**Project Report**

**on**

**Intrusion Detection**

**Using**

**Machine Learning Algorithms**

Under the Guidance of

Dr. Chaoyang Zhang

Group : 2

Submitted By:

Srikanth Moolagundla (W10156092)

Gopi Nagabhiru (W10175847)

**ABSTRACT:**

Network environments must be protected against malicious activity and unauthorized access via intrusion detection systems (IDS). Novel and complex threat detection is a limitation of traditional signature-based intrusion detection systems. In order to improve intrusion detection capabilities, this project uses machine learning methods to address these issues. We concentrate on three well-known algorithms: Deep Neural Networks (DNN), Support Vector Machines (SVM), and Random Forest (RF). We want to preprocess the data, apply, and assess these models to compare their efficacy in spotting network intrusions using the labeled network traffic data from the NSL-KDD dataset.

The pretreatment stages for the data were dividing the dataset into training and testing subsets, normalizing feature values, and handling categorical features using one-hot encoding. Every model underwent training and optimization processes: RandomizedSearchCV was utilized to refine RF, SVM was experimented with diverse kernels and hyperparameters, and a DNN with multiple architectures and training approaches was created. AUC, F1-score, recall, accuracy, precision, and other performance evaluation metrics were utilized to evaluate the efficacy of each model.

Though computationally demanding, the Deep Neural Network produced impressive results with complicated pattern identification. An examination of these models' comparative aspects revealed important details about their advantages and disadvantages in relation to intrusion detection. This study demonstrates how machine learning techniques can improve network security and makes recommendations for further research and model advancements.

**SOFTWARE TOOLS USED :**

The entire code is written and executed in jupyter notebook with the help of python programming language.

Some of the packages used in jupyter notebook are as follows:

Pandas - Used for reading .csv files and data analysis.

Matplotlib – Used for Data Visualization.

Seaborn - Also used for Data Visualisation and in an enhanced state.

Numpy – It is used for numerical computing in python.

Scikit-learn – It is used in machine learning for implementing various classification and regression.

**INTRODUCTION:**

In cybersecurity, intrusion detection systems (IDS) are essential for spotting hostile activity and illegal access to network systems. IDSs based on traditional signatures are constrained in that they cannot identify novel or unidentified threats. A potential answer is provided by machine learning (ML), which allows intrusion detection systems (IDS) to recognize patterns in past data that point to possible invasions. In order to enhance intrusion detection capabilities, this research investigates the use of machine learning methods.

**PROBLEM OF INTRUSION DETECTION:**

In today's digital world, intrusion detection has become a pressing issue. Nevertheless, the frequency and sophistication of cyberattacks are rising. By keeping an eye on system activity and network traffic and looking for potential infractions, intrusion detection systems establish a crucial first line of protection. As a result of traditional rule-based systems' incapacity to handle ever-increasing complexity, machine learning approaches have once again shown themselves to be extremely effective instruments for boosting IDS's capabilities.

**LITERATURE REVIEW OF MACHINE LEARNING IN INTRUSION DETECTION:**

Machine learning has been extensively studied for intrusion detection, with various algorithms demonstrating different strengths:

Random Forest (RF): A group technique that constructs several decision trees and combines their results. Robustness, handling of big datasets, and interpretability through feature importance are the hallmarks of reinforcement learning. Research demonstrates that by taking advantage of the diversity of trees, RF performs well in identifying incursions and routine activities.  
  
SVM: A supervised learning model called Support Vector Machines (SVM) determines the best hyperplane in high-dimensional space to divide classes. Because SVMs create decision boundaries that optimize the margin between classes, they are good at detecting intrusions. Different data distributions can be handled with flexibility thanks to variants like the RBF and linear kernels.  
  
DNN: Multiple layers of neurons make up Deep Neural Networks (DNNs), which are capable of learning intricate, non-linear correlations from input.

**IMPORTANCE OF USING RANDOM FOREST, SVM, AND DNN:**

Comparing RF, SVM, and DNN allows for a comprehensive evaluation of their effectiveness in intrusion detection. Each algorithm offers unique benefits:

* RF provides robustness and feature importance metrics.
* SVM excels in high-dimensional spaces and various kernel functions.
* DNN captures complex patterns and interactions among features.

