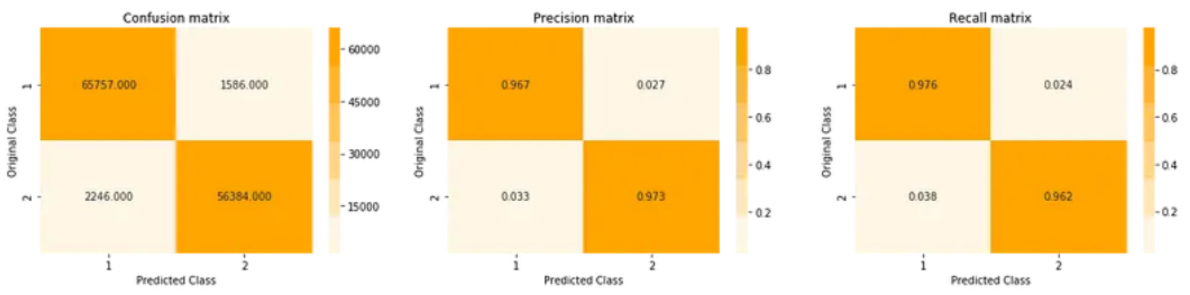
Training data f1 score is 0.9671355060034306

Test data f1 score is 0.7264708642207001

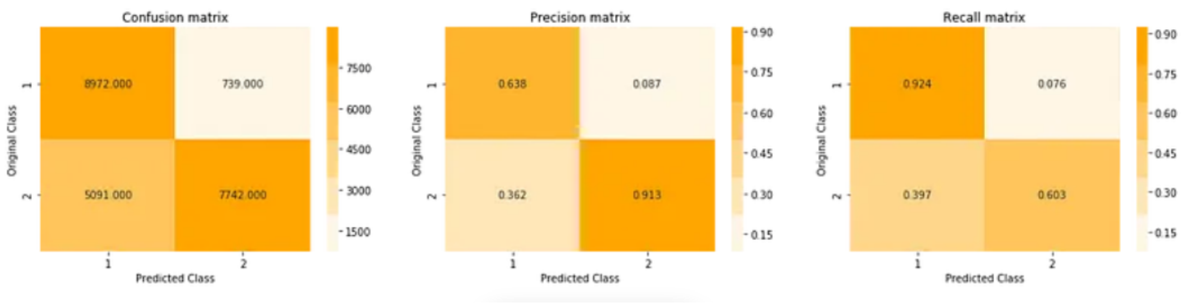
Training data recall score / detection rate is 0.9616919665700153

Test data recall score / detection rate is 0.6032883971012234

train confusion matrix



Test confusion matrix



Observation :

1 -> class 0 & 2 -> class 1

- Although the model's f1 and recall are lower than those of the baseline model, it has a high auc value.
- Also, recall model does have some ambiguity here.

DECISION TREE WITH HYPERPARAMETER TUNING

A decision tree is a well-known machine learning technique that is used for regression and classification applications. It operates by dividing the input space recursively into smaller and smaller sections dependent on the input's value characteristics. The partitions are chosen in such a way that the impurity of the resulting subsets is minimized.

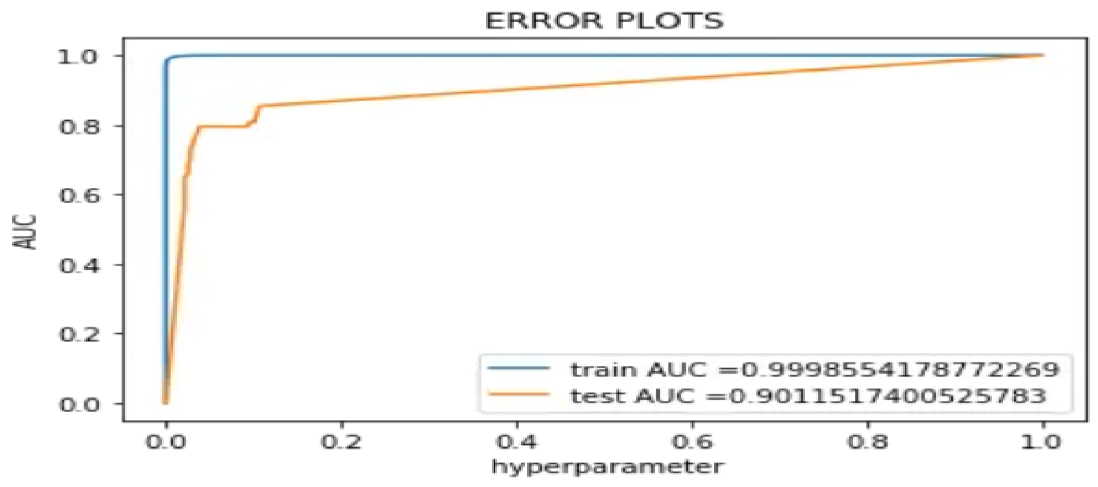
Hyperparameters are variables that are chosen in advance of the training process rather than ones that are learned from the data. The maximum depth of a tree, the minimum number of samples needed to split the internal node, the minimum number of samples needed to be at leaf node, and the criterion used to assess the quality of a split are a few examples of hyperparameters for decision trees.

Decision tree hyperparameter tuning entails picking the best hyperparameters to improve the model's performance on the validation dataset. There are various methods for adjusting hyperparameters, including random search, grid search, and Bayesian optimization. Here, we will focus on grid search, which is a simple and commonly used method.

The following are the steps involved in hyperparameter tuning for decision trees:

- Create validation, training, and test sets from the dataset.
- Set a range of values for hyperparameters that will be adjusted, such as the maximum depth of a tree, the minimum number of samples needed to split a internal node, a minimum number of samples needed to be at leaf node, and the criterion for judging the quality of a split.
- Using a performance metric like accuracy, F1-score, or area under the ROC curve, train a decision tree on the training data for each combination of hyperparameters, and then assess its performance on the validation data (AUC).
- Choose the set of hyperparameters that performs the best on the validation data.
- Train a decision tree using the selected hyperparameters on combined training and validation data, and then assess its performance using the test data.

- The math behind decision tree hyperparameter tuning involves the impurity measures used to split the tree. Entropy and Gini impurity are the two most frequently used impurity measurements. Entropy gauges the level of disorder in a collection of elements, while Gini impurity gauges the likelihood of misclassifying an randomly selected element from given class.

