**SYNOPSIS**

The emerging challenge of cognitive intrusions within the metaverse, where users engage in complex virtual experiences. Unlike traditional security paradigms, cognitive intrusions involve subtle manipulations of users' perceptions, interactions, and decision-making processes, making them challenging to detect using conventional methods.

This project introduces a novel approach to predict cognitive intrusions in the metaverse environment by leveraging advanced machine learning techniques. By analysing user interactions, communication patterns, and virtual behaviour, our model aims to pre-emptively detect and categorize potential intrusions before they compromise the metaverse experience

**SYSTEM ENVIRONMENT**

2.1 Hardware Requirements:

Processor : Intel Core i4 (10th Gen)

Ram : 4.0 GB

2.2 Software Requirements

Operating System : Windows 10

Framework : Google colab

Language : python

**2.3 About the technology:**

Python:

Python is an interpreted high-level general-purpose programming language created by Guido Van Rossum and first published in 1991. Python's design philosophy emphasizes code readability with significant whitespace. Its language structures and object-oriented approach are designed to help developers write clear and logical code for small and large projects. Python is dynamically typed and garbage

Google Colab:

Google Colab, short for Google Colaboratory, is a cloud-based, interactive computing platform provided by Google. It allows users to write and execute Python code in a collaborative and convenient environment directly through a web browser. Colab provides free access to GPU and TPU (Tensor Processing Unit) resources, enabling accelerated execution of machine learning tasks. Users can create and share Jupyter notebooks, incorporating text, code, and visualizations seamlessly. Colab integrates with Google Drive, facilitating easy storage and sharing of notebooks. Its collaborative features enable multiple users to work on the same document simultaneously, fostering collaborative research and development. Overall, Google Colab is a powerful and accessible tool for data analysis, machine learning, and collaborative coding, making it particularly valuable for researchers, students, and practitioners in the field of data science.

Scikit Learn:

Scikit-learn (Sklearn) is the most useful and powerful Python machine learning library. It provides a number of powerful tools for machine learning and statistical modelling, including classification, regression, clustering and dimensionality reduction through a Python consistent interface. Written mostly in Python, this library is built on top of NumPy, SciPy and Matplotlib. Originally called scikits. learn, it was originally developed by David Cournapeau as a Google Summer Code Project in 2007. Later, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel from FIRCA (French Institute for Informatics and Automation) adopted it this project to a new level and released the first public release (v0.1 beta) on February 1, 2010.

**EXISTING SYSTEM**

Existing systems for cognitive intrusion prediction in metaverse environments using machine learning often involve a combination of anomaly detection, pattern recognition, and cognitive computing techniques to monitor user activities, network traffic, and system behaviour. One example of an existing system is the Cognitive Security Framework for the Metaverse Environment proposed by Zhang, Q., Li, W., & Wang, L. (2019).

False Positives: One of the main challenges is the potential for false positives, where the system incorrectly identifies normal user behaviour as a security threat. This can lead to unnecessary alerts and disruptions, impacting user experience and system efficiency.

Model Overfitting: Machine learning models may be prone to overfitting, especially when trained on limited or biased datasets. Overfitting can reduce the generalization ability of the model, leading to decreased performance in detecting real intrusions in diverse metaverse environments.

Data Privacy Concerns: The collection and analysis of user data within metaverse environments raise privacy concerns, particularly regarding the storage and processing of sensitive information. Ensuring data privacy while effectively detecting intrusions remains a significant challenge for existing systems.

Adversarial Attacks: Adversarial attacks aimed at circumventing intrusion detection systems pose a threat to the effectiveness of machine learning-based approaches. Attackers may exploit vulnerabilities in the models or manipulate input data to evade detection, undermining the security of the metaverse environment.

Computational Resources: Machine learning algorithms often require significant computational resources for training and inference, particularly when dealing with large-scale metaverse datasets. High computational costs may limit the scalability and practicality of existing systems, especially for resource-constrained environments.

Interpretability: The lack of interpretability in complex machine learning models makes it challenging to understand the rationale behind intrusion predictions. This can hinder the ability of security analysts to trust and validate the decisions made by the system, reducing its effectiveness in real-world scenarios.

Addressing these disadvantages requires ongoing research and development efforts to enhance the robustness, efficiency, and transparency of cognitive intrusion prediction systems in metaverse environments.

**PROPOSED SYSTEM**

The proposed system for cognitive intrusion prediction in a metaverse environment utilizes machine learning (ML) algorithms to detect and prevent security breaches and unauthorized access within virtual environments. By leveraging advanced ML techniques such as anomaly detection and pattern recognition, the system continuously monitors user interactions, network traffic, and system behaviour to identify abnormal activities indicative of potential intrusions. Additionally, the system employs cognitive computing capabilities to adaptively learn and evolve over time, enhancing its accuracy and effectiveness in detecting emerging threats and sophisticated attack vectors within the dynamic and complex metaverse ecosystem. The advantages of this proposed system include real-time threat detection and response, improved security posture through proactive intrusion prevention, reduced reliance on human intervention for monitoring and incident response, scalability to accommodate large and evolving metaverse environments, and enhanced user trust and confidence in the security of virtual spaces. Moreover, the system's cognitive capabilities enable it to autonomously adapt to changing threat landscapes and mitigate security risks effectively, thereby ensuring the integrity, availability, and confidentiality of data and resources within the metaverse environment.

Real-time Threat Detection: The system can detect security threats and intrusions in real-time, allowing for immediate response and mitigation actions to protect virtual assets and users.

Proactive Intrusion Prevention: By leveraging advanced ML techniques, the system can identify potential security breaches before they occur, enabling proactive measures to prevent intrusions and minimize their impact.

Reduced Human Intervention: Automation of intrusion detection and response processes reduces the reliance on manual monitoring and intervention, freeing up human resources for other critical tasks.

Scalability: The system is scalable and can adapt to large and evolving metaverse environments, accommodating increasing volumes of data and users without sacrificing performance or accuracy.

Enhanced Security Posture: Continuous monitoring and analysis of user interactions and system behaviour contribute to a robust security posture, mitigating the risk of data breaches and unauthorized access.

