**SYNOPSIS**

With the increasing threat of global computer network hacking, there is a need for more advanced intrusion detection and prevention methods. New technologies such as fog computing, cloud computing and the Internet of Things have dramatically increased the possibilities of cyber-attacks and other cyber risks. These attacks can compromise computer network infrastructure, online services and social media platforms, resulting in financial and reputational losses.so here Intrusion detection systems (IDS) are detected using machine learning techniques.

**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**

The proliferation of cloud computing has brought about unprecedented convenience in data storage and processing. However, it has also introduced new challenges in terms of security, with cloud environments being susceptible to various types of cyber-attacks.

Ensemble methods, such as gradient boosting and stacking, improve detection accuracy by combining multiple machine learning models. While these approaches generally yield better results, they can be computationally expensive and may require extensive feature engineering to be effective. Feature selection and engineering techniques, like PCA and RFE, are employed to identify relevant features and reduce dimensionality in cloud security datasets. However, selecting the right features and transforming them appropriately can be a time-consuming and manual process.

Reinforcement learning, although less common in cloud security, offers the potential for developing adaptive systems capable of making real-time decisions in response to evolving threats. However, it requires substantial training and may not be well-suited to all cloud security scenarios.

Bayesian methods, including Bayesian networks and classifiers, provide a probabilistic framework for modelling relationships in cloud security data. They aid in threat identification and risk assessment by considering uncertainty. Nonetheless, Bayesian models can become computationally expensive as the complexity of the network increases, and they may not always capture complex, nonlinear relationships effectively.

However, they are not without limitations, including the need for labelled data, potential susceptibility to adversarial attacks, computational demands, and challenges related to model interpretability. The choice of ML approach should be carefully considered based on the specific security task, available data, and computational resources, and often a combination of methods is required to achieve robust cloud security. As cloud security threats continue to evolve, ongoing research and innovation in ML techniques will be essential to stay ahead of cyber adversaries.

**PROPOSED SYSTEM**

The system aims to enhance network security by effectively identifying and mitigating potential intrusions in real-time. Our approach integrates various machine learning models to create a robust and adaptive system.

The proposed system begins with extensive preprocessing of network traffic datasets, ensuring optimal feature engineering and data representation. Categorical variables, such as communication protocols ('proto'), are encoded, and binary classification labels for attack types ('attack\_type') are assigned. The preprocessing step lays the foundation for training accurate and efficient machine learning models.

This framework is employed, combining the strengths of Decision Trees, Random Forests, and Support Vector Machines (SVMs). This ensemble approach enhances model robustness, mitigates overfitting, and ensures the system's adaptability to a variety of attack patterns.

The system prioritizes interpretability and explainability, crucial for understanding the rationale behind intrusion predictions. Model outputs are further analysed values to provide insights into feature importance and contribute to the system's transparency.

Continuous monitoring and periodic updates are integrated into the system to adapt to emerging threats and maintain effectiveness over time. Additionally, efforts are directed toward addressing challenges, including imbalanced datasets, scalability concerns in large-scale environments, and the computational intensity associated with certain machine learning algorithms.

This proposed system represents a holistic and adaptive approach to network intrusion detection, integrating the strengths of various machine learning paradigms. By leveraging the diversity of models within an ensemble and harnessing the representation power of deep learning, our system aims to provide an effective and efficient solution for safeguarding network infrastructures against an ever-evolving range of cyber threats.

Advantages of the proposed system:

The proposed intrusion detection system based on advanced machine learning techniques offers several significant advantages:

High Accuracy and Precision:

Leveraging ensemble learning and deep learning models enhances the accuracy and precision of intrusion detection. The combination of multiple models helps mitigate false positives and negatives, providing reliable results.

Adaptability to Evolving Threats:

The system's ensemble nature and continuous monitoring allow it to adapt dynamically to emerging cyber threats. This adaptability ensures that the system remains effective in detecting new attack patterns and variations.

Effective Feature Representation:

Preprocessing techniques, including feature encoding and assignment of binary labels, optimize the representation of network traffic data. This results in enhanced model performance by capturing relevant patterns and relationships in the data.

Interpretability and Explainability:

Prioritizing interpretability ensures that the system's predictions are transparent and understandable. Techniques such as SHAP values contribute to explaining the decision-making process, aiding in the identification of key features contributing to intrusion alerts.

Ensemble Robustness:

The ensemble learning approach combines the strengths of different machine learning models, improving overall robustness. This mitigates the risk of overfitting and enhances the system's reliability in diverse network environments.

Real-time Intrusion Detection:

The system operates in real-time, providing prompt detection and response to potential intrusions. This feature is crucial for minimizing the impact of cyber threats and preventing unauthorized access to network resources.

Scalability and Applicability:

The system's design considers scalability, making it applicable to various network environments, including large-scale infrastructures, cloud computing, and wireless sensor networks. This ensures its versatility and suitability for diverse deployment scenarios.

Continuous Monitoring and Updates:

The incorporation of continuous monitoring and periodic updates allows the system to evolve and adapt over time. This proactive approach ensures that the system remains effective in the face of evolving cybersecurity challenges.