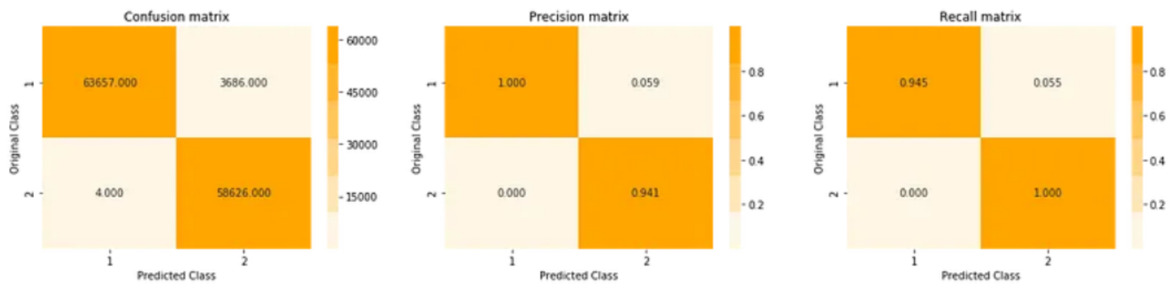


Training data f1 score is 0.9694895073671678

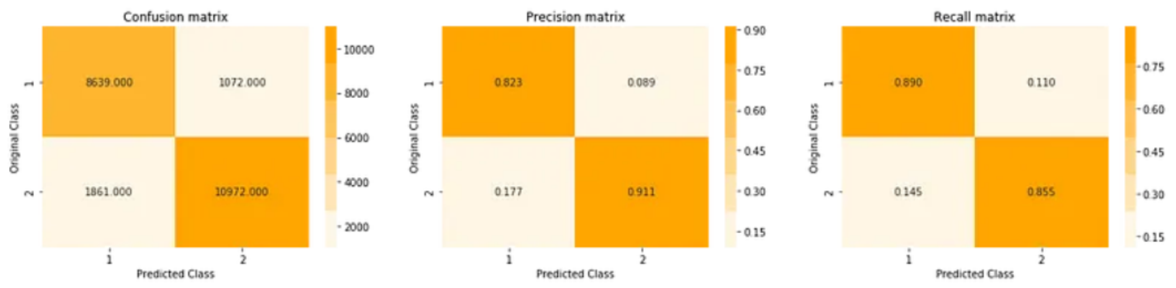
Test data f1 score is 0.8820999316637859

Training data recall score or detection rate is 0.9999317755415317

Test data recall score or detection rate is 0.8549832463180862



train confusion matrix



test confusion matrix

Observation :

1 -> class 0 & 2 -> class 1

- This is currently our best available model.
- After implementing some class balancing, we achieved rather high AUC, F1 scores, and recall.
- Model is no longer as confused as it was previously on classes 1 and 0.

RANDOM FOREST HYPERPARAMETER TUNING

An ensemble learning system called Random Forest mixes various decision trees to produce predictions. Each decision tree in a Random Forest is trained using a randomly chosen subset of the characteristics and data points. The computer then integrates all of the trees' forecasts to arrive at a final prediction. Hyperparameter -tuning is a crucial step to boost the performance of the well-liked and successful Random Forest technique for classification issues.

Here are the main hyperparameters for Random Forest:

n_estimators: The Random Forest's decision tree count is controlled by this parameter. The model's performance may be enhanced by adding more trees, but this would increase the computational expense.

max_depth: This setting regulates how deep each decision tree can go. The complexity of the model and the danger of overfitting can both increase when the depth is increased.

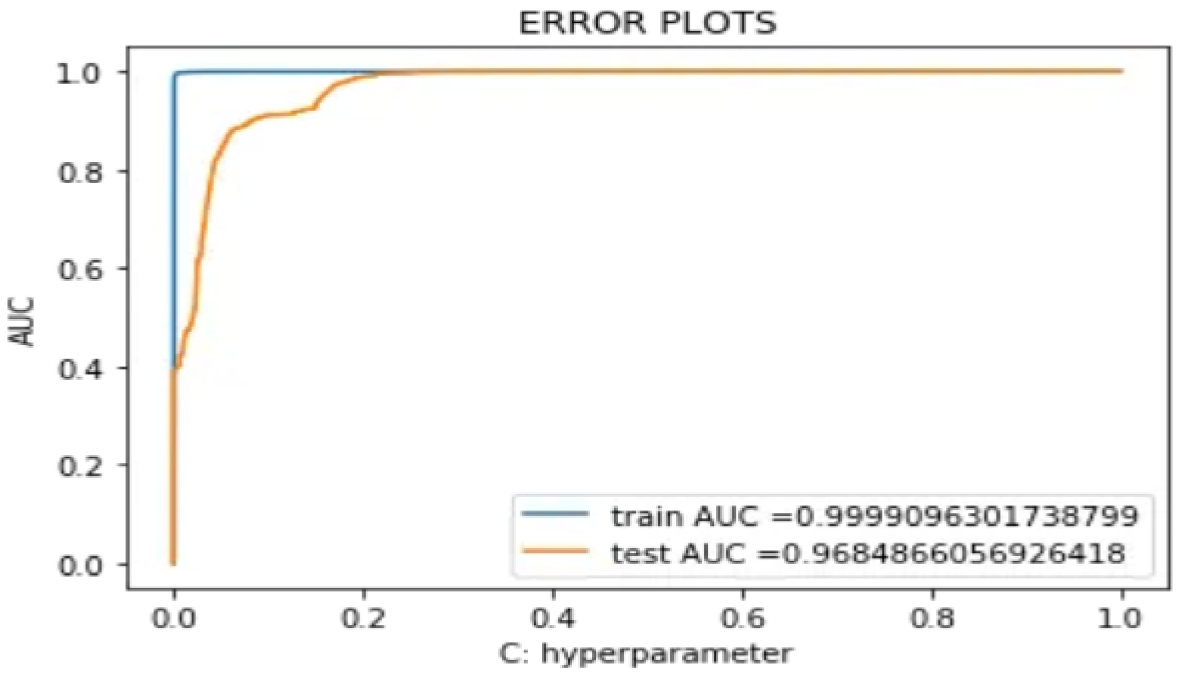
min_samples_split: The minimum number of samples necessary to split an internal node is specified by this parameter. Increasing this value can reduce the risk of overfitting by preventing the model from creating small leaves that only fit a few data points.

min_samples_leaf: This parameter specifies the minimum number of samples required to be at a leaf node. Increasing this value can also reduce the risk of overfitting by preventing the model from creating leaves that only fit a few data points.

max_features: This option indicates the most features that should be taken into account while determining the optimal split. Reducing the number of features can reduce the model complexity and the risk of overfitting.

Hyperparameter tuning for Random Forest can be done using methods such as grid search, random search, or Bayesian optimization. These methods work on the fundamental premise of experimenting with various combinations of hyper - parameters and assessing the model's performance against a validation set. The hyperparameters that give the best performance are then selected.

Mathematically, Random Forest works by combining the predictions of multiple decision trees. Each decision tree is trained using a randomly selected subset of the features and data points. The algorithm uses a voting scheme to make a final prediction. Each decision tree specifically makes a forecast for a each test data point, and final prediction is the result of the decision trees' majority vote.

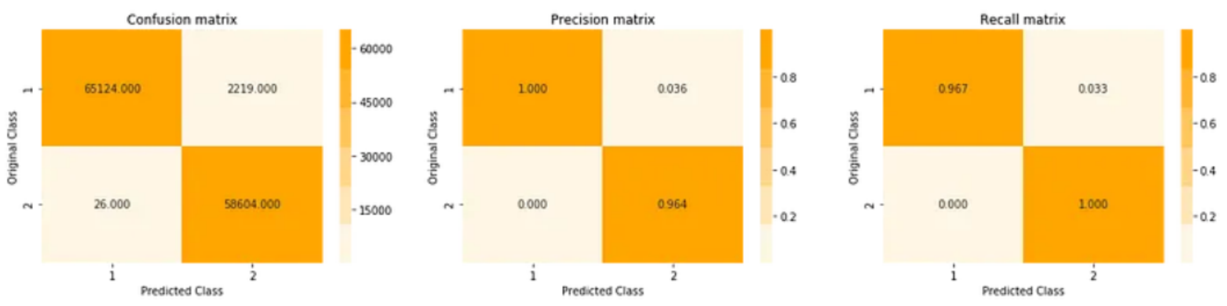


Training data f1 score is 0.9812059973378651

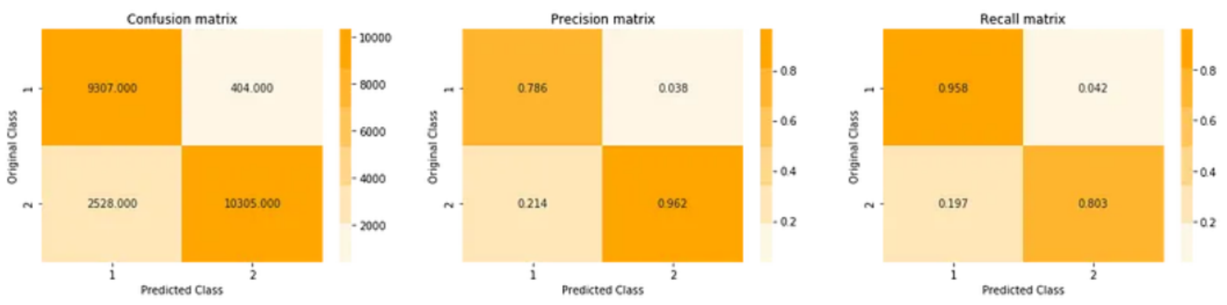
Test data f1 score is 0.8754566307025742