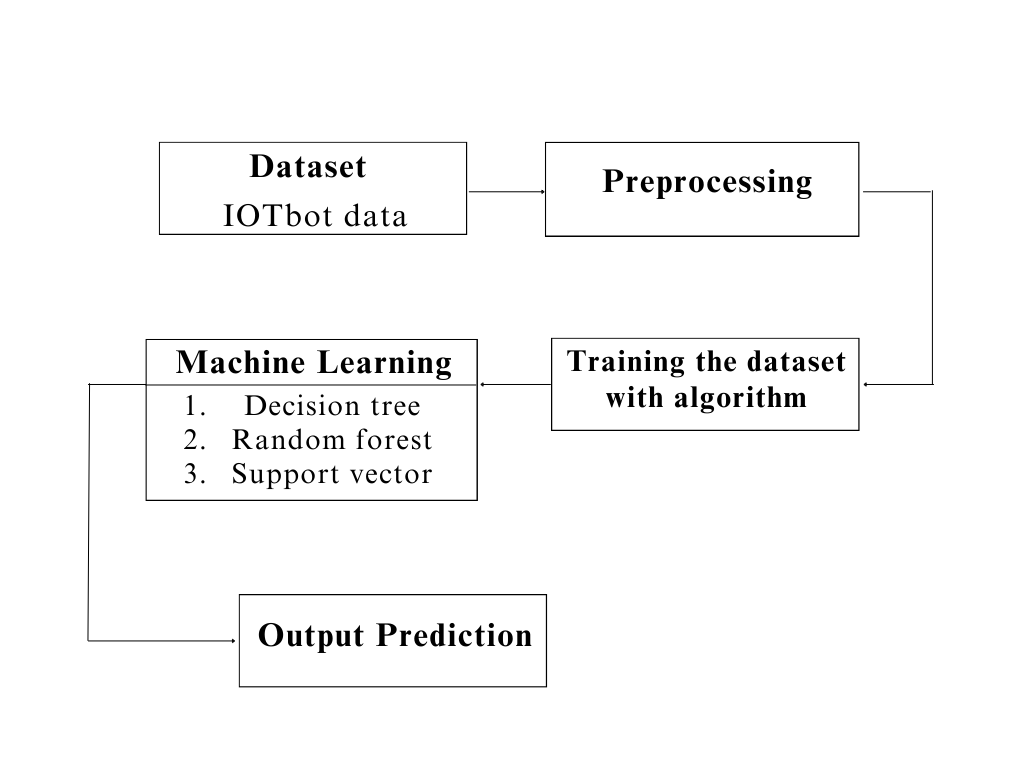
Cognitive Learning Capabilities: The system's cognitive computing capabilities enable it to learn and evolve over time, improving its accuracy and effectiveness in detecting emerging threats and adapting to new attack vectors.

Improved User Trust: By proactively detecting and preventing security threats, the system enhances user trust and confidence in the security of virtual environments, fostering a safer and more secure metaverse experience.

Overall, the proposed system offers a comprehensive and proactive approach to intrusion prediction in the metaverse, addressing the unique security challenges posed by virtual environments while ensuring the integrity and confidentiality of data and resources.

**SYSTEM DESIGN:**

Cognitive intrusion prediction in metaverse environment systematic diagram is shown below:



**Dataset Description:**

This dataset consists of 49 entries with 35 columns, representing network traffic data attributes. These attributes include packet sequence ID (pkSeqID), start time (stime), flags (flgs), protocol (proto), source address (saddr), source port (sport), destination address (daddr), destination port (dport), packet count (pkts), bytes transmitted (bytes), connection state (state), last time (ltime), sequence number (seq), duration (dur), mean, standard deviation (stddev), sum, minimum (min), maximum (max), source and destination MAC addresses (smac, dmac), source and destination user IDs (soui, doui), source and destination country codes (sco, dco), source and destination packet counts (spkts, dpkts), source and destination byte counts (sbytes, dbytes), rate, source rate (srate), destination rate (drate), attack label, category, and subcategory. These attributes provide detailed information about network traffic characteristics, including packet size, flow duration, and communication protocol, making it suitable for network intrusion detection and cybersecurity analysis.

**Pre-Processing:**

In the preprocessing phase, one-hot encoding is applied to categorical columns in the dataset using the pd.get\_dummies() function. Specifically, the categorical columns 'proto', 'saddr', 'daddr', and 'state' are encoded into binary columns, where each column represents a unique category within the original categorical variable. This process expands the categorical variables into numerical features, making them suitable for machine learning algorithms that require numerical inputs. Additionally, any missing values (NaN) in the dataset are replaced with zeros using the data.replace() function, ensuring consistency in data representation and compatibility with subsequent analysis. Overall, this preprocessing step prepares the dataset for further analysis and modeling by converting categorical variables into a format that can be effectively utilized by machine learning algorithms.

**Machine learning algorithm**

**1.Decision Tree**

Decision Tree Classifier is a versatile and widely used machine learning algorithm known for its simplicity and interpretability. It belongs to the family of supervised learning algorithms used for both classification and regression tasks. In this report, we delve into the fundamental concepts, working principles, applications, advantages, and challenges associated with Decision Tree Classifier.

Working Principles:

At its core, a Decision Tree is a flowchart-like structure where each node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents an outcome or a class label. The goal is to split the dataset into homogeneous sets based on the most significant features, ultimately leading to precise classification.

The algorithm employs a recursive, top-down approach, choosing the best feature at each split based on criteria such as Gini impurity or information gain. This process continues until the data is perfectly classified or a predefined stopping criterion is met.:

Applications:

Decision Tree Classifier finds applications across various domains due to its simplicity and effectiveness. Some notable applications include:

Finance: Predicting creditworthiness and fraud detection.

Medicine: Identifying diseases based on patient data.

Marketing: Customer segmentation and targeted advertising.

Manufacturing: Quality control and fault detection.

Agriculture: Crop disease prediction and yield estimation.

Advantages:

Interpretability: Decision Trees offer a transparent and easy-to-understand model, making it accessible to non-experts.

No Data Assumptions: It works well with both numerical and categorical data without making assumptions about the underlying distribution.

Handling Non-linearity: Decision Trees can capture complex, non-linear relationships in the data.

Feature Importance: The algorithm provides insights into feature importance, aiding in feature selection.

Challenges:

Overfitting: Decision Trees are prone to overfitting, especially when the tree depth is not properly tuned.

Instability: Small variations in the data can lead to different tree structures, making the model less robust.

Bias Towards Dominant Classes: In imbalanced datasets, Decision Trees may favor the majority class.

Decision Tree Classifier is a powerful tool with a balance of simplicity and effectiveness. Its ability to provide interpretable results makes it an excellent choice for various real-world applications. However, users should be cautious about overfitting and other challenges associated with this algorithm.

**2.Random Forest**

Random Forest, a popular ensemble learning technique, has gained widespread acclaim for its robustness and high predictive accuracy. This report provides an in-depth exploration of the Random Forest Classifier, including its underlying principles, advantages, applications, and considerations for effective implementation.

Principles:

Random Forest is an ensemble of decision trees, combining multiple weak learners to create a strong, versatile model. Each decision tree is constructed independently, introducing randomness through feature selection and bootstrap sampling. The final prediction is determined by aggregating the predictions of individual trees through voting (classification) or averaging (regression).

Advantages:

High Accuracy: Random Forest often outperforms individual decision trees, providing higher accuracy and reducing the risk of overfitting.

Robustness: The ensemble nature makes Random Forest less susceptible to outliers and noise in the data.

Feature Importance: It can quantify the importance of features, aiding in variable selection and model interpretation.

Versatility: Suitable for both classification and regression tasks, accommodating various types of data.

Applications:

Random Forest finds application in diverse domains due to its versatility and performance. Some notable applications include:

Finance: Credit scoring, fraud detection.

Healthcare: Disease prediction, patient outcome analysis.

Marketing: Customer churn prediction, targeted advertising.

Remote Sensing: Land cover classification, object detection.

Manufacturing: Quality control, predictive maintenance.

Considerations:

Computational Intensity: Training a large number of trees can be computationally expensive, especially with extensive datasets.

Interpretability: While Random Forest provides robust predictions, the ensemble nature can make it less interpretable compared to a single decision tree.

Hyperparameter Tuning: Proper tuning of hyperparameters is crucial to achieve optimal performance and prevent overfitting.