Understanding these models' performance helps in selecting the most suitable algorithm for practical IDS applications.

**OVERVIEW OF THE DATASET and DATA PREPROCESSING:**

The dataset is downloaded from Kaggle famously known as NSL-KDD dataset. A popular benchmark for assessing intrusion detection systems is the NSL-KDD dataset. It is comprised of network traffic data with attributes such protocol type, service, connection duration, and a label designating attack or regular traffic. Training and test sets are included in the dataset.

The attributes in the dataset which includes numerical and categorical features are as follows:

'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', 'rerror\_rate', 'srv\_rerror\_rate','same\_srv\_rate','diff\_srv\_rate','srv\_diff\_host\_rate','dst\_host\_count','dst\_host\_srv\_count','dst\_host\_same\_srv\_rate','dst\_host\_diff\_srv\_rate','dst\_host\_same\_src\_port\_rate','dst\_host\_srv\_diff\_host\_rate', 'dst\_host\_serror\_rate','dst\_host\_srv\_serror\_rate', 'dst\_host\_rerror\_rate', 'dst\_host\_srv\_rerror\_rate', 'attack', 'difficulty\_level']

**Features**: 41 attributes including categorical and numerical features.

**Label**: The attack column categorizes traffic as 'normal' or various attack types (e.g., DoS, Probe).

**Decription of the attributes:**

1. Duration: the extent of the association
2. protocol\_type: the protocol type (such as tcp or udp); • service: the destination network service (such as http or telnet).
3. flag: the connection's error-free or normal state
4. Land: 1 if the connection is from/to the same host/port; 0 otherwise.
5. src\_bytes: number of data bytes from source to destination.
6. dst\_bytes: number of data bytes from destination to source.
7. urgent: the quantity of urgent packets;
8. wrong\_fragment: the number of "wrong" pieces
9. dst\_host\_count: number of connections to the same host
10. dst\_host\_srv\_count: number of connections to the same service
11. dst\_host\_same\_srv\_rate: % of connections to the same service
12. dst\_host\_diff\_srv\_rate: % of connections to different services
13. dst\_host\_same\_src\_port\_rate: % of connections from the same source port
14. dst\_host\_srv\_diff\_host\_rate: % of connections to different hosts
15. dst\_host\_serror\_rate: % of connections that have "SYN" errors
16. dst\_host\_srv\_serror\_rate: % of connections that have "SYN" errors
17. dst\_host\_rerror\_rate: % of connections that have "REJ" errors
18. dst\_host\_srv\_rerror\_rate: % of connections that have "REJ" errors
19. hot: number of "hot" indicators
20. num\_failed\_logins: number of failed login attempts
21. logged\_in: 1 if successfully logged in; 0 otherwise
22. num\_compromised: number of "compromised" conditions
23. root\_shell: 1 if root shell is obtained; 0 otherwise
24. su\_attempted: 1 if "su root" command attempted; 0 otherwise
25. num\_root: number of "root" accesses
26. num\_file\_creations: number of file creation operations
27. num\_shells: number of shell prompts
28. num\_access\_files: number of operations on access control files
29. num\_outbound\_cmds: number of outbound commands in an ftp session
30. is\_host\_login: 1 if the login belongs to the "host" list; 0 otherwise
31. is\_guest\_login: 1 if the login is a "guest" login; 0 otherwise
32. count: number of connections to the same host as the current connection in the past two seconds
33. srv\_count: number of connections to the same service as the current connection in the past two seconds
34. serror\_rate: % of connections that have "SYN" errors
35. srv\_serror\_rate: % of connections that have "SYN" errors
36. rerror\_rate: % of connections that have "REJ" errors
37. srv\_rerror\_rate: % of connections that have "REJ" errors
38. same\_srv\_rate: % of connections to the same service
39. diff\_srv\_rate: % of connections to different services • srv\_diff\_host\_rate: % of connections to different hosts
40. attack:it is the target variable with 0 and 1 as normal or intrusion is there.