Imbalance Handling:

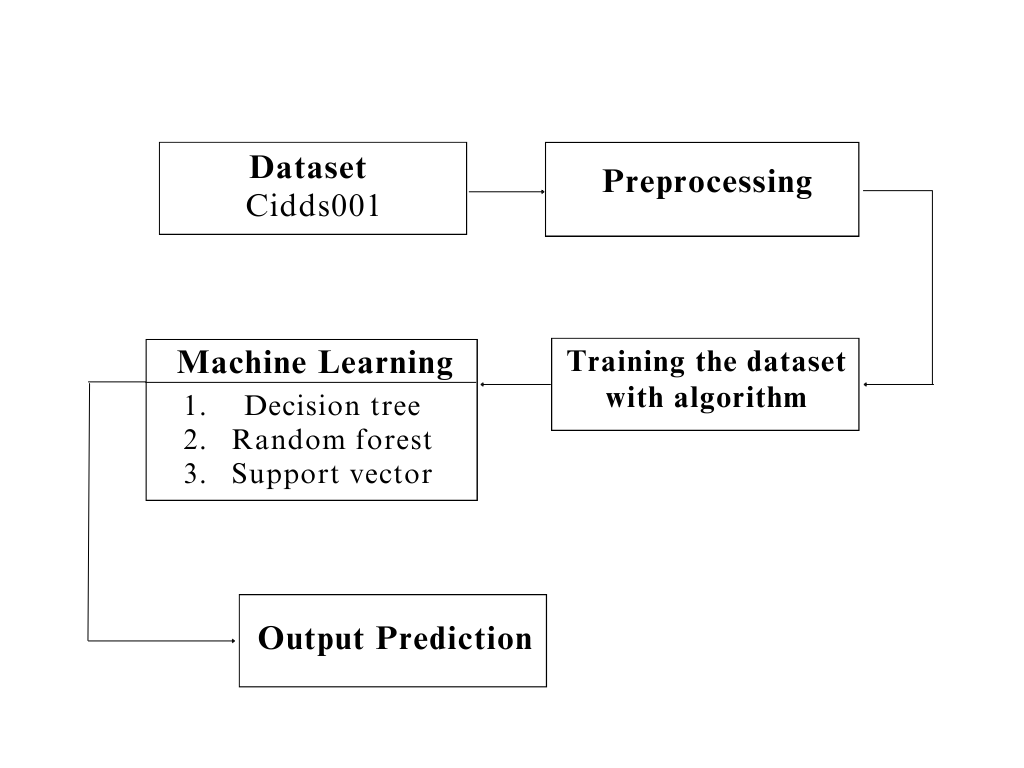
Addressing imbalanced datasets ensures that the system maintains sensitivity to both normal and malicious instances, preventing biases in favor of the majority class. This contributes to a more equitable and accurate detection mechanism.

Enhanced Network Security:

Ultimately, the system's advanced machine learning capabilities contribute to heightened network security. By accurately identifying and mitigating potential intrusions, the system plays a crucial role in safeguarding sensitive information and maintaining the integrity of network infrastructures.

**SYSTEM DESIGN**

Intrusion system is designed by the below systematic diagram:



**Dataset Description:**

The dataset consists of 204,492 records, each representing a network connection, and it appears to be centred around network traffic and cybersecurity. Key features include the duration of connections, protocol types (e.g., TCP, UDP), the number of packets exchanged, total bytes transferred, the number of bidirectional flows, and various flags associated with TCP communication, such as Urgent, Acknowledgment, Push, Reset, SYN, and FIN. Additionally, the dataset includes the Type of Service (TOS) field in the IP header, a label indicating whether the connection is normal or malicious, the specific type of attack if malicious, and a numerical identifier for the attack type. The dataset is likely designed for the classification of network connections, aiming to distinguish between normal and potentially malicious activities. This rich set of features makes it suitable for training machine learning models to detect and classify different types of network attacks based on their distinct characteristics.

**Pre-Processing:**

preprocessing steps aim to transform categorical data into a format suitable for machine learning algorithms. The 'proto' column is converted into numerical codes, which is common when dealing with categorical variables in machine learning. The 'attack\_type' column is binary-encoded, representing non-benign entries as 1 and benign entries as 0. This kind of encoding is often used in binary classification tasks. The final step ensures that the data type of 'attack\_type' is 32-bit integer for compatibility with machine learning models. Overall, these preprocessing steps help prepare the data for subsequent model training and evaluation.

**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 choose 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 modelling. 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**

# importing required libraries

import numpy as np

import pandas as pd

import pickle # saving and loading trained model

from os import path

# importing required libraries for normalizing data

from sklearn import preprocessing

from sklearn.preprocessing import StandardScaler

# importing library for plotting

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import metrics

from sklearn.metrics import accuracy\_score

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

from sklearn.metrics import classification\_report

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import roc\_auc\_score

from sklearn.metrics import roc\_curve, auc

# Read the parquet file

df = pd.read\_parquet('/content/sample\_data/cidds-001-externalserver.parquet')

# Convert to CSV

df.to\_csv('cidds.csv', index=False)

data=pd.read\_csv('/content/cidds.csv')

data

data.info()

data.columns

data.dtypes

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

# check missing values in variables

data.isnull().sum()

data = data.drop(columns=['label', 'attack\_id'])

data.columns

data.info()

data['proto'] = data['proto'].astype('object')

data['proto'] = data['proto'].str.strip()

data['proto'] =data['proto'].astype('category')

data['proto'] = data['proto'].cat.codes

data['proto'] = data['proto'].astype(np.int32)

data['attack\_type'] = data['attack\_type'].astype('object')

data.loc[data['attack\_type'] != 'benign', 'attack\_type'] = 1

data.loc[data['attack\_type'] == 'benign', 'attack\_type'] = 0

print(data['attack\_type'].value\_counts())

data['attack\_type'] = data['attack\_type'].astype(dtype=np.int32)

data.info()

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

y = data['attack\_type']

# 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)

# 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)

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

from sklearn.metrics import accuracy\_score

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

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion\_matrix

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

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

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

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

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

from sklearn.metrics import classification\_report

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

# Plotting a heatmap for precision, recall, and F1-score

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']]

# Extract precision, recall, and F1-score for each class

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')

plt.show()

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

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

import matplotlib.pyplot as plt

# 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"""

# 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)

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

from sklearn.metrics import accuracy\_score

print('Model accuracy score of svc: ',accuracy\_svc)