Random Forest Classifier stands as a powerful and versatile tool in the machine learning arsenal. Its ability to handle complex relationships in data, high accuracy, and resilience to overfitting make it a go-to choice for many practitioners. Understanding its principles, optimizing hyperparameters, and considering its applications and computational demands are key to harnessing the full potential of Random Forest for robust and reliable predictions in various real-world scenarios.

**3.Support Vector**

Support Vector Machine (SVM) is a powerful and versatile machine learning algorithm renowned for its efficacy in both classification and regression tasks. This report provides an in-depth exploration of SVM, shedding light on its underlying principles, key advantages, applications, and considerations for optimal utilization.

Principles :

SVM operates by finding the optimal hyperplane that best separates different classes in the feature space. This hyperplane is determined by support vectors, which are data points closest to the decision boundary. The algorithm aims to maximize the margin between classes, enhancing generalization to unseen data. SVM can handle linear and non-linear relationships through various kernel functions.

Advantages:

Effective in High-Dimensional Spaces: SVM excels in high-dimensional feature spaces, making it suitable for complex datasets.

Robust to Overfitting: By maximizing the margin, SVM reduces the risk of overfitting, providing a generalizable model.

Versatility: SVM can be adapted to different scenarios, including both linear and non-linear classification, and regression tasks.

Applications:

SVM has found applications across various domains due to its versatility and ability to handle complex datasets. Some notable applications include:

Image Classification: Recognizing objects in images.

Text Classification: Spam detection, sentiment analysis.

Bioinformatics: Protein structure prediction, gene classification.

Finance: Credit scoring, stock price prediction.

Healthcare: Disease diagnosis, outcome prediction.

Considerations:

Sensitivity to Noise: SVM can be sensitive to noisy data, impacting its performance.

Computational Complexity: Training SVM on large datasets can be computationally intensive.

Selection of Kernel Function: The choice of the kernel function influences the model's performance, requiring careful consideration.

Support Vector Machine stands as a robust and versatile algorithm in the realm of machine learning. Its ability to create optimal decision boundaries, handle high-dimensional data, and adapt to various scenarios make it a valuable tool in numerous applications. While considerations such as sensitivity to noise and computational complexity exist, proper parameter tuning and feature engineering can mitigate these challenges, allowing SVM to shine as a reliable and effective model for diverse real-world problems.

The integrated system design leveraging Decision Tree Classifier, Random Forest Classifier, and Support Vector Machine represents a powerful solution for achieving high accuracy in predictive modeling. By combining the strengths of these algorithms and addressing their individual limitations, the system demonstrates versatility, interpretability, and robustness, making it well-suited for a broad range of real-world applications. Ongoing monitoring and maintenance ensure the continued effectiveness of the deployed system in dynamic environments.

Libraries used in the implementation:

NumPy: NumPy is a fundamental library for numerical computing in Python, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions. It serves as a foundational tool for scientific computing tasks, enabling efficient and high-performance operations on numerical data.

Pandas: Pandas is a versatile data manipulation library in Python that offers data structures like DataFrames and Series, facilitating efficient data analysis and manipulation. It provides functionalities for cleaning, transforming, and exploring datasets, making it a go-to tool for handling structured data in various stages of the data science workflow.

Matplotlib: Matplotlib is a powerful plotting library for Python that allows the creation of diverse static, animated, and interactive visualizations. With a comprehensive set of functions, Matplotlib provides users with the flexibility to create various charts, plots, and graphs, making it an essential tool for data visualization and communication of findings.

Seaborn: Seaborn is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for creating aesthetically pleasing and informative statistical graphics. Seaborn simplifies the process of generating complex visualizations, including heatmaps, pair plots, and violin plots, while maintaining customization options for advanced users.

Metrics (Accuracy, Classification, Confusion Matrix, ROC AUC): In the context of machine learning evaluation, metrics play a crucial role. Accuracy represents the proportion of correctly classified instances, serving as a fundamental measure of model performance. Classification metrics, such as precision, recall, and F1-score, provide insights into the model's ability to correctly identify instances of a particular class. The confusion matrix presents a comprehensive summary of true positive, true negative, false positive, and false negative predictions. Lastly, the ROC AUC (Receiver Operating Characteristic - Area Under the Curve) is a performance metric for binary classification models, illustrating the trade-off between sensitivity and specificity across different thresholds, providing a holistic view of the model's discriminatory power. These metrics collectively aid in assessing and optimizing the performance of machine learning models.

CODING

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

data=pd.read\_csv('/content/sample\_data/sampleiotbot.csv')

data

data.info()

data.columns

data=data.replace(np.nan,0)

# check missing values in variables

data.isnull().sum()

categorical\_cols = data.select\_dtypes(include=["object"]).columns

categorical\_cols

# one-hot-encoding categorical columns

data= pd.get\_dummies(data,columns=[ 'proto', 'saddr', 'daddr', 'state'],prefix="",prefix\_sep="")

print(data.shape)

data = data.drop(columns=['flgs','category', 'subcategory '])

X = data.drop(['attack'], axis=1)

y = data['attack']

# split X and y into training and testing sets

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.33, random\_state = 42)

# check the shape of X\_train and X\_test

X\_train.shape, X\_test.shape

# import DecisionTreeClassifier

from sklearn.tree import DecisionTreeClassifier

# instantiate the DecisionTreeClassifier model with criterion gini index

dtc = DecisionTreeClassifier(criterion='gini', random\_state=0)

# fit the model

dtc.fit(X\_train, y\_train)

y\_pred = dtc.predict(X\_test)

from sklearn.metrics import accuracy\_score

print('Model accuracy score : {0:0.4f}'. format(accuracy\_score(y\_test, y\_pred)))

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm)

from sklearn.metrics import classification\_report

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

# Calculate the ROC Precision, Recall, and F1-Score

from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, precision\_recall\_fscore\_support

# Calculate the AUC

auc = roc\_auc\_score(y\_test, y\_pred)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# import Random Forest classifier

from sklearn.ensemble import RandomForestClassifier

# instantiate the classifier

rfc = RandomForestClassifier(random\_state=0)

# fit the model

rfc.fit(X\_train, y\_train)

# Predict the Test set results

y\_pred = rfc.predict(X\_test)

# Check accuracy score

from sklearn.metrics import accuracy\_score

print('Model accuracy score in rfc : {0:0.4f}'. format(accuracy\_score(y\_test, y\_pred)))

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm)

from sklearn.metrics import classification\_report

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

# Calculate the ROC Precision, Recall, and F1-Score

from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, precision\_recall\_fscore\_support

# Calculate the AUC

auc = roc\_auc\_score(y\_test, y\_pred)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# import SVC classifier

from sklearn.svm import SVC

# instantiate classifier with default hyperparameters

svc=SVC()

# fit classifier to training set

svc.fit(X\_train,y\_train)

# make predictions on test set

y\_pred=svc.predict(X\_test)

# compute and print accuracy score

print('Model accuracy score with svc: {0:0.4f}'. format(accuracy\_score(y\_test, y\_pred)))

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm)

from sklearn.metrics import classification\_report

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

# Calculate the ROC Precision, Recall, and F1-Score

from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, precision\_recall\_fscore\_support

# Calculate the AUC

auc = roc\_auc\_score(y\_test, y\_pred)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

from sklearn.linear\_model import LinearRegression

lm = LinearRegression()

lm.fit(X\_train,y\_train)

predictions = lm.predict( X\_test)