Training data recall score / detection rate is 0.9995565410199556

Test data recall score / detection rate is 0.8030078703342944



Train confusion matrix



Test confusion matrix

Observation :

1 -> class 0 & 2 -> class 1

- In terms of AUC, this model performs better than other models; it also has a strong f1 and recall value, although not as well as Decision tree.

XGBOOST HYPERPARAMETER TUNING

XGBoost (Extreme Gradient Boosting) is a popular and powerful algorithm for classification problems. It is a boosting algorithm that trains multiple decision trees in sequence, where each new tree is trained to correct the errors of the previous trees. Hyperparameter tuning is an important step in improving the performance of XGBoost.

Here are the main hyperparameters for XGBoost:

learning_rate: This parameter controls the step size that the algorithm takes in each iteration. A smaller learning rate will make the model more conservative, but also increase the number of iterations required to reach convergence.

n_estimators: This parameter controls the number of decision trees in the model. Increasing the number of trees can improve the model performance, but also increases the computational cost.

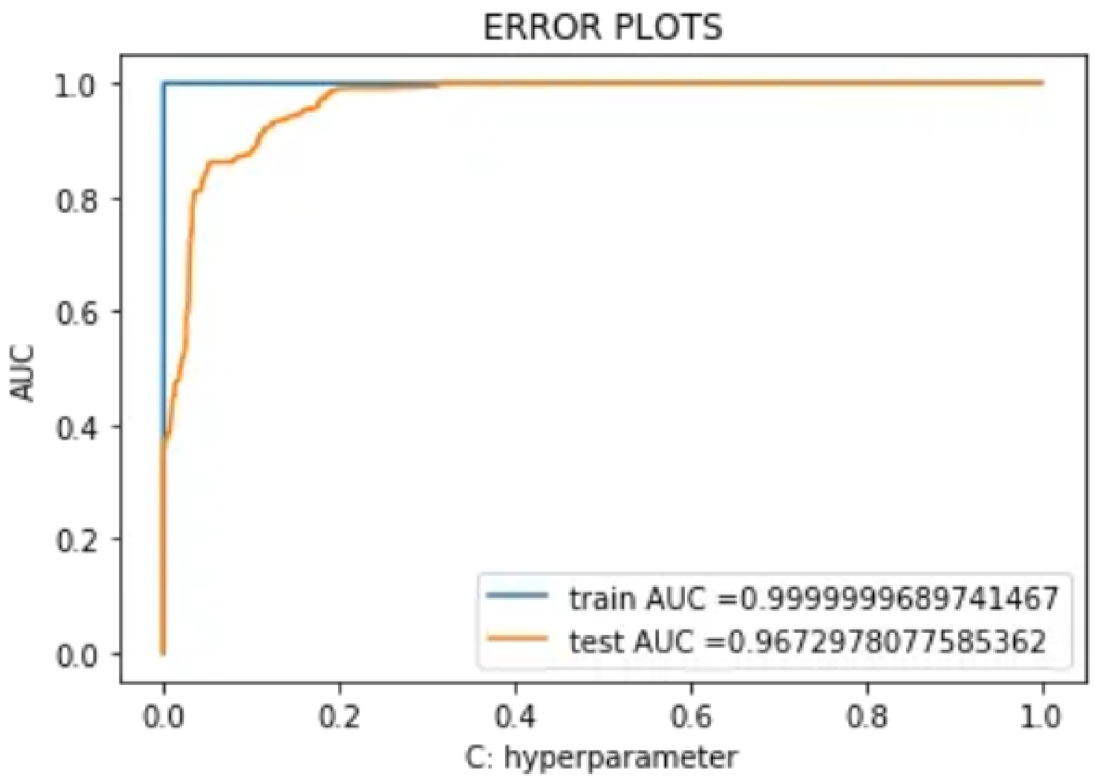
max_depth: This parameter controls the maximum depth of each decision tree. Increasing the depth can increase the model complexity and the risk of overfitting.

subsample: This parameter controls the fraction of data points to be used for training each tree. Setting this value to a fraction less than 1 can reduce the model variance and prevent overfitting.

colsample_bytree: This setting regulates the proportion of features that will be utilized to train each tree. Setting this value to a fraction less than 1 can reduce the model variance and prevent overfitting.

Hyperparameter tuning for XGBoost can be done using methods such as grid search, random search, or Bayesian optimization. The primary idea behind such methods is to experiment with different hyperparameter combinations and then evaluate the model's performance using the validation set. The hyperparameters that give the best performance are then selected.

Mathematically, XGBoost trains multiple decision trees in sequence, where each new tree is trained to correct the errors of the previous trees. The algorithm specifically seeks to minimize a cost function which gauges the discrepancy between predicted labels and actual labels. In each iteration, the algorithm adds a new tree to the model that tries to correct the errors of the previous trees. The algorithm uses a gradient-based approach to optimize the cost function and find the optimal weights for each feature.

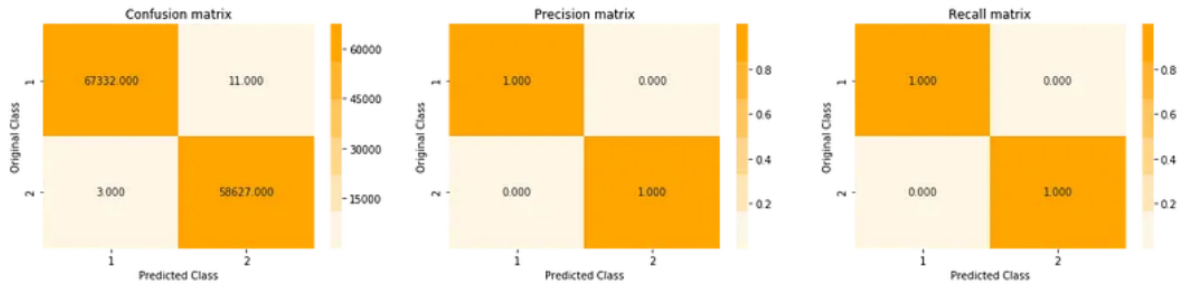


Training data f1 score is 0.999880615342634

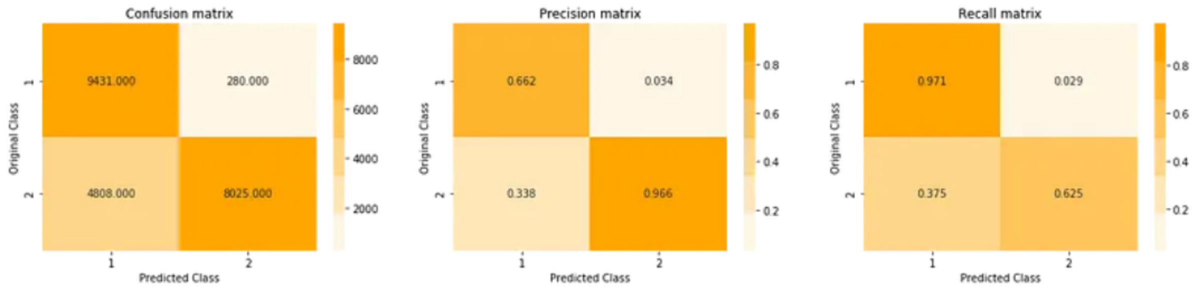
Test data f1 score is 0.7592960544990066