**Handling Missing Values**

The dataset was checked for missing values. If missing values were present, strategies like mean imputation or removal would be used. In this case, no missing values were detected.

**Encoding Categorical Features**

Categorical features were encoded using One-Hot Encoding to convert them into a format suitable for machine learning models. This involves creating binary columns for each category.

**Normalizing/Standardizing Feature Values**

Feature values were standardized to have zero mean and unit variance, which improves the performance of many machine learning algorithms

**MODEL IMPLEMENTATION:**

**RANDOM FOREST:**

Using an ensemble learning technique called Random Forest, overfitting is reduced and accuracy is increased by combining the predictions of several decision trees. During training, it builds a large number of decision trees, and it produces a class that is the mean prediction (regression) or the mode of the classes (classification) of the individual trees.

**Specifics of Implementation:**

Library Scikit-learn was utilized.

Hyper-parameters:

* n\_estimators: tuned The count of trees within the forest.
* max\_depth: The trees' maximum depth.
* The bare minimum of samples needed to separate an internal node is called min\_samples\_split.
* Optimization: By sampling from a distribution of potential values, RandomizedSearchCV was used to determine the ideal hyperparameters.

**SUPPORT VECTOR MACHINES:**

Support Vector Machines (SVM) are supervised learning models used for classification and regression tasks. SVMs are effective in high-dimensional spaces and are versatile with the use of different kernel functions to handle various data distributions. The goal is to find the optimal hyperplane that best separates the data into different classes.

**Implementation Details:**

Library Used: scikit-learn

Kernels Experimented With:

* Linear Kernel: Suitable for linearly separable data.
* Polynomial Kernel: Captures interactions between features up to a specified degree.
* RBF Kernel: Handles non-linear relationships by mapping data into a higher-dimensional space.

Hyperparameters Tuned:

* C: Regularization parameter.
* kernel: Type of kernel function.
* gamma: Kernel coefficient for 'rbf', 'poly', and 'sigmoid' kernels.

**DEEP NEURAL NETWORKS:**

Deep Neural Networks (DNN) consist of multiple layers of neurons, allowing them to model complex, non-linear relationships in data. DNNs can capture intricate patterns through their architecture, which typically includes an input layer, one or more hidden layers, and an output layer.

**Implementation Details:**

Library Used: TensorFlow/Keras

Architecture:

Input Layer: Matches the number of features.

Hidden Layers: Multiple layers with varying numbers of neurons. Activation functions such as ReLU (Rectified Linear Unit) were used.

Output Layer: A single neuron with a sigmoid activation function for binary classification.

Hyperparameters Tuned:

Number of Layers: Varies from 2 to 4 layers.

Neurons per Layer: Typically between 32 and 128.

Learning Rate: Adjusted to optimize convergence.

Batch Size: The number of samples per gradient update.

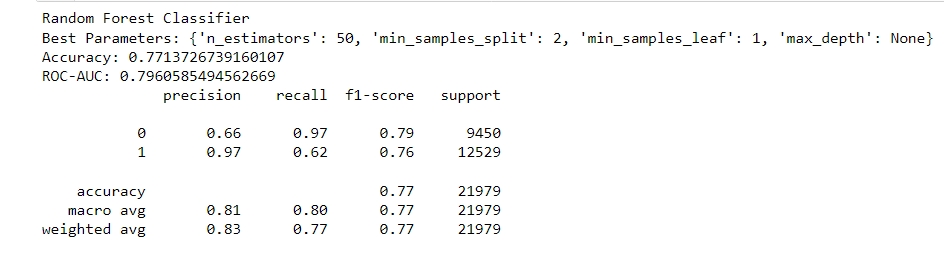
Epochs: Number of iterations over the entire dataset.