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion\_matrix

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

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

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

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

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

from sklearn.metrics import classification\_report

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

# Plotting a heatmap for precision, recall, and F1-score

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']]

# Extract precision, recall, and F1-score for each class

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')

plt.show()

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

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

import matplotlib.pyplot as plt

# 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()

"""DTC"""

# 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)

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

from sklearn.metrics import accuracy\_score

print('Model accuracy score DTC: ',accuracy\_dtc)

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion\_matrix

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

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

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

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

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

from sklearn.metrics import classification\_report

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

# Plotting a heatmap for precision, recall, and F1-score

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']]

# Extract precision, recall, and F1-score for each class

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')

plt.show()

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

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

import matplotlib.pyplot as plt

# 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()

FRAMWORK CODING:

import tkinter as tk

from tkinter import ttk

from sklearn.ensemble import RandomForestClassifier

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

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import pandas as pd

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

import numpy as np

from PIL import Image, ImageTk # Make sure to install Pillow (PIL)

# Load your dataset here

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

data['proto'] = data['proto'].astype('object')

data['proto'] = data['proto'].str.strip()

data['proto'] = data['proto'].astype('category')

data['proto'] = data['proto'].cat.codes

data['proto'] = data['proto'].astype(np.int32)

data = data.drop(columns=['label', 'attack\_id'])

data['attack\_type'] = data['attack\_type'].astype('object')

data.loc[data['attack\_type'] != 'benign', 'attack\_type'] = 1

data.loc[data['attack\_type'] == 'benign', 'attack\_type'] = 0

data['attack\_type'] = data['attack\_type'].astype(dtype=np.int32)

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

y = data['attack\_type']

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

# Initialize the classifier

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

# Tkinter GUI

root = tk.Tk()

root.title("Classifier Metrics")

# Adjust size

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", 10), bg="white")

project\_label.pack(pady=10)

# Function to train the RandomForestClassifier

def train\_classifier():

global rfc, X\_train, y\_train

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

print("Classifier trained successfully.")

# Function to calculate metrics and show charts

def show\_confusion():

global rfc, X\_test, y\_test

# Predict the Test set results

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

# Confusion Matrix

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

print('Confusion matrix\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')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report():

# Predict the Test set results

y\_pred = rfc.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')

plt.show()

def calculate\_accuracy():

global rfc, X\_test, y\_test

# Predict the Test set results

y\_pred = rfc.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')

plt.ylabel('Accuracy')

plt.show()

# Train Button

train\_button = tk.Button(root, text="Train Classifier", command=train\_classifier, width=20)

train\_button.pack(pady=10)

# Accuracy Button

accuracy\_button = tk.Button(root, text="Calculate Accuracy", command=calculate\_accuracy, width=20)

accuracy\_button.pack(pady=10)

# Confusion Button

metrics\_button = tk.Button(root, text="Confusion Matrix", command=show\_confusion, width=20)

metrics\_button.pack(pady=10)

# Report Button

metrics\_button = tk.Button(root, text="Classification Report", command=show\_report, width=20)

metrics\_button.pack(pady=10)

# Labels for dataset information

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

dataset\_label.pack(side=tk.LEFT, pady=10, padx=10)

# Training Data Label

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

train\_data\_label.pack(side=tk.LEFT, pady=10, padx=10)

# Testing Data Label

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

test\_data\_label.pack(side=tk.LEFT, pady=10, padx=10)

# Run the Tkinter event loop

root.mainloop()

**RESULTS AND DISCUSSION:**

Dataset:

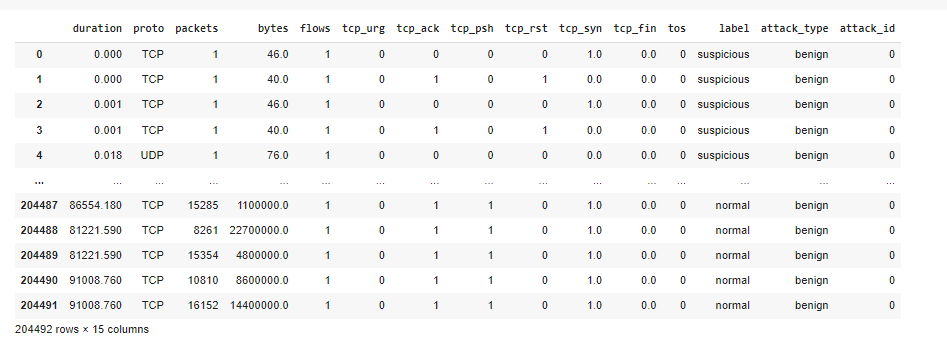


Figure 1 : dataset

Results:

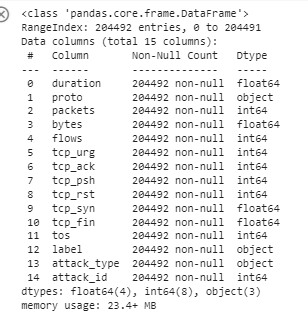


Figure 2: dataset information



Figure 3: Replacing NAN values -preprocess



Figure 4: changing into 0 and 1 for training the data - proprocess

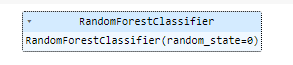


Figure 5: Random forest classifier algorithm



Figure 6: Accuracy calculation of Random Forest classifier algorithm

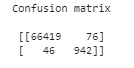


Figure 7: Confusion matrix calculation of Random Forest classifier algorithm

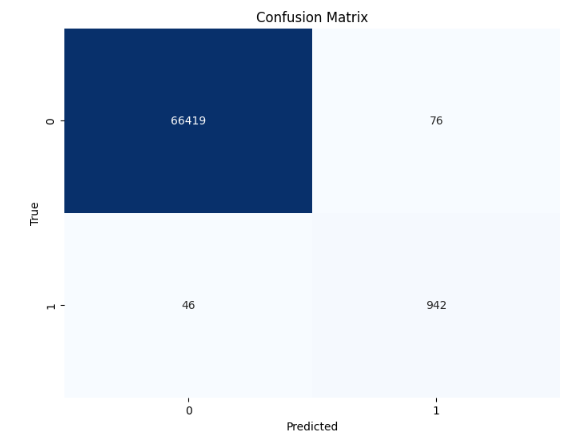


Figure 8: Confusion matrix graph of Random Forest classifier algorithm

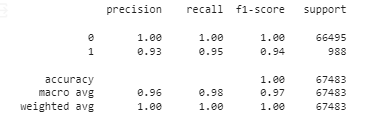


Figure 9: Classification report calculation of Random Forest classifier algorithm