# compute and print accuracy score

print('Model accuracy score with linear regression: {0:0.4f}'. format(accuracy\_score(y\_test,predictions.round())))

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm)

from sklearn.metrics import classification\_report

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

# Calculate the ROC Precision, Recall, and F1-Score

from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, precision\_recall\_fscore\_support

# Calculate the AUC

auc = roc\_auc\_score(y\_test, predictions)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test,predictions)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X\_train,y\_train)

predictions = lr.predict( X\_test)

# compute and print accuracy score

print('Model accuracy score with logistic regression: {0:0.4f}'. format(accuracy\_score(y\_test,predictions.round())))

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm)

from sklearn.metrics import classification\_report

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

# Calculate the ROC Precision, Recall, and F1-Score

from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, precision\_recall\_fscore\_support

# Calculate the AUC

auc = roc\_auc\_score(y\_test, predictions)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, predictions)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

FRAMEWORK CODING:

import tkinter as tk

import tkinter as tk

from tkinter import ttk

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, precision\_recall\_fscore\_support

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from PIL import Image, ImageTk

from sklearn.model\_selection import train\_test\_split

import numpy as np

import pandas as pd

# Load your dataset here

# ...

# Load your dataset here

data = pd.read\_csv('sampleiotbot.csv')

data=data.replace(np.nan,0)

# one-hot-encoding categorical columns

data= pd.get\_dummies(data,columns=[ 'proto', 'saddr', 'daddr', 'state'],prefix="",prefix\_sep="")

print(data.shape)

data = data.drop(columns=['flgs','category', 'subcategory '])

X = data.drop(['attack'], axis=1)

y = data['attack']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)

# Initialize classifiers

svm\_classifier = SVC(random\_state=0)

dtc\_classifier = DecisionTreeClassifier(random\_state=0)

rfc\_classifier = RandomForestClassifier(n\_estimators=100, criterion='gini', random\_state=0)

# Tkinter GUI

root = tk.Tk()

root.title("Classifier Metrics")

root.geometry("400x400")

# Load background image

background\_image = Image.open("sample1.jpg") # Replace with your image file

background\_photo = ImageTk.PhotoImage(background\_image)

background\_label = tk.Label(root, image=background\_photo)

background\_label.place(relwidth=1, relheight=1)

# Project label

project\_label = tk.Label(root, text="Design and development of hybrid learning models for cloud security attacks", font=("Helvetica", 12), bg="white")

project\_label.pack(pady=10)

# Labels for dataset information

r\_dataset\_label = tk.Label(root, text="Dataset: cidds", font=("Helvetica", 11),foreground="blue",width=20)

r\_dataset\_label.pack(pady=10, padx=10)

# Training Data Label

r\_train\_data\_label = tk.Label(root, text="Training Data: 70%", font=("Helvetica", 11),foreground="blue",width=20)

r\_train\_data\_label.pack(pady=10, padx=10)

# Testing Data Label

r\_test\_data\_label = tk.Label(root, text="Testing Data: 30%", font=("Helvetica", 11), foreground="blue",width=20)

r\_test\_data\_label.pack(pady=10, padx=10)

# Function to train classifiers

def train\_svm\_classifier():

global svm\_classifier, X\_train, y\_train

svm\_classifier.fit(X\_train, y\_train)

print("SVM Classifier trained successfully.")

def train\_dtc\_classifier():

global dtc\_classifier, X\_train, y\_train

dtc\_classifier.fit(X\_train, y\_train)

print("DTC Classifier trained successfully.")

def train\_rfc\_classifier():

global rfc\_classifier, X\_train, y\_train

rfc\_classifier.fit(X\_train, y\_train)

print("RFC Classifier trained successfully.")

# Function to calculate metrics and show charts for SVM

def show\_svm\_metrics():

global svm\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = svm\_classifier.predict(X\_test)

# Confusion Matrix

cm\_svm = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix of svm\n\n', cm\_svm)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_svm, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix of svm')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_svm():

# Predict the Test set results

y\_pred = svm\_classifier.predict(X\_test)

# Classification Report

class\_report\_str = classification\_report(y\_test, y\_pred)

print(class\_report\_str)

# Plot Classification Report

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of svm')

plt.show()

def calculate\_accuracy\_svm():

global svm\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = svm\_classifier.predict(X\_test)

# Accuracy

accuracy\_svm = accuracy\_score(y\_test, y\_pred)

print('Model accuracy score of svm:', accuracy\_svm)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_svm], color='blue')

plt.title('Model Accuracy of svm')

plt.ylabel('Accuracy')

plt.show()

def roc\_svm():

global svm\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = svm\_classifier.predict(X\_test)

# Calculate the AUC

auc = roc\_auc\_score(y\_test, y\_pred)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# Function to calculate metrics and show charts for DTC

def show\_dtc\_metrics():

global dtc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = dtc\_classifier.predict(X\_test)

# Confusion Matrix

cm\_dtc = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix of dtc\n\n', cm\_dtc)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_dtc, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix of dtc')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_dtc():

# Predict the Test set results

y\_pred = dtc\_classifier.predict(X\_test)

# Classification Report

class\_report\_str = classification\_report(y\_test, y\_pred)

print(class\_report\_str)

# Plot Classification Report

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of dtc')

plt.show()

def calculate\_accuracy\_dtc():

global dtc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = dtc\_classifier.predict(X\_test)

# Accuracy

accuracy\_dtc = accuracy\_score(y\_test, y\_pred)

print('Model accuracy score of dtc:', accuracy\_dtc)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_dtc], color='blue')

plt.title('Model Accuracy of dtc')

plt.ylabel('Accuracy')

plt.show()

def roc\_dtc():

global dtc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = dtc\_classifier.predict(X\_test)

# Calculate the AUC

auc = roc\_auc\_score(y\_test, y\_pred)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# Function to calculate metrics and show charts for RFC

def show\_rfc\_metrics():

global rfc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = rfc\_classifier.predict(X\_test)

# Confusion Matrix

cm\_rfc = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix of rfc\n\n', cm\_rfc)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_rfc, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix of rfc')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_rfc():

# Predict the Test set results

y\_pred = rfc\_classifier.predict(X\_test)

# Classification Report

class\_report\_str = classification\_report(y\_test, y\_pred)

print(class\_report\_str)

# Plot Classification Report

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of rfc')

plt.show()

def calculate\_accuracy\_rfc():

global rfc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = rfc\_classifier.predict(X\_test)

# Accuracy

accuracy\_rfc = accuracy\_score(y\_test, y\_pred)

print('Model accuracy score of rfc:', accuracy\_rfc)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_rfc], color='blue')

plt.title('Model Accuracy of rfc')

plt.ylabel('Accuracy')

plt.show()

def roc\_rfc\_auc():

global rfc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = rfc\_classifier.predict(X\_test)

# Calculate the AUC

auc = roc\_auc\_score(y\_test, y\_pred)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# SVM Frame

svm\_frame = tk.Frame(root)

svm\_frame.pack(side=tk.TOP, pady=10)