**RESULTS:**

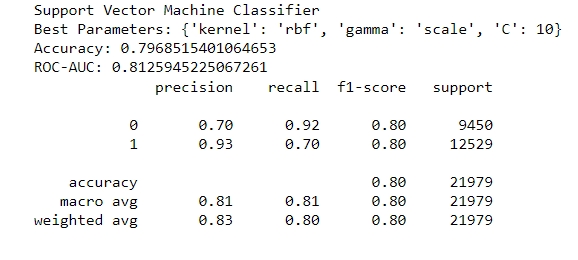
Each model is evaluated by using the performance metrics:

* **Accuracy**: The proportion of correctly classified instances.
* **Precision**: The proportion of true positives among the predicted positives.
* **Recall**: The proportion of true positives among the actual positives.
* **F1-Score**: The harmonic mean of precision and recall.
* **ROC-AUC**: The area under the receiver operating characteristic curve

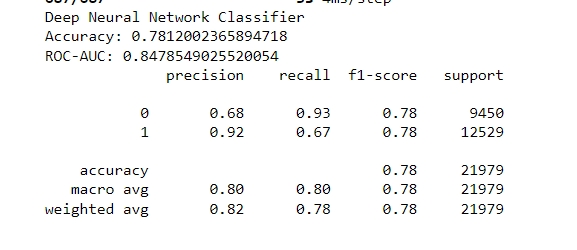
The metrics for each of the algorithms are as follows:

**RANDOM FOREST:**

**SUPPORT VECTOR MACHINES:**



**DEEP NEURAL NETWORK :**



**DISCUSSION OF RESULTS:**

Support vector machine gives the highest accuracy when compared to other algorithms for the dataset.

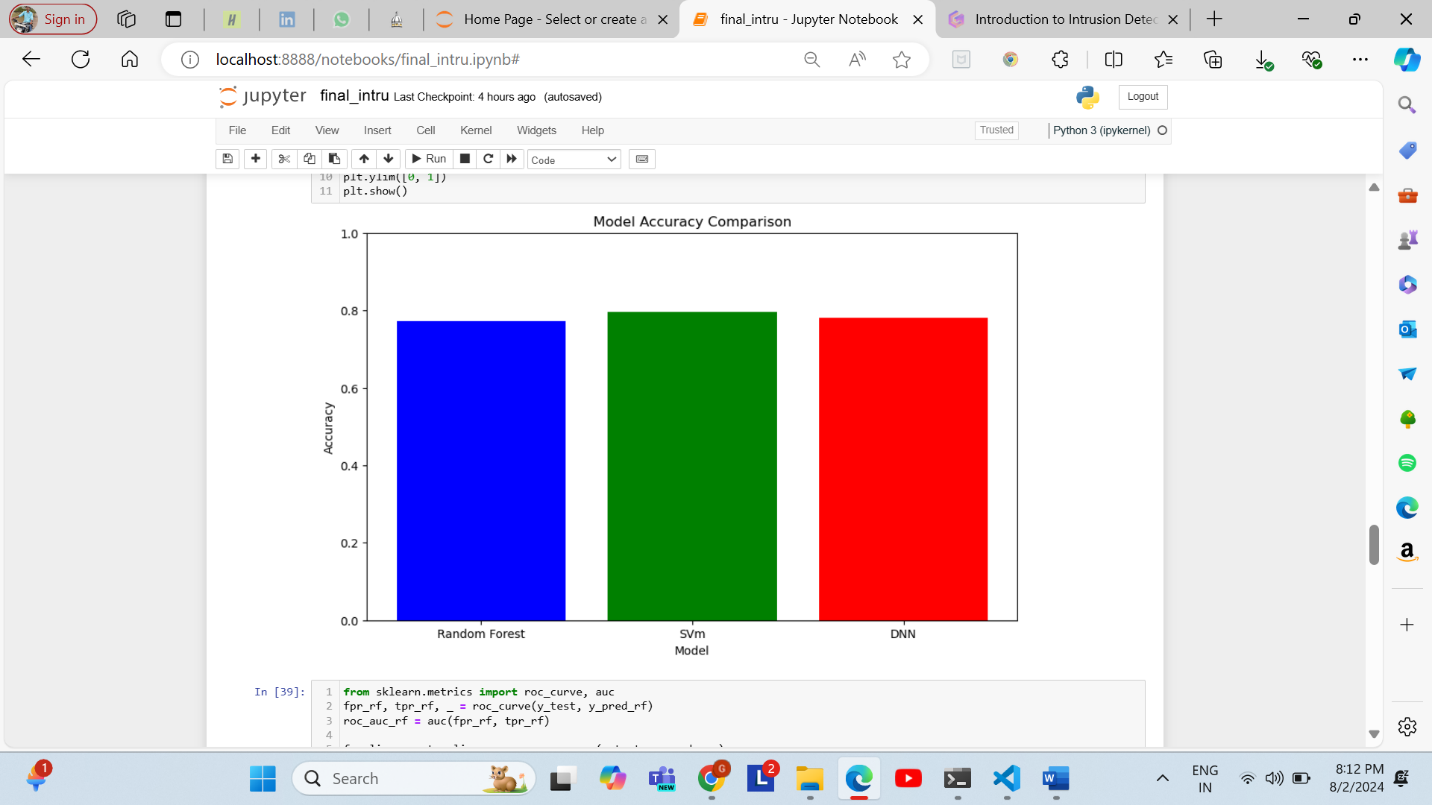
With a ROC-AUC score of 0.796, the Random Forest Classifier attained an accuracy of 77.1%. The model demonstrates efficacy in managing the dataset's imbalance, as evidenced by its high recall for regular traffic, which accurately detects the majority of normal instances. Its accuracy for regular traffic is worse, though, thus some attacks might get mistakenly identified as regular. The high precision of the model for attacks indicates that it can reliably detect attacks, however it may miss certain attacks because of its lower recall. With strong macro and weighted averages and a balanced overall performance, the categorization ability is strong.