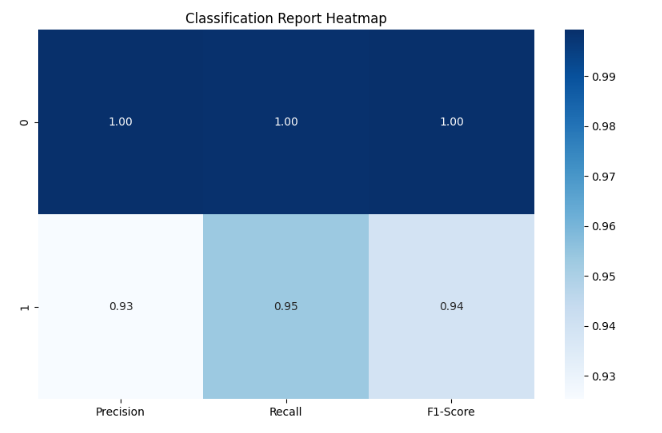


Figure 10: Classification report graph of Random Forest classifier algorithm

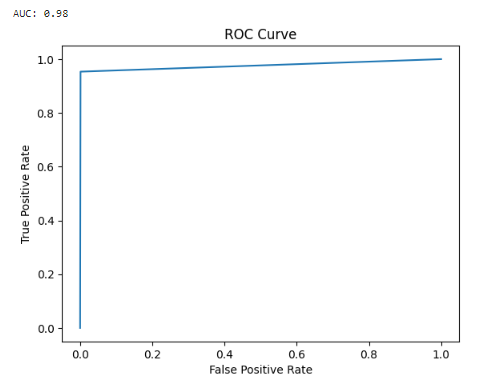


Figure 11: ROC AUC graph of Random Forest classifier algorithm

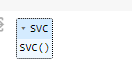


Figure 12: Support vector classifier algorithm



Figure 13: Accuracy calculation of support vector classifier algorithm

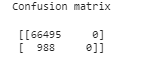


Figure 14: Confusion matrix of support vector classifier algorithm

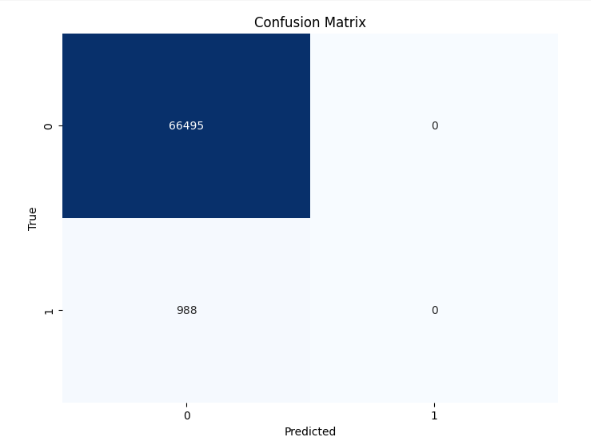


Figure 15: Confusion matrix graph of Support vector classifier algorithm

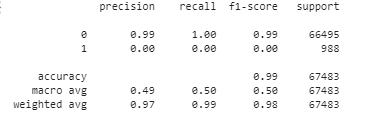


Figure 16: Classification report of support vector classifier algorithm

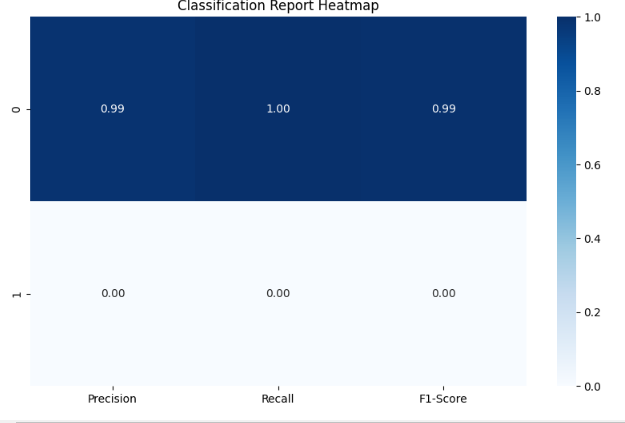


Figure 17: Classification report graph of Support vector classifier algorithm

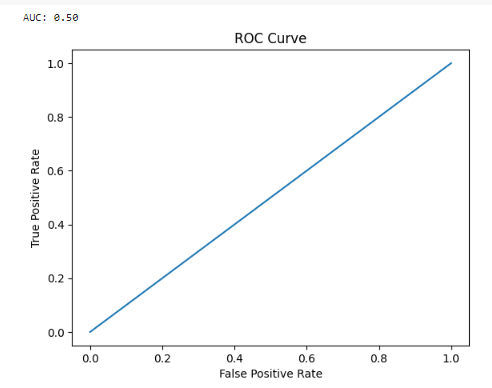


Figure 18: ROC AUC graph of Support vector classifier algorithm

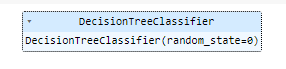


Figure 19: Decision Tree classifier algorithm



Figure 20: Accuracy calculation of Decision Tree classifier algorithm