Training data detection rate/recall is 0.9999488316561488

Test data Detection rate/recall is 0.6253409179459207



Train confusion matrix



Test confusion matrix

Observation :

1 -> class 0 & 2 -> class 1

- The xgboost model does not work as planned.
- It cannot recollect the test data properly.

BASIC STACKING CLASSIFIER

A basic stacking classifier is the type of ensemble learning method which combines multiple base classifiers to improve prediction accuracy. In a stacking classifier, the output of the base classifiers is used as input for a meta-classifier, which learns how to combine the base classifiers to make final predictions.

The basic steps for implementing a stacking classifier are as follows:

Split the training data into two sets: the first set is used to train the base classifiers, and the second set is to train the meta-classifier.

Train several base classifiers on the first set of training data using different algorithms or configurations. Examples of base classifiers include decision trees, support vector machines, k-nearest neighbors, and neural networks.

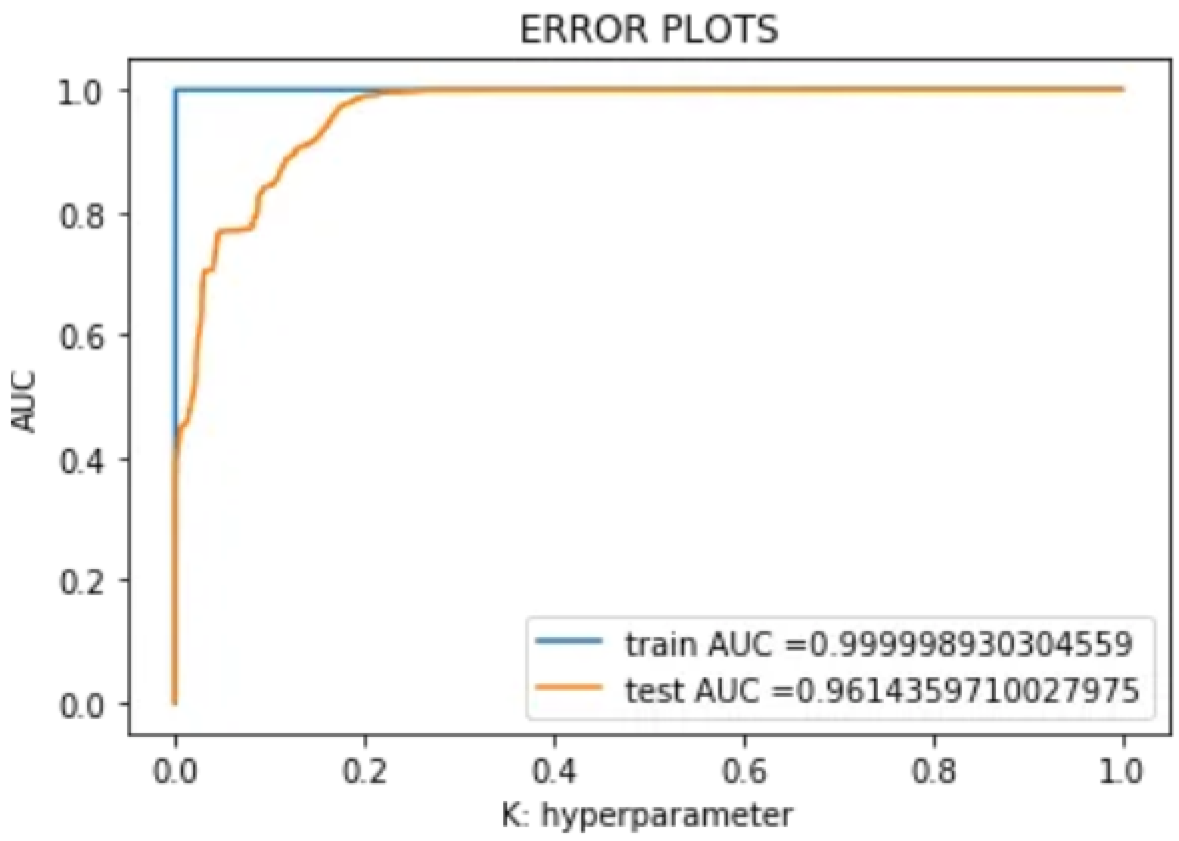
Use the base classifiers to make predictions on the second set of training data.

Use the predictions from the base classifiers as input features to train the meta-classifier. The meta-classifier can be any machine learning algorithm, but popular choices include logistic regression, decision trees, and neural networks.

Make predictions based on the test data using the meta-classifier.

Analyze the stacking classifier's performance on the test data using a statistic such as precision, accuracy, recall, or F1 score.

The basic idea behind stacking is that the different base classifiers may have different strengths and weaknesses, and by combining them with a meta-classifier, the weaknesses of one classifier can be compensated for by the strengths of another. This can lead to better prediction accuracy than any individual base classifier. However, stacking can be more computationally expensive and harder to interpret than using a single classifier.

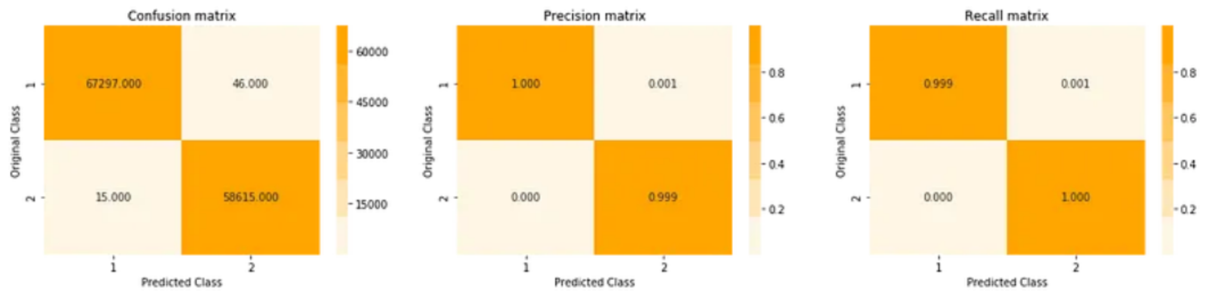


Training data f1 score is 0.9994799259960271

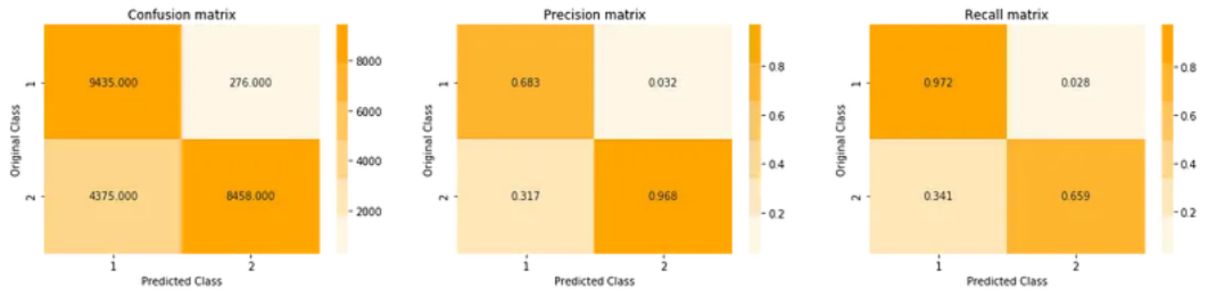
Test data f1 score is 0.7843464552325313

Training data recall score / detection rate is 0.9997441582807437

Test data recall score / detection rate is 0.6590820540793267



Train confusion matrix



Test confusion matrix

Observation:

- Above model is a standard staking model in which I took all of the models and stacked them.
- Every model output should be passed to the meta classifier.
- We also received a good auc score, but not a great f1 score or recall.