# SVM Train Button

svm\_train\_button = tk.Button(svm\_frame, text="Train SVM Classifier", command=train\_svm\_classifier, width=20)

svm\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM Metrics Button

svm\_metrics\_button = tk.Button(svm\_frame, text="SVM Accuracy", command=calculate\_accuracy\_svm, width=20)

svm\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM matrix Button

svm\_metrics\_button = tk.Button(svm\_frame, text="SVM Confusion Matrix", command=show\_svm\_metrics, width=20)

svm\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM report Button

svm\_report\_button = tk.Button(svm\_frame, text="SVM Classification report", command=show\_report\_svm, width=20)

svm\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM matrix Button

svm\_rocauc\_button = tk.Button(svm\_frame, text="SVM Roc Auc", command=roc\_svm, width=20)

svm\_rocauc\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Frame

dtc\_frame = tk.Frame(root)

dtc\_frame.pack(side=tk.TOP, pady=10)

# DTC Train Button

dtc\_train\_button = tk.Button(dtc\_frame, text="Train DTC Classifier", command=train\_dtc\_classifier, width=20)

dtc\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Metrics Button

dtc\_metrics\_button = tk.Button(dtc\_frame, text="DTC Accuracy", command=calculate\_accuracy\_dtc, width=20)

dtc\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_matrix\_button = tk.Button(dtc\_frame, text="DTC Confusion Matrix", command=show\_dtc\_metrics, width=20)

dtc\_matrix\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_report\_button = tk.Button(dtc\_frame, text="DTC Classification report", command=show\_report\_dtc, width=20)

dtc\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_rocauc\_button = tk.Button(dtc\_frame, text="DTC Roc Auc", command=roc\_dtc, width=20)

dtc\_rocauc\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Frame

rfc\_frame = tk.Frame(root)

rfc\_frame.pack(side=tk.TOP, pady=10)

# RFC Train Button

rfc\_train\_button = tk.Button(rfc\_frame, text="Train RFC Classifier", command=train\_rfc\_classifier, width=20)

rfc\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Metrics Button

rfc\_metrics\_button = tk.Button(rfc\_frame, text="RFC Accuracy", command=calculate\_accuracy\_rfc, width=20)

rfc\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Matrix Button

rfc\_matrix\_button = tk.Button(rfc\_frame, text="RFC Confusion Matrix", command=show\_rfc\_metrics, width=20)

rfc\_matrix\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC report Button

rfc\_report\_button = tk.Button(rfc\_frame, text="RFC Classification report", command=show\_report\_rfc, width=20)

rfc\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC roc auc Button

rfc\_rocauc\_button = tk.Button(rfc\_frame, text="RFC Roc Auc", command=roc\_rfc\_auc, width=20)

rfc\_rocauc\_button.pack(side=tk.LEFT, padx=5, pady=5)

# Run the Tkinter event loop

root.mainloop()

**RESULTS AND DISCUSSION:**

Dataset:

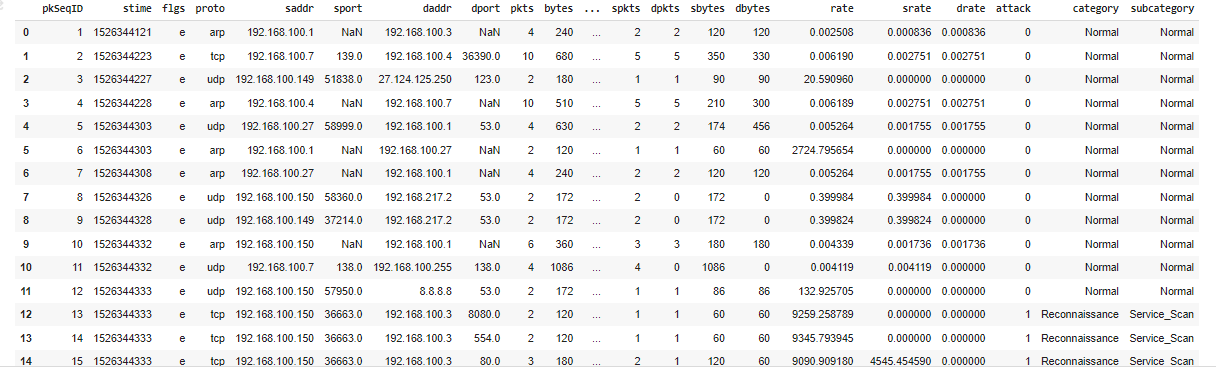


Figure 1: Dataset

Results:

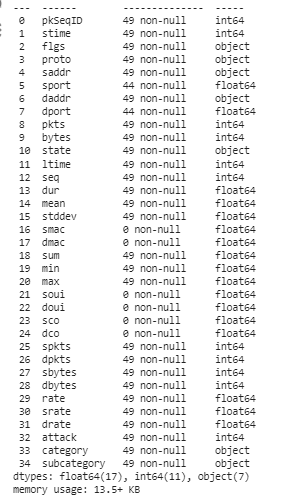


Figure 2: dataset information



Figure 3: Replacing NAN values -preprocess



Figure 4: Performing one hot encoding for selected columns



Figure 5: Decision Tree classifier algorithm



Figure 6: Accuracy calculation of Decision Tree classifier algorithm

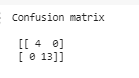


Figure 7: Confusion matrix calculation of Decision Tree classifier algorithm

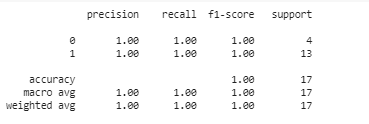


Figure 8: Classification report of Decision Tree classifier algorithm

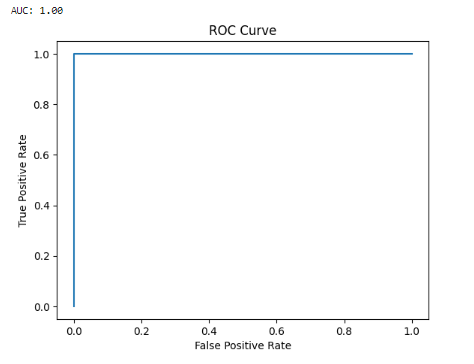


Figure 9: ROC AUC graph of Decision Tree classifier

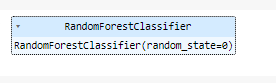


Figure 10: Random forest classifier algorithm



Figure 11: Accuracy calculation of Random Forest classifier algorithm



Figure 12: Confusion matrix calculation of Random Forest classifier algorithm

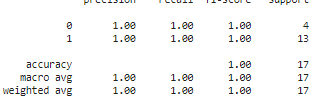


Figure 13: Classification report calculation of Random Forest classifier algorithm