The ROC-AUC score of 0.813 and accuracy of 79.7% were attained by the SVM Classifier. For attacks, it does exceptionally well in differentiating between attack and normal occurrences with a high ROC-AUC score. Although the model's precision for normal traffic is lower, showing that some attacks are incorrectly categorized as normal, its high recall for regular traffic suggests that it accurately recognizes the majority of normal incidents. The weighted averages and balanced macro revealed by the SVM point to great overall performance. The SVM is a useful tool despite its processing complexity because of its ability to handle complicated patterns and attain high precision for assaults.

The DNN Classifier had a ROC-AUC score of 0.848 and an accuracy of 78.1%. It performs well in identifying intricate patterns, exhibiting excellent precision during attacks and great recall during regular traffic. The strong ROC-AUC score of the DNN suggests good class separation. On the other hand, the lower recall for attacks suggests that some threats are overlooked, while the lower precision for regular traffic implies that some attacks may be mistakenly categorized as normal. The model is a good competitor because to its competitive performance and capacity to handle complicated data; nevertheless, more resource management and optimization are required.

**COMPARISON OF RESULTS:**

The visual representation of the accuracies are as follows:



**Accuracy**:

* **SVM**: Highest accuracy (79.7%)
* **DNN**: Second-highest accuracy (78.1%)
* **Random Forest**: Lowest accuracy (77.1%)

**ROC-AUC**:

* **DNN**: Highest ROC-AUC (0.848)
* **SVM**: Second-highest ROC-AUC (0.813)
* **Random Forest**: Lowest ROC-AUC (0.796)

**Precision and Recall**:

* **SVM**: Highest precision for attack detection (93%), good recall for normal traffic (92%).
* **DNN**: High precision for attack detection (92%) and high recall for normal traffic (93%).
* **Random Forest**: High recall for normal traffic (97%), but lower precision and recall for attack detection.

**CONCLUSION:**

Every model has advantages and disadvantages. The SVM has a low recall for attacks but a high accuracy and precision. The DNN performs well in identifying intricate patterns and attaining a high ROC-AUC, however more optimization could be necessary. For typical traffic, the Random Forest model offers a balanced performance with good recall; however, attack detection still need work. The intrusion detection system's unique criteria, such as the need for precision, recall, or overall accuracy, will determine which type is ideal.