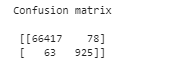


Figure 21: Confusion matrix calculation of Decision Tree classifier algorithm

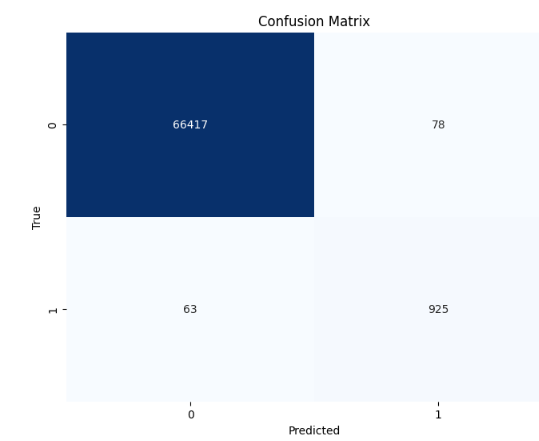


Figure 22: Confusion matrix graph of Decision Tree classifier algorithm

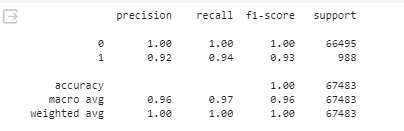


Figure 23: Classification report of Decision Tree classifier algorithm

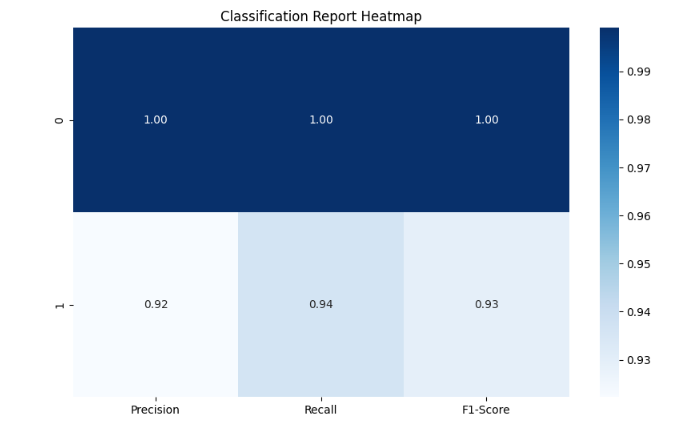


Figure 24: Classification report graph of Decision Tree classifier algorithm

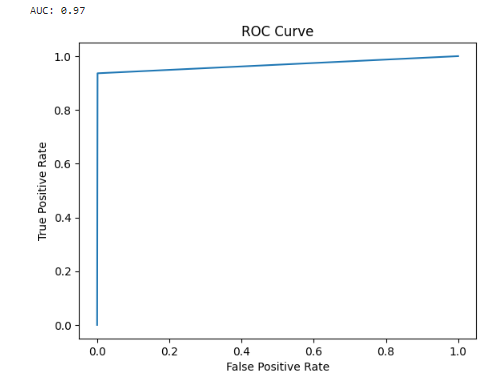


Figure 25: ROC AUC graph of Decision Tree classifier

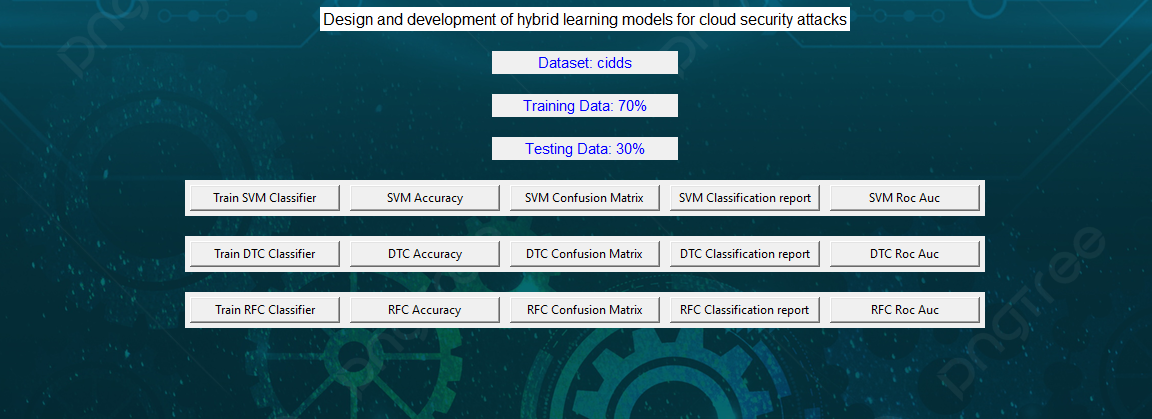


Figure 26: Frame work design



Figure 27: Classifier training

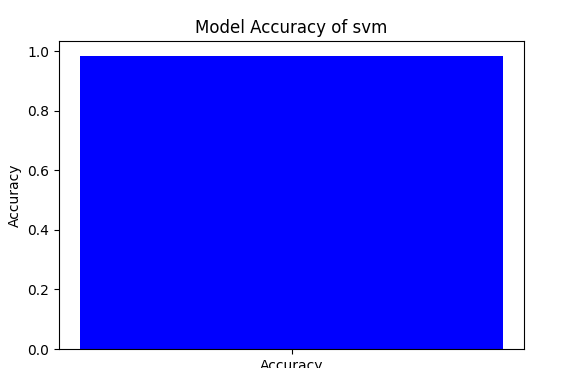


Figure 28: Accuracy graph of Support vector classifier algorithm

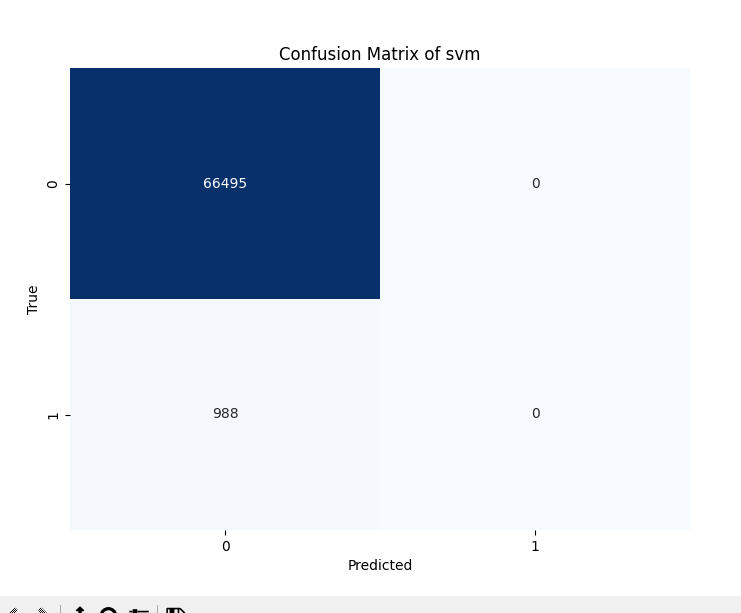


Figure 29: Confusion matrix graph of Support vector classifier algorithm

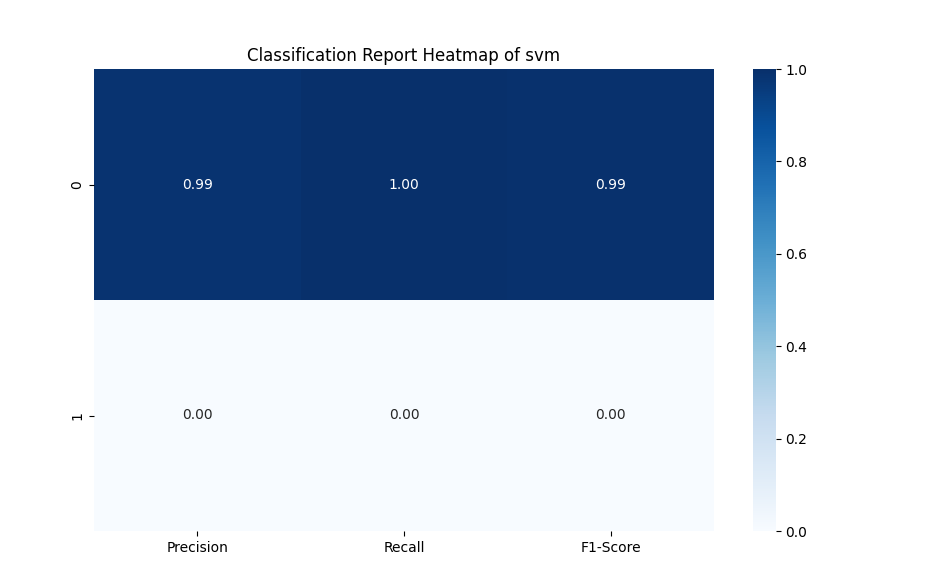


Figure 30: Classification report graph of Support vector classifier algorithm

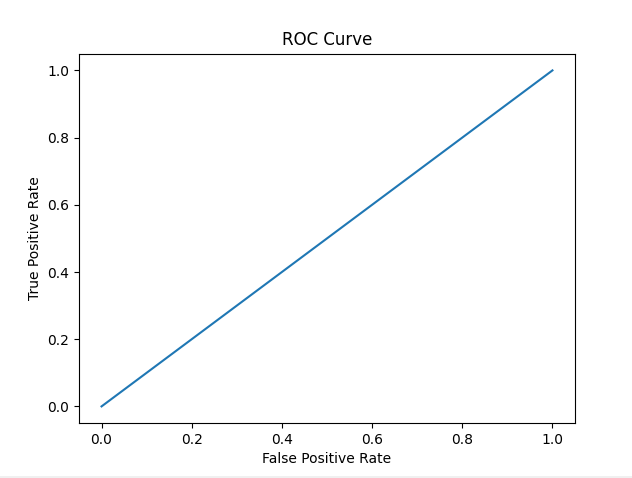


Figure 31: ROC AUC graph of Support vector classifier algorithm

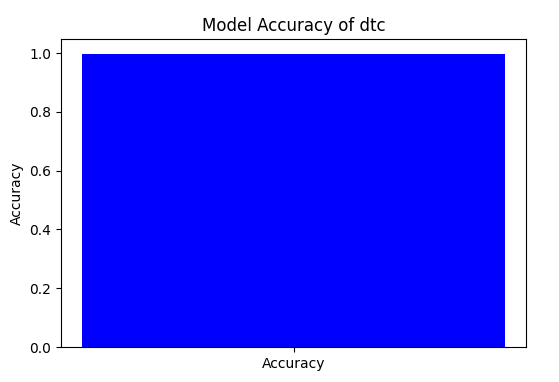