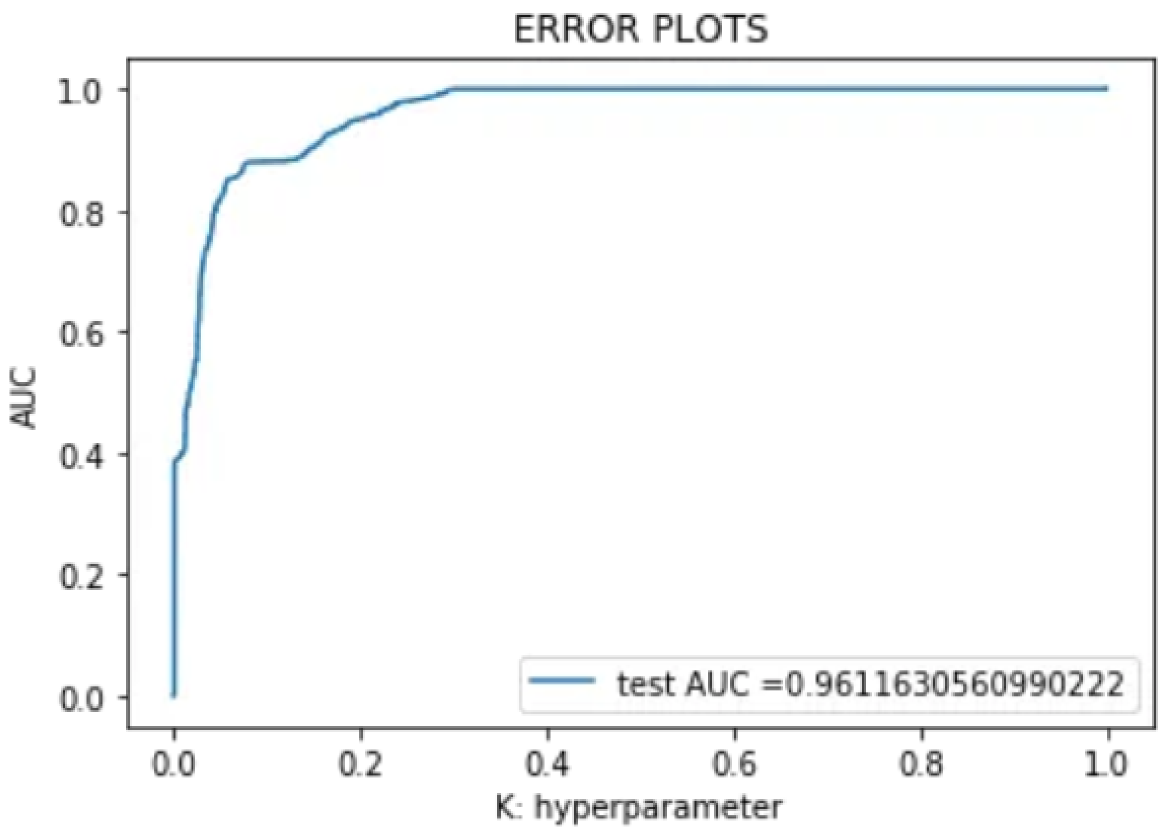
CUSTOMIZED STACKING CLASSIFIER

Method:

- Take the entire collection of data.
- What are train and test data?
- divide the data from the train into two parts: data 1 and 2 (here, we choose a 50/50 split)

- create an m-sample using the train data and the base learner (xgboost)
- now make predictions by distributing the data2 to each fitted sample.
- then use the provided predicted value and goal value of the data to train the meta classifier (meta clf : logistic regression)
- Predict the test data now by utilizing the fitted meta classifier.