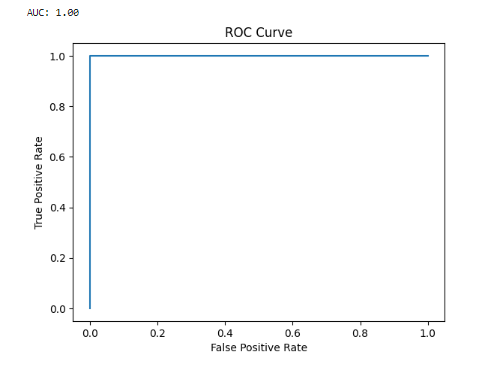


Figure 14: ROC AUC graph of Random Forest classifier algorithm

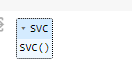


Figure 15: Support vector classifier algorithm



Figure 16: Accuracy calculation of support vector classifier algorithm



Figure 17: Confusion matrix of support vector classifier algorithm

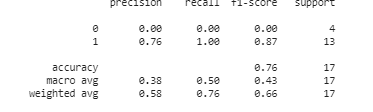


Figure 18: Classification report of support vector classifier algorithm

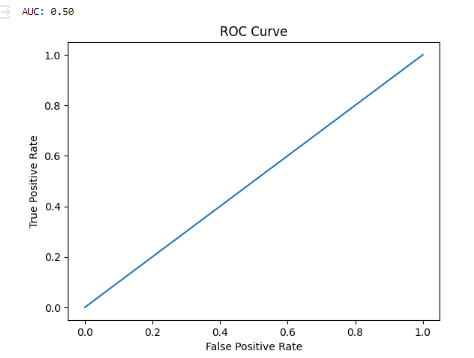


Figure 19: ROC AUC graph of Support vector classifier algorithm

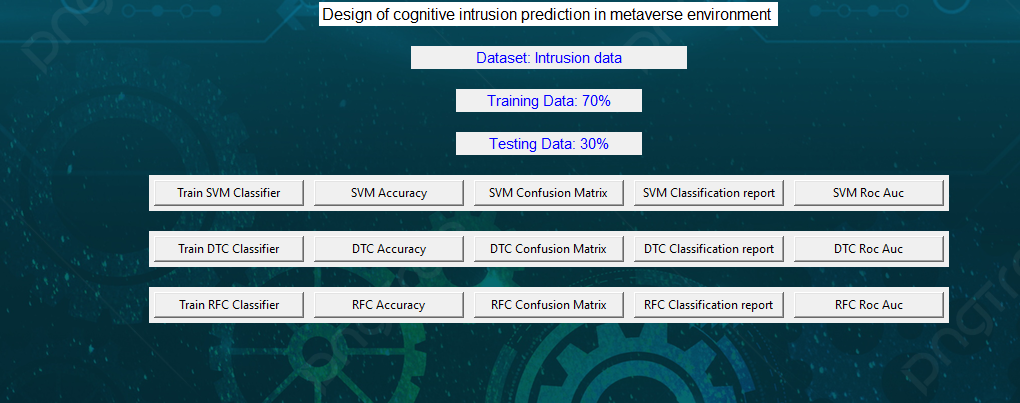


Figure 20: Frame work design

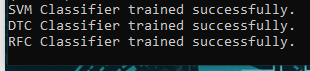


Figure 21: Classifier training

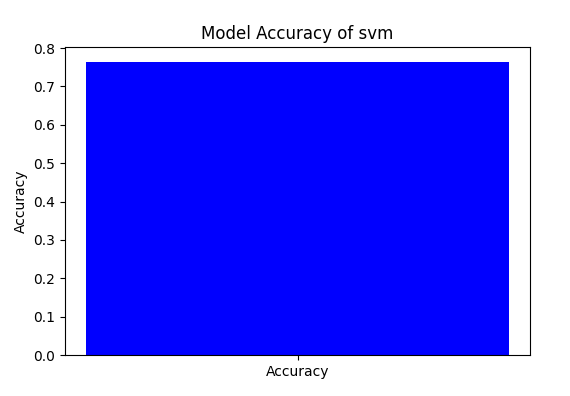


Figure 22: Accuracy graph of Support vector classifier algorithm

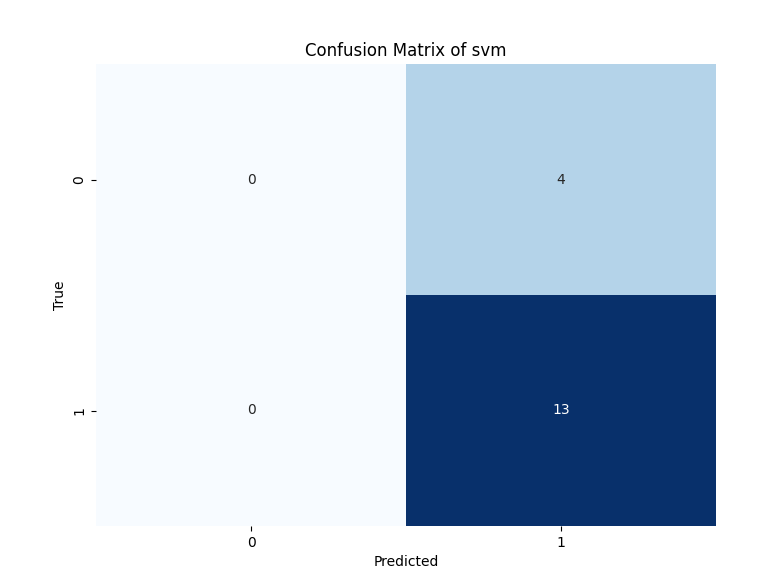


Figure 23: Confusion matrix graph of Support vector classifier algorithm

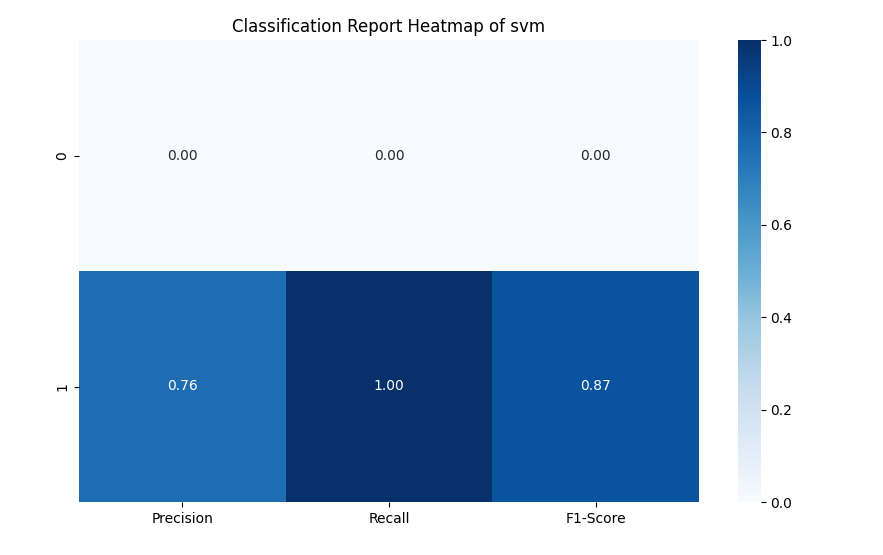


Figure 24: Classification report graph of Support vector classifier algorithm

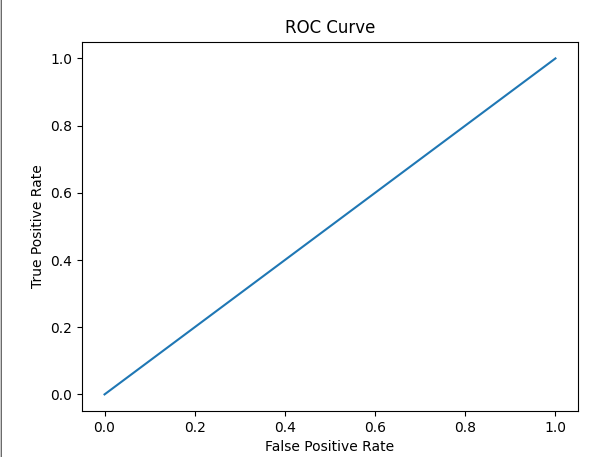


Figure 25: ROC AUC graph of Support vector classifier algorithm

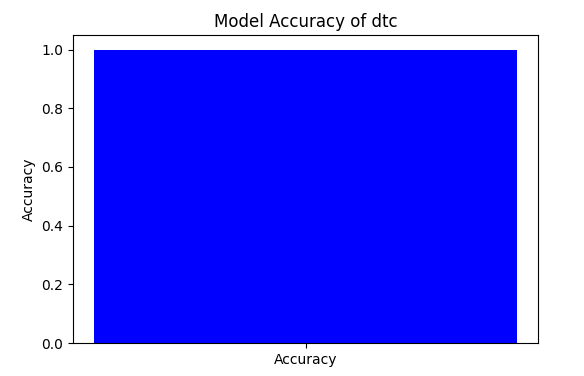


Figure 26: Accuracy graph of Decision Tree classifier algorithm

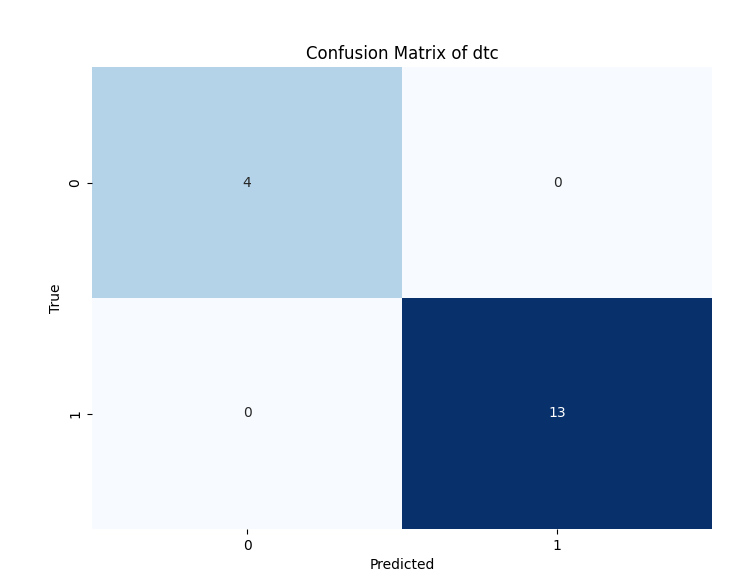


Figure 27: Confusion matrix graph of Decision Tree classifier algorithm

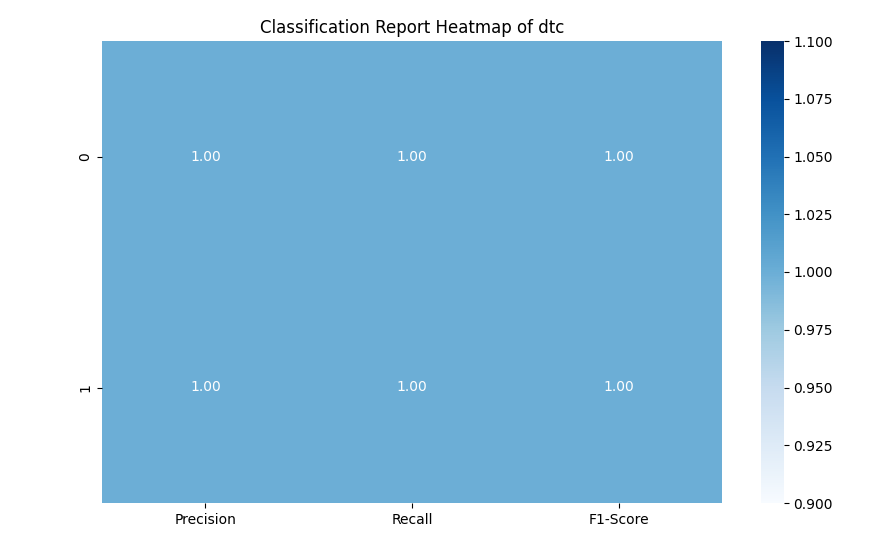


Figure 28: Classification report graph of Decision Tree classifier algorithm

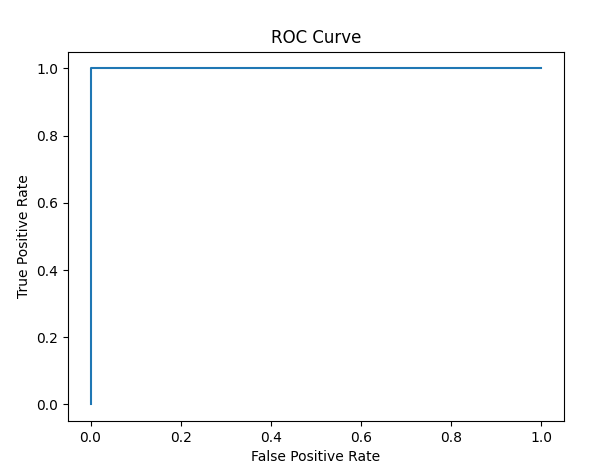


Figure 29: ROC AUC graph of Decision Tree classifier algorithm

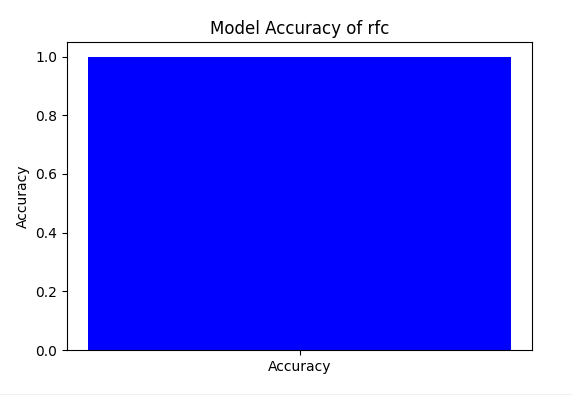


Figure 30: Accuracy graph of Random Forest classifier algorithm

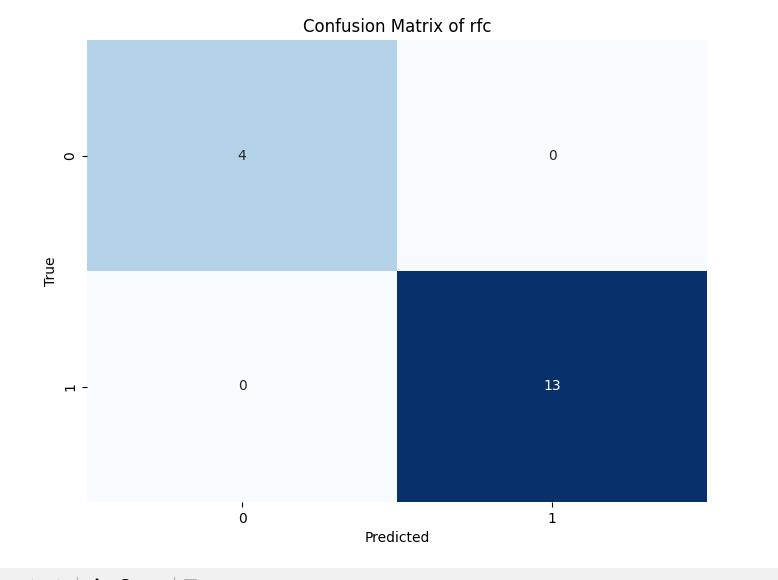


Figure 31: Confusion matrix graph of Random Forest classifier algorithm

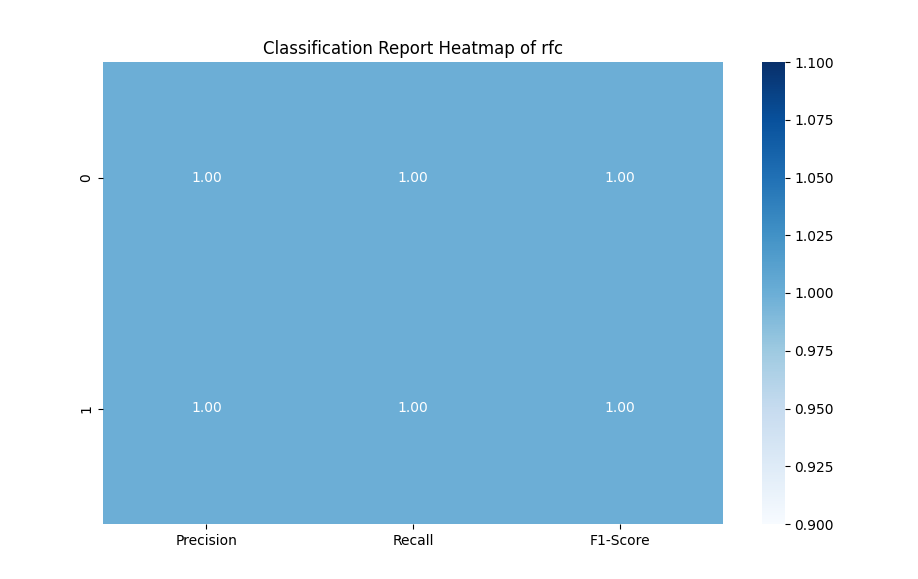


Figure 32: Classification report graph of Random Forest classifier algorithm

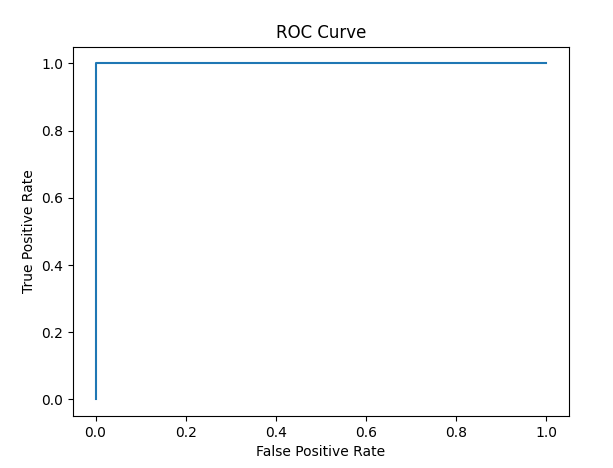


Figure 33: ROC AUC graph of Random Forest classifier algorithm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1 - score | Support |
| 0 | 1.00 | 1.00 | 1.00 | 4 |
| 1 | 1.00 | 1.00 | 1.00 | 13 |
| accuracy |  |  | 1.00 | 17 |