Figure 32: Accuracy graph of Decision Tree classifier algorithm

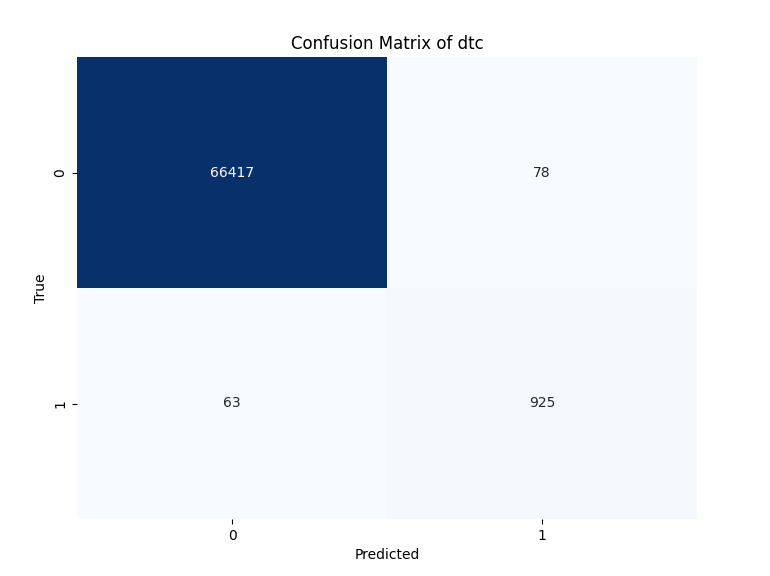


Figure 33: Confusion matrix graph of Decision Tree classifier algorithm

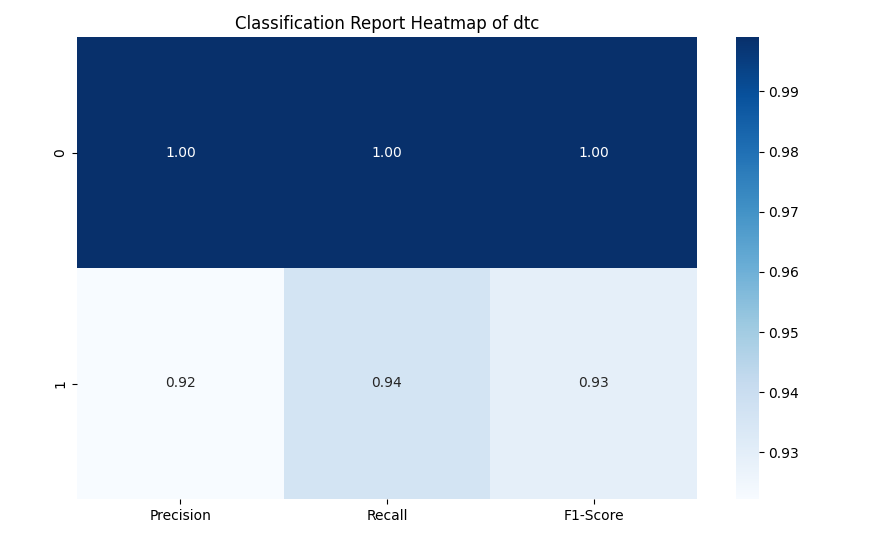


Figure 34: Classification report graph of Decision Tree classifier algorithm

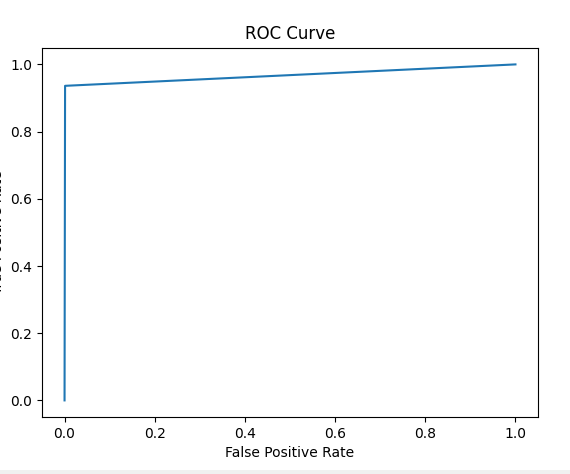


Figure 35: ROC AUC graph of Decision Tree classifier algorithm

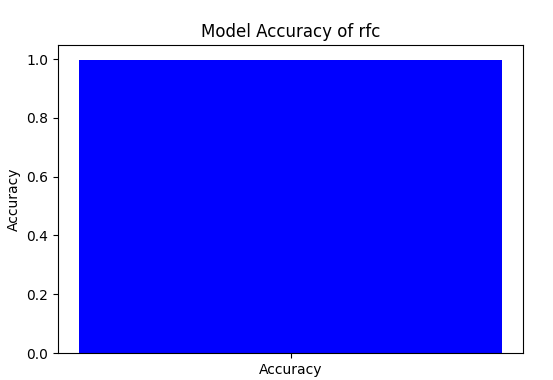


Figure 36: Accuracy graph of Random Forest classifier algorithm

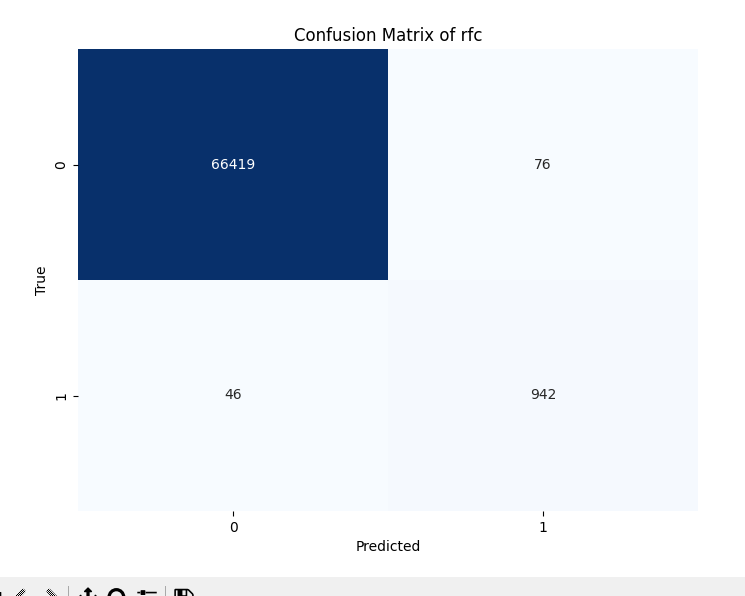


Figure 37: Confusion matrix graph of Random Forest classifier algorithm

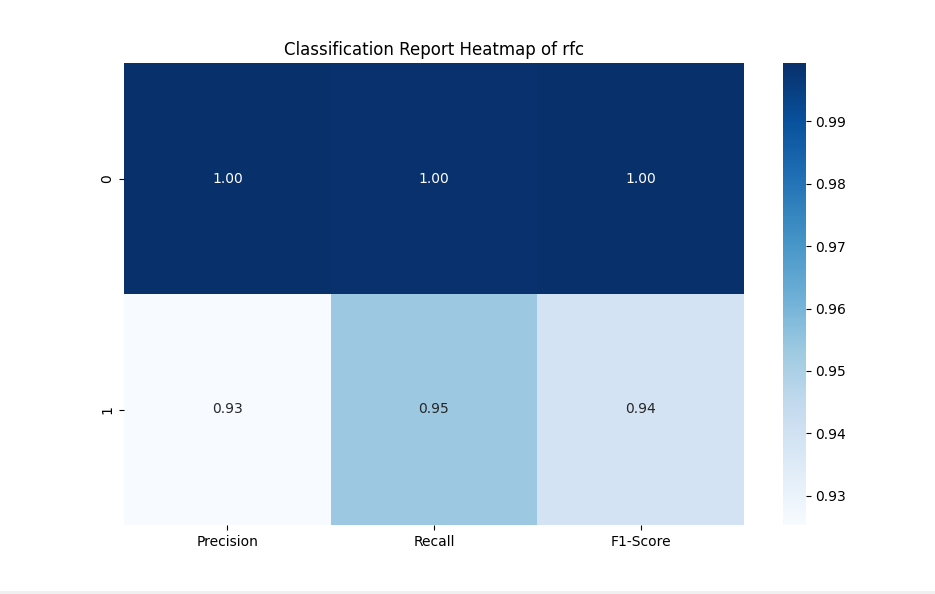


Figure 38: Classification report graph of Random Forest classifier algorithm

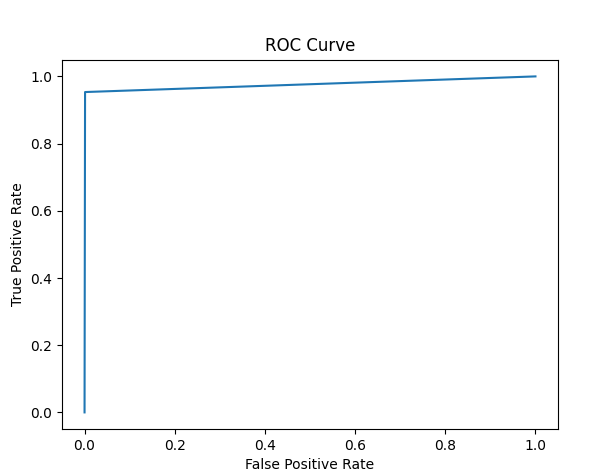


Figure 39: ROC AUC graph of Random Forest classifier algorithm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1 - score | Support |
| 0 | 1.00 | 1.00 | 1.00 | 66495 |
| 1 | 0.92 | 0.95 | 0.94 | 988 |
| accuracy |  |  | 1.00 | 67483 |
| Macro avg | 0.96 | 0.98 | 0.97 | 67483 |
| Weighted avg | 1.00 | 1.00 | 1.00 | 67483 |