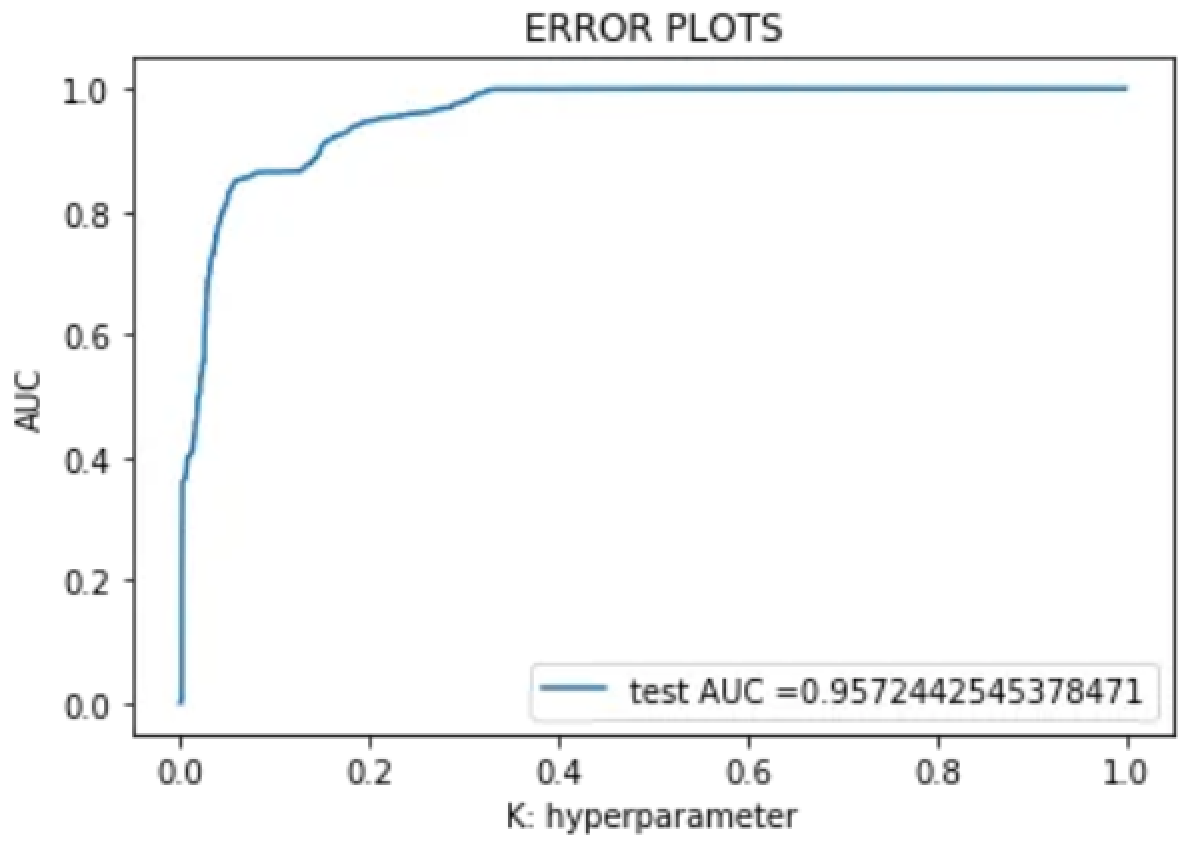
with 1000 sample



Test data f1 score is 0.7845882789317508

Test data recall score / detection rate is 0.6593158263851009

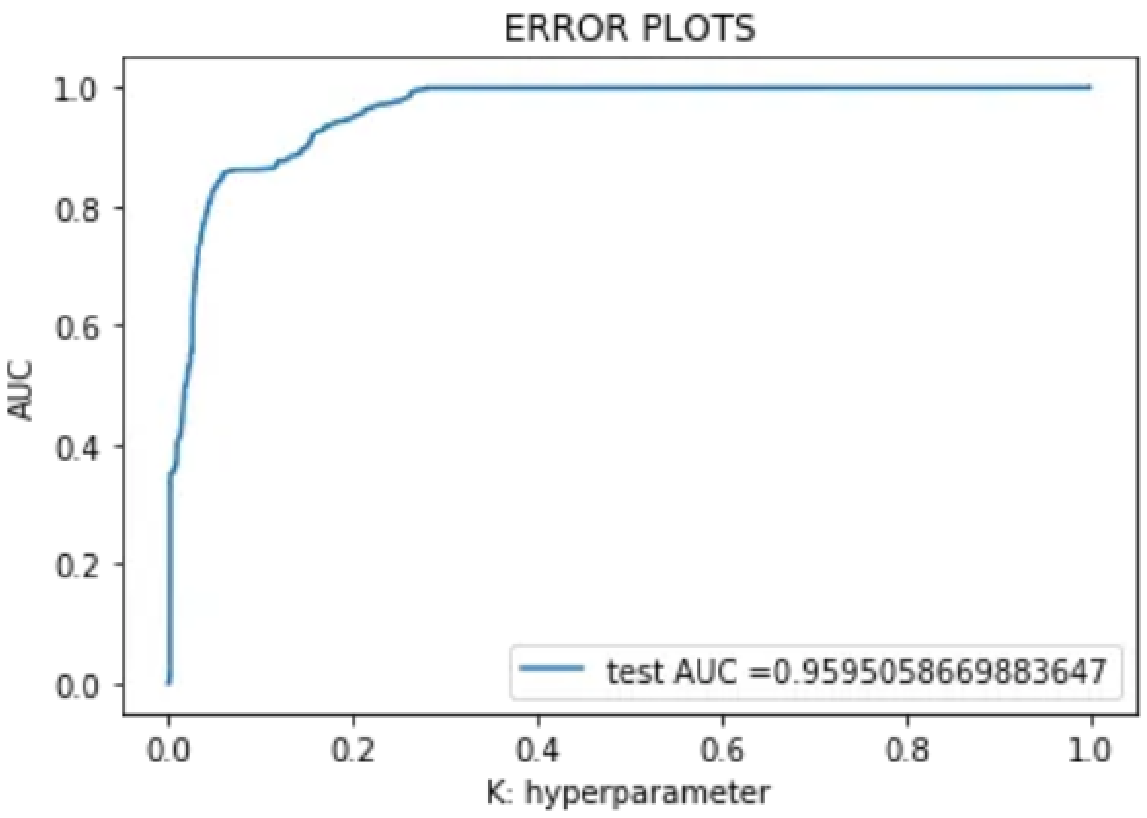
with 1500 sample



Test data f1 score is 0.7895077122009792

Test data recall score / detection rate is 0.666095223252552

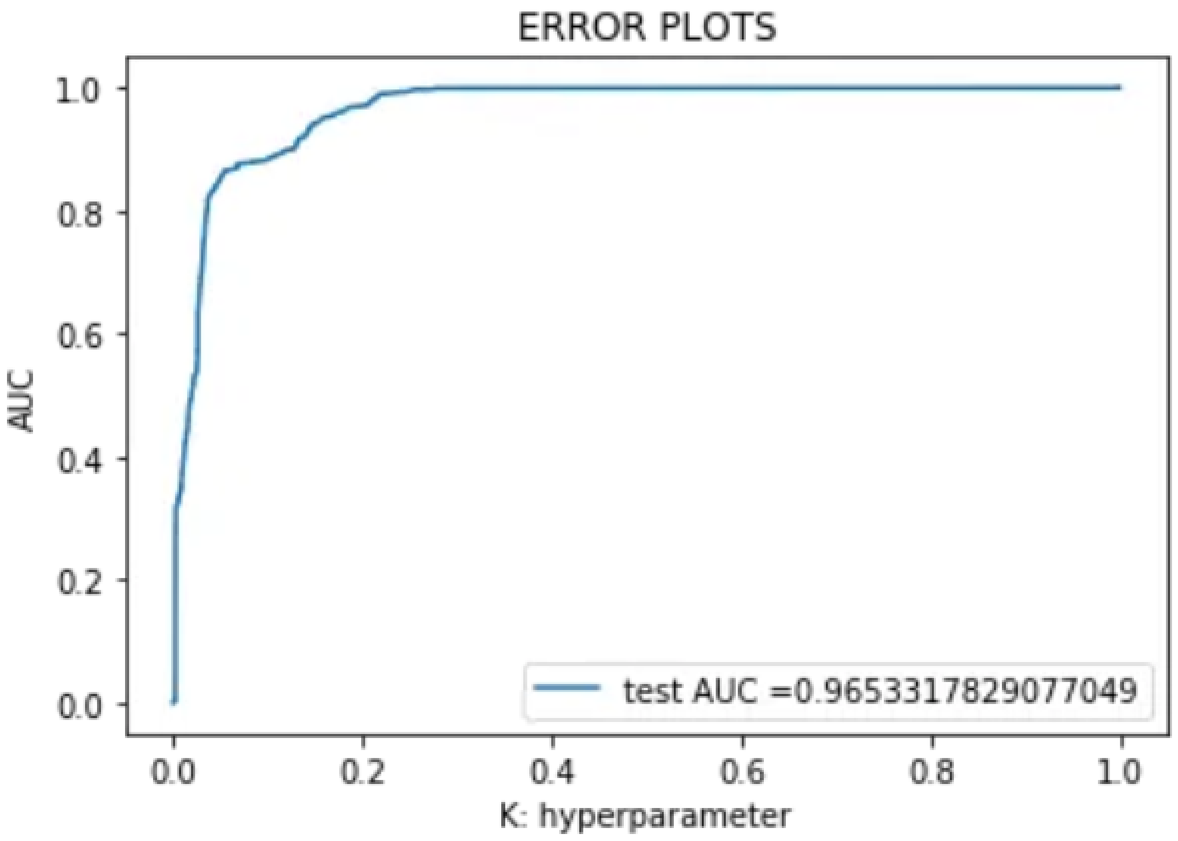
with 2000 sample



Test data f1 score is 0.7801108059034406

Test data recall score / detection rate is 0.6528481259253487

10000 data point and 100 samples

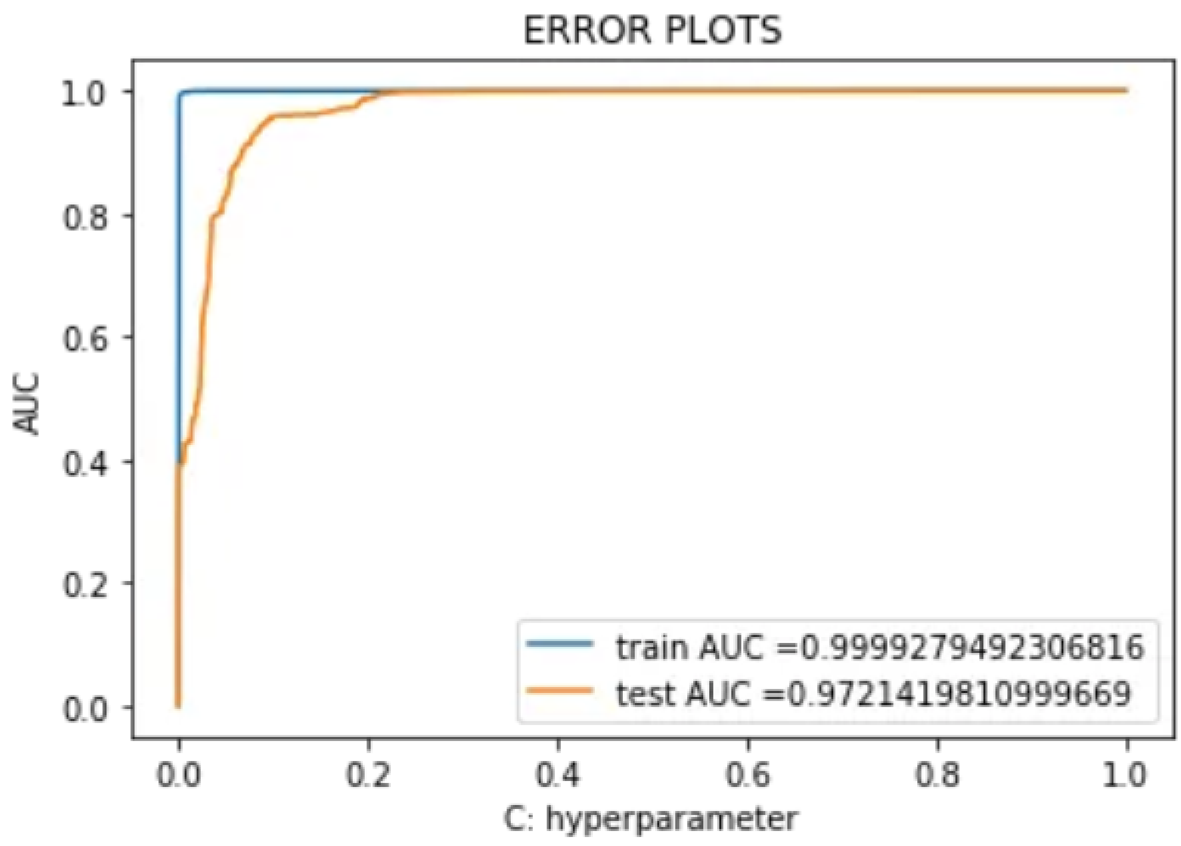


Test data f1 score is 0.7667231957791597