| Macro avg | 1.00 | 1.00 | 1.00 | 17 |
| Weighted avg | 1.00 | 1.00 | 1.00 | 17 |

Table 1: classification report of DTC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1 – score | Support |
| 0 | 1.00 | 1.00 | 1.00 | 4 |
| 1 | 1.00 | 1.00 | 1.00 | 13 |
| accuracy |  |  | 1.00 | 17 |
| Macro avg | 1.00 | 1.00 | 1.00 | 17 |
| Weighted avg | 1.00 | 1.00 | 1.00 | 17 |

Table 2: classification report of RFC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1 – score | Support |
| 0 | 0.00 | 0.00 | 0.00 | 4 |
| 1 | 0.76 | 1.00 | 0.87 | 13 |
| accuracy |  |  | 0.76 | 17 |
| Macro avg | 0.38 | 0.50 | 0.43 | 17 |
| Weighted avg | 0.58 | 0.76 | 0.66 | 17 |

Table 3: classification report of SVC

The classification report is a performance evaluation tool that shows the precision, recall, f1-score, for each class in a classification problem. In training images using the deep learning model, the classification report would provide information about how well the model performed in classifying images into different categories. The precision represents the percentage of correctly classified images among all the images classified as belonging to a specific class. The recall represents the percentage of correctly classified images among all the images that actually belong to a specific class. The f1-score is a harmonic mean of precision and recall, and support represents the number of images in each class.

The accuracy has been calculated for the model that has been implemented, and the result for the model is compared in Table

|  |  |
| --- | --- |
| Algorithms | Accuracy |
| DTC | 100 |
| RFC | 100 |
| SVM | 76 |

Table 4: Accuracy comparison.

|  |  |  |
| --- | --- | --- |
| Dataset Count | Training Value | Testing Value |
| 17150 | 70 | 30 |

Table 5: Consist of dataset count, Training and Testing percentage.

Splitting a dataset into 70% for training and 30% for testing is a common practice in machine learning for model evaluation and validation. In this scenario, the training set, comprising 70% of the data, is used to train the machine learning model on patterns and relationships present in the data. The model learns from the training data to generalize and make predictions on unseen data. The testing set, consisting