Table 1: classification report of DTC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1 - score | Support |
| 0 | 0.99 | 1.00 | 0.99 | 66495 |
| 1 | 0.00 | 0.00 | 0.00 | 988 |
| accuracy |  |  | 0.99 | 67483 |
| Macro avg | 0.49 | 0.50 | 0.50 | 67483 |
| Weighted avg | 0.97 | 0.99 | 0.98 | 67483 |

Table 2: classification report of SVM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1 - score | Support |
| 0 | 1.00 | 1.00 | 1.00 | 66495 |
| 1 | 0.93 | 0.95 | 0.94 | 988 |
| accuracy |  |  | 1.00 | 67483 |
| Macro avg | 0.96 | 0.98 | 0.97 | 67483 |
| Weighted avg | 1.00 | 1.00 | 1.00 | 67483 |

Table 3: classification report of RFC

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 | 84 |
| RFC | 84 |
| SVM | 99 |

Table 4: Accuracy comparison.

|  |  |  |
| --- | --- | --- |
| Dataset Count | Training Value | Testing Value |
| 204492 | 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**

The implementation of machine learning techniques on network traffic datasets for intrusion detection holds significant promise and has been a subject of extensive research. The references provided shed light on diverse methodologies and approaches, emphasizing the importance of robust and accurate intrusion detection systems. The surveyed literature also acknowledges the challenges associated with real-world implementation, such as the need for handling imbalanced datasets, ensuring adaptability to evolving attack patterns, and addressing the computational complexities associated with large-scale network environments, including cloud computing and wireless sensor networks. Overall, the wealth of research in this domain underscores the continuous efforts to advance the capabilities of intrusion detection systems through the application of machine learning, aiming to enhance network security and safeguard against a diverse range of cyber threats.

**REFERNCES**

P. Garcia-Teodoro, et al. (2009). "Anomaly-based network intrusion detection: Techniques, systems and challenges."

S. Mukkamala, et al. (2005). "Intrusion Detection Using Ensemble of Soft Computing Paradigms."

I. K. Altunbasak, et al. (2015). "A survey of intrusion detection systems in wireless sensor networks."

A. Kumar and H. Raghav (2016). "A survey on intrusion detection systems for cloud computing."