**FUTURE ENHANCEMENTS AND SCOPE:**

In the future we would like to implement a real time intrusion detection system. Implementing adaptive learning systems that continuously update and refine models based on new data could help in keeping pace with evolving attack patterns and techniques.

Implementing more algorithms to check if accuracy can be improved.

In the future we would like to detect what type of attack is made by the attacker.

**REFERENCES:**

* M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Data Set," *2010 IEEE International Conference on Computational Intelligence for Security and Defense Applications*, 2010
* A. A. Al-Omari and A. A. Al-Baali, "Network Intrusion Detection Using Random Forest Classifier," *International Journal of Computer Applications*, vol. 125, no. 2, pp. 11-17, 2015.
* M. A. Rajab and D. K. K. Kumar, "Intrusion Detection Using Support Vector Machine: A Review," *International Journal of Computer Applications*, vol. 163, no. 6, pp. 1-7, 2016.

**Appendix-1**

import pandas as pd

# Load the datasets

kdd\_train = pd.read\_csv('KDDTrain+\_20Percent.txt', header=None)

kdd\_test = pd.read\_csv('KDDTest+.csv', header=None)

kdd\_test\_21 = pd.read\_csv('KDDTest-21.csv', header=None)

# Print the first few rows

print(kdd\_train.head())

print(kdd\_test.head())

print(kdd\_test\_21.head())

# Print the number of columns in each dataset

print(f'KDDTrain+ columns: {kdd\_train.shape[1]}')

print(f'KDDTest+ columns: {kdd\_test.shape[1]}')

print(f'KDDTest-21 columns: {kdd\_test\_21.shape[1]}')

columns = ['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', 'rerror\_rate', 'srv\_rerror\_rate',

'same\_srv\_rate', 'diff\_srv\_rate', 'srv\_diff\_host\_rate', 'dst\_host\_count',

'dst\_host\_srv\_count', 'dst\_host\_same\_srv\_rate', 'dst\_host\_diff\_srv\_rate',

'dst\_host\_same\_src\_port\_rate', 'dst\_host\_srv\_diff\_host\_rate', 'dst\_host\_serror\_rate',

'dst\_host\_srv\_serror\_rate', 'dst\_host\_rerror\_rate', 'dst\_host\_srv\_rerror\_rate',

'attack', 'difficulty\_level']

# Add column names to the datasets

kdd\_train.columns = columns

kdd\_test.columns = columns

kdd\_test\_21.columns = columns

# Handle missing values if any

kdd\_train = kdd\_train.dropna()

kdd\_test = kdd\_test.dropna()

kdd\_test\_21 = kdd\_test\_21.dropna()

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.pipeline import Pipeline

from sklearn.model\_selection import RandomizedSearchCV

# Define the encoder with handle\_unknown='ignore'

encoder = ColumnTransformer(

transformers=[

('cat', OneHotEncoder(handle\_unknown='ignore'), categorical\_features)

],

remainder='passthrough'

)

# Split dataset into features and labels for the full dataset

X\_train = kdd\_train.drop(['attack', 'difficulty\_level'], axis=1)

y\_train = kdd\_train['attack']

X\_test = kdd\_test.drop(['attack', 'difficulty\_level'], axis=1)

y\_test = kdd\_test['attack']

# Encode labels

y\_train = y\_train.map(lambda x: 0 if x == 'normal' else 1)

y\_test = y\_test.map(lambda x: 0 if x == 'normal' else 1)

# Normalize or standardize the feature values

scaler = StandardScaler()

# Create preprocessing pipeline

preprocessor = Pipeline(steps=[

('encoder', encoder),

('scaler', scaler)

])

# Preprocess the data

X\_train = preprocessor.fit\_transform(X\_train)

X\_test = preprocessor.transform(X\_test)

# Create and train models with the smaller subset for initial testing

# Random Forest

rf = RandomForestClassifier(random\_state=42)

param\_dist\_rf = {

'n\_estimators': [50, 100],

'max\_depth': [10, 20, None],

'min\_samples\_split': [2, 5],

'min\_samples\_leaf': [1, 2]