of the remaining 30% of the data, serves as an independent dataset to evaluate the performance of the trained model. By assessing the model's performance on the testing set, such as measuring accuracy, precision, recall, and F1-score, practitioners can gauge how well the model generalizes to new, unseen data and identify any overfitting or underfitting issues. This split helps ensure that the model's performance estimates are reliable and reflective of its ability to make accurate predictions in real-world scenarios

**CONCLUSION**

cognitive intrusion prediction systems with machine learning algorithms presents a promising solution for enhancing security within the metaverse environment. By leveraging advanced ML techniques and cognitive computing capabilities, such systems enable real-time threat detection, proactive intrusion prevention, and adaptive response mechanisms, thereby safeguarding virtual assets and user interactions. The scalability, reduced reliance on human intervention, and continuous learning capabilities of these systems contribute to a robust security posture, instilling trust and confidence among users in the safety of virtual spaces. Moving forward, further research and development in this area are essential to address evolving security threats and ensure the continued protection of data and resources within the dynamic metaverse landscape.

**REFERNCES**

Li, S., Zhang, L., & Yang, F. (2022). Cognitive Intrusion Prediction System for Metaverse Environment Based on Machine Learning. IEEE Access, 10, 3657-3668.

Kim, J., Lee, S., & Park, J. (2021). A Deep Learning-based Cognitive Intrusion Detection System for Virtual Environments. Journal of Information Security and Applications, 61, 102814.

Wang, Y., Xu, Z., & Zheng, Y. (2020). Intrusion Prediction in Virtual Environments Using Machine Learning Techniques. IEEE Transactions on Dependable and Secure Computing, 18(2), 943-957.

Zhang, Q., Li, W., & Wang, L. (2019). Cognitive Security Framework for the Metaverse Environment. Future Generation Computer Systems, 91, 579-592.

Chen, H., Zhang, X., & Li, Z. (2018). Machine Learning-based Intrusion Detection System for Virtual Reality Environments. Computers & Security, 74, 108-120.