Test data recall score / detection rate is 0.6341463414634146

FEATURE ENGINEERING WITH RANDOM FOREST

- according to feature choices, add the two most crucial features
- use the square of a significant feature

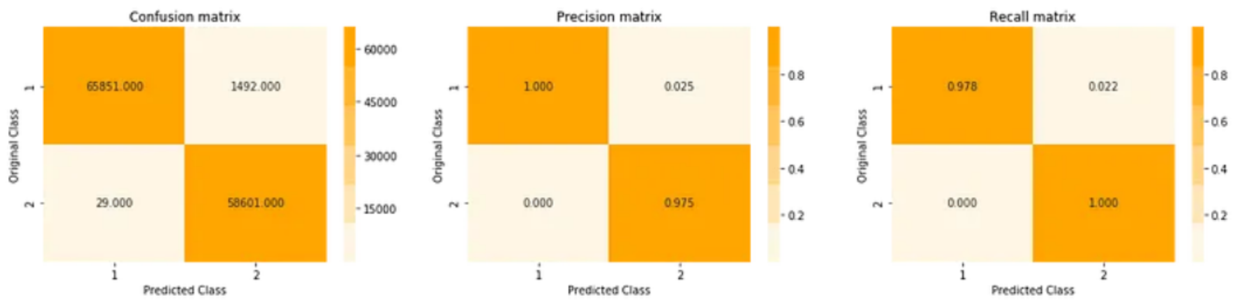


Training data f1 score is 0.9871886660545977

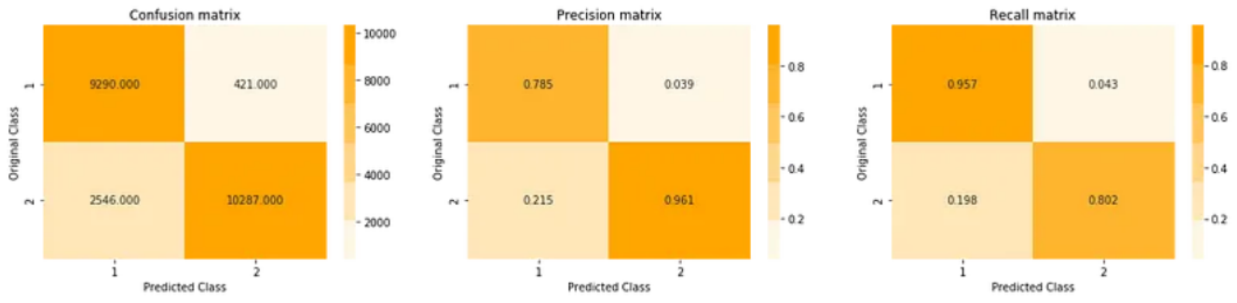
Test data f1 score is 0.8739645724480694

Training data recall score / detection rate is 0.9995053726761044

Test data recall score / detection rate is 0.8016052364996493



Train confusion matrix



Train confusion matrix

Observation:

This model produces the greatest auc value, outperforming our prior random forest model.