}

random\_search\_rf = RandomizedSearchCV(estimator=rf, param\_distributions=param\_dist\_rf, n\_iter=10, cv=3, random\_state=42, n\_jobs=-1)

random\_search\_rf.fit(X\_train, y\_train)

best\_rf = random\_search\_rf.best\_estimator\_

y\_pred\_rf = best\_rf.predict(X\_test)

print("Random Forest Classifier")

print("Best Parameters:", random\_search\_rf.best\_params\_)

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_rf))

print("ROC-AUC:", roc\_auc\_score(y\_test, y\_pred\_rf))

print(classification\_report(y\_test, y\_pred\_rf))

from sklearn.svm import SVC

X\_train\_sample = X\_train[:10000]

y\_train\_sample = y\_train[:10000]

# Support Vector Machines

svm = SVC(probability=True, random\_state=42)

param\_dist\_svm = {

'kernel': ['linear', 'rbf'],

'C': [0.1, 1, 10],

'gamma': ['scale', 'auto']

}

# Limit n\_iter for faster computation

random\_search\_svm = RandomizedSearchCV(estimator=svm, param\_distributions=param\_dist\_svm, n\_iter=5, cv=3, random\_state=42, n\_jobs=-1)

random\_search\_svm.fit(X\_train\_sample, y\_train\_sample)

best\_svm = random\_search\_svm.best\_estimator\_

y\_pred\_svm = best\_svm.predict(X\_test)

print("Support Vector Machine Classifier")

print("Best Parameters:", random\_search\_svm.best\_params\_)

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_svm))

print("ROC-AUC:", roc\_auc\_score(y\_test, y\_pred\_svm))

print(classification\_report(y\_test, y\_pred\_svm))

# Deep Neural Networks

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, BatchNormalization

from tensorflow.keras.callbacks import EarlyStopping

def create\_dnn\_model(input\_dim):

model = Sequential([

Dense(32, activation='relu', input\_dim=input\_dim),

BatchNormalization(),

Dropout(0.5),

Dense(16, activation='relu'),

BatchNormalization(),

Dropout(0.5),

Dense(1, activation='sigmoid')

])

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

return model

input\_dim = X\_train.shape[1]

dnn\_model = create\_dnn\_model(input\_dim)

early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

history = dnn\_model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_split=0.2, callbacks=[early\_stopping])

y\_pred\_dnn\_prob = dnn\_model.predict(X\_test)

y\_pred\_dnn = (y\_pred\_dnn\_prob > 0.5).astype(int)

print("Deep Neural Network Classifier")

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_dnn))

print("ROC-AUC:", roc\_auc\_score(y\_test, y\_pred\_dnn\_prob))

print(classification\_report(y\_test, y\_pred\_dnn))

print("Model Comparison:")

print(f"Random Forest Accuracy: {accuracy\_score(y\_test, y\_pred\_rf):.4f}")

print(f"SVC Accuracy: {accuracy\_score(y\_test, y\_pred\_svm):.4f}")

print(f"Deep Neural Network Accuracy: {accuracy\_score(y\_test, y\_pred\_dnn):.4f}")

import matplotlib.pyplot as plt

models = ['Random Forest', 'SVm', 'DNN']

accuracies = [accuracy\_score(y\_test, y\_pred\_rf), accuracy\_score(y\_test, y\_pred\_svm), accuracy\_score(y\_test, y\_pred\_dnn)]

plt.figure(figsize=(10, 6))

plt.bar(models, accuracies, color=['blue', 'green', 'red'])

plt.xlabel('Model')

plt.ylabel('Accuracy')

plt.title('Model Accuracy Comparison')

plt.ylim([0, 1])

plt.show()

appendix-2

|  |  |
| --- | --- |
| Name | workdone |
| Gopi nagabhiru | Find the dataset and perform the preprocessing tasks on the data encode the features hyperparameters tuning |
| Srikanth moolagundla | Created the model RF, SVM, DNN and trained them, tested them and then compared the accuracies to find the best model |

There is a collective work that involves both of us in preparing the report and presentation and finding the research papers which serves a reference for the problem.