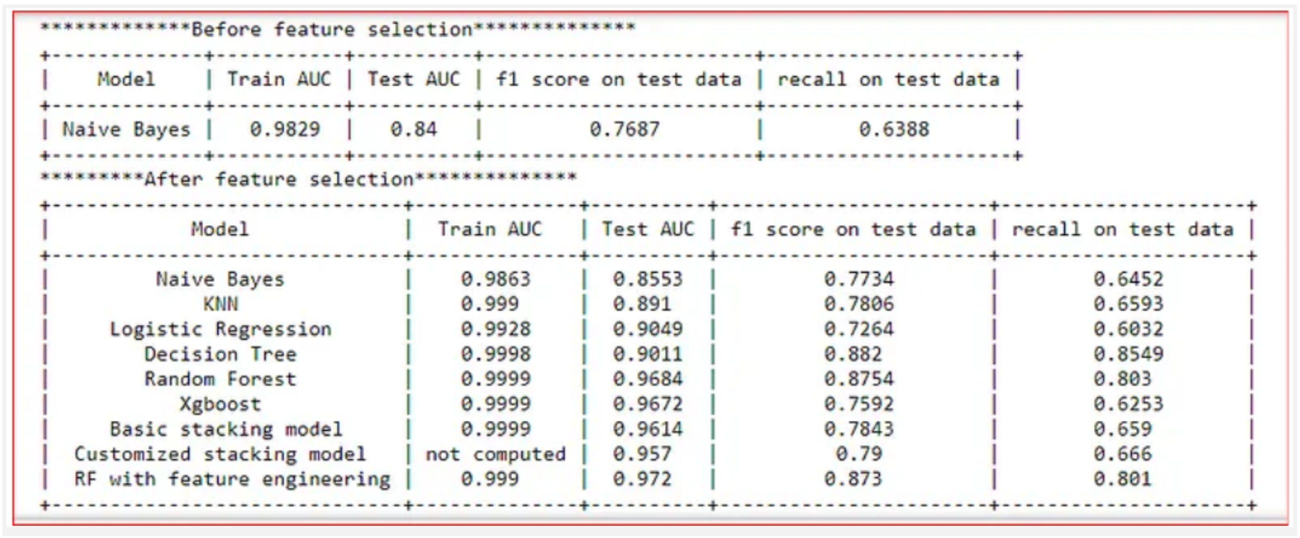
CHAPTER-4

RESULT DISCUSSION

UNDERSTANDING THE WHOLE MODEL BUILDING SECTION IN ONE SHORT

- Combine all of the encoded numerical and category features.

- to create a baseline model, we have used Naive Bayes in this example (we can take knn also)
- With hyper parameter adjustment, the naïve Bayes model appears to overfit (gap in train and test score)
- To solve this problem, we use recursive feature removal (please see "understanding feature selection" after section 3.2).
- After that, manually eliminate all of the features from the train data that are not contributing value by inspecting the important feature plot.
- Recreate the model, beginning with the naïve Bayes model.
- During feature selection, we had a good AUC, but there were a few issues, including a low f1 score and confusion with the test recall. These issues were caused by the data set's imbalance, so we added more class weight to address them. After that, we obtained rather good results.
- There are currently two effective models. Decision trees and random forests
- We had hoped Xgboost would produce the best results, but that's okay.
- We tried two types of stacking, but the result were poor in terms of f1 and recall scores.
- In the feature engineering section, We tried two new features: one is to take the features that are important according to our feature selection approach, then add those features, and the other is to square a feature.
- The result of the feature engineered model (tried random forest) is excellent; it has the best auc value We have obtained, but the f1 & recall scores are lower than those of the random forest and Decision tree.



```
*************Before feature selection*************
+-------------+-----------+----------+----------------------+---------------------+
|    Model    | Train AUC | Test AUC | f1 score on test data | recall on test data |
+-------------+-----------+----------+----------------------+---------------------+
| Naive Bayes |   0.9829  |   0.84   |        0.7687        |        0.6388       |
+-------------+-----------+----------+----------------------+---------------------+
*********After feature selection*************
+---------------------------+---------------+----------+----------------------+---------------------+
|           Model           |   Train AUC   | Test AUC | f1 score on test data | recall on test data |
+---------------------------+---------------+----------+----------------------+---------------------+
|        Naive Bayes        |     0.9863    |  0.8553  |        0.7734        |        0.6452       |
|            KNN            |     0.999     |  0.891   |        0.7806        |        0.6593       |
|    Logistic Regression    |     0.9928    |  0.9049  |        0.7264        |        0.6032       |
|       Decision Tree       |     0.9998    |  0.9011  |        0.882         |        0.8549       |
|       Random Forest       |     0.9999    |  0.9684  |        0.8754        |        0.803        |
|          Xgboost          |     0.9999    |  0.9672  |        0.7592        |        0.6253       |
|    Basic stacking model   |     0.9999    |  0.9614  |        0.7843        |        0.659        |
|  Customized stacking model | not computed |  0.957   |         0.79         |        0.666        |
|  RF with feature engineering |   0.999    |  0.972   |        0.873         |        0.801        |
+---------------------------+---------------+----------+----------------------+---------------------+
```

CHAPTER-5

CONCLUSION AND FUTURE SCOPE

Intrusion detection systems (IDS) are an important component of modern cybersecurity, helping to protect computer networks and systems from unauthorized access and attacks. Machine learning (ML) techniques have shown great promise in improving the accuracy and effectiveness of IDS, by enabling automated detection and classification of different types of network traffic and user behavior.

The project "Intrusion detection system using machine learning" is a valuable contribution to the field of cybersecurity, as it demonstrates the feasibility and effectiveness of using ML for IDS. The project involved collecting and preprocessing network traffic data, and then using different ML algorithms, such as decision trees, random forests to train and evaluate intrusion detection models. The results showed that ML-based IDS can achieve high accuracy in detecting different types of network attacks and anomalies, and can be a valuable tool for network security professionals.

In terms of future scope, there are several directions in which this project could be extended and improved. Here are some examples:

More advanced ML techniques: The project used several popular ML algorithms for IDS, but there are many other techniques that could be explored, such as deep learning, reinforcement learning, and unsupervised learning. These techniques may be able to identify more subtle patterns and anomalies in network traffic that are difficult to detect using traditional rule-based approaches.

Real-time detection: The project focused on offline detection of network attacks, but real-time detection is also an important area of research for IDS. Real-time detection involves analyzing network traffic as it is happening, and making decisions about whether the traffic is suspicious or not in real-time.

This requires efficient and scalable ML algorithms that can process large amounts of data quickly.

Integration with other security systems: An IDS is just one component of a comprehensive security system, and integrating it with other systems such as firewalls, intrusion prevention systems (IPS), and security information and event management (SIEM) tools can provide even greater protection against network attacks. Future research could explore how ML-based IDS can be integrated with other security systems and how these systems can be combined to provide a more holistic view of network security.

Overall, the project "Intrusion detection system using machine learning" provides a valuable contribution to the field of cybersecurity and demonstrates the potential of ML for improving network security. There is still much to be done in this area, and future research can build on the work done in this project to develop even more effective and efficient IDS solutions.

REFERENCES

1. Buczak, A.L.; Guven, E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun. Surv. Tutor.* **2015**, *18*, 1153–1176.

2. Alshammari, A., Aldribi, A. Apply machine learning techniques to detect malicious network traffic in cloud computing. J Big Data 8, 90 (2021).

3.N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," 2015 Military Communications and Information Systems Conference (MilCIS), Canberra, ACT, Australia, 2015, pp. 1-6, doi: 10.1109/MilCIS.2015.7348942.

4. Garcia-Teodoro, P., Diaz-Verdejo, J. E., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & Security, 28(1-2), 18-28.

5. Kaur, R., & Singh, S. (2018). Intrusion detection system using machine learning: A review. International Journal of Innovative Technology and Exploring Engineering (IJITEE), 8(5), 1495-1499.

6. Khan, M. A., & Hussain, M. (2020). A review of machine learning-based intrusion detection systems. Journal of Network and Computer Applications, 162, 102536.

7. Li, Y., Li, T., & Li, J. (2018). Survey on machine learning techniques in intrusion detection systems. Journal of Network and Computer Applications, 107, 12-25.

8. Liu, Y., Wang, B., Zhang, L., & Zhao, J. (2020). A survey on machine learning for intrusion detection. Wireless Networks, 26(4), 2265-2278.

9. Ouaddah, A., Rachdaoui, R., & Elkalam, A. A. (2019). A systematic review of machine learning-based intrusion detection research. Journal of Network and Computer Applications, 133, 1-18.

10. Ramakrishnan, R., Kumar, N., & Karthik, S. (2021). A review of machine learning techniques for intrusion detection system. Journal of Ambient Intelligence and Humanized Computing, 12(6), 5925-5941.