NAME: Oluwatoba Adeoye

STUDENT ID: 23032031

COURSE ID: Individual assignment: Machine learning tutorial GITHUB LINK:

https://github.com/OluwatobaAdeoye/EVALUATING-SUPERVISED-AND-ENSEMBLE-MACHINE-LEARNING-TECHNIQUES-FOR-NETWORKINTRUSION

LINK TO DATASET:

https://www.kaggle.com/datasets/dhoogla/unswnb15

TOPIC: EVALUATING SUPERVISED AND ENSEMBLE MACHINE LEARNING TECHNIQUES FOR NETWORK INTRUSION DETECTION

ABSTRACT

This study contrasts the effectiveness of ensemble learning approaches like Gradient Boosting, Bagging, Stacking, and Voting Classifiers with supervised machine learning techniques like Logistic Regression, Decision Trees, Naive Bayes, and K-Nearest Neighbors. The intention is to demonstrate how effective ensemble approaches are at handling problems like noise, overfitting, and intricate data linkages. This investigation illustrates how ensemble models attain greater accuracy and robustness, making them more appropriate for identifying network intrusions in dynamic contexts, using the UNSW_NB15 dataset, a benchmark for network intrusion detection.

1. INTRODUCTION

In the current digital era, network security is a major concern since criminal activity and cyberattacks pose serious hazards to system integrity and sensitive data. As a defensive measure, intrusion detection systems (IDS) spot questionable activity in network traffic. However, there are issues with conventional IDS techniques, which rely on signature-based detection, include high false positive rates and a lack of flexibility in responding to new threats. (Pathmanaban et al., 2024)

A dynamic solution is provided by machine learning, which uses algorithms to more accurately and effectively detect intrusions. In IDS, supervised methods such as Decision Trees and Logistic Regression are fundamental models. These models work well for basic tasks, but they frequently have trouble with noisy data and complex interactions. By pooling the predictions of several models, ensemble approaches improve accuracy and robustness while addressing these drawbacks. The effectiveness of supervised and ensemble learning approaches for network intrusion detection is compared in this study.

2. LITERATURE SURVEY

Due to issues like overfitting and noisy datasets, early IDS implementations mostly depended on supervised learning methods like Decision Trees and Support Vector Machines, which showed only patchy results. More recently, ensemble methods have emerged as a better option. Furthermore, the DOME architecture highlights how crucial feature selection, data preprocessing, and thorough evaluation are to creating scalable machine learning models. (Walsh et al., 2020)

2.1 MACHINE LEARNING IN IDS

By allowing algorithms to recognise similarities in past data and extrapolate to novel situations, machine learning has completely transformed intrusion detection systems. The ease of use and interpretability of supervised learning methods have led to their widespread adoption. Studies using the NSL-KDD datasets, for example, have shown a modest level of success with supervised classifiers, with accuracies ranging from 70 to 90%, depending on the model and preprocessing used. However, their performance is frequently constrained by their noise sensitivity, overfitting propensities, and incapacity to accurately predict intricate, non-linear interactions. (*Machine Leaming Based Intrustion Detection*, 2023)

2.2 MACHINE LEARNING OVERVIEW

A subfield of artificial intelligence called machine learning (ML) uses datasets to train models in order to find patterns and forecast outcomes. ML can be divided into two categories for structured tasks, such as malware detection: supervised and unsupervised learning. For classification tasks, supervised and ensemble approaches are essential.

Supervised learning is the process of teaching an algorithm to map input features by using labelled data.

Ensemble learning aggregates predictions from several models to increase accuracy and decrease errors, improving overall performance.

3. METHODOLOGY

3.1 DATASET DESCRIPTION

The UNSW_NB15 dataset, which was obtained from Kaggle and is frequently used in intrusion detection research, has 45 features that represent malicious and legitimate network traffic across a range of attack types.

```
[3]: import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib as plt

     import warnings
     warnings.filterwarnings("ignore", category=FutureWarning, module="sklearn")
```

FIGURE 1: LIBRARY USED FOR IMPORTATION OF DATASET

```
[4]: df_train=pd.read_csv('UNSW_NB15_training-set.csv')
     df_test=pd.read_csv('UNSW_NB15_testing-set.csv')
```

```
[5]: df_train.head()
```

[5]:

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	...	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	ct_ftp_cmd	ct_flw_http_mthd	ct_src_
0	1	0.000011	udp	-	INT	2	0	496	0	90909.0902	...	1	2	0	0	0	
1	2	0.000008	udp	-	INT	2	0	1762	0	125000.0003	...	1	2	0	0	0	
2	3	0.000005	udp	-	INT	2	0	1068	0	200000.0051	...	1	3	0	0	0	
3	4	0.000006	udp	-	INT	2	0	900	0	166666.6608	...	1	3	0	0	0	
4	5	0.000010	udp	-	INT	2	0	2126	0	100000.0025	...	1	3	0	0	0	

5 rows × 45 columns

```
[6]: df_test.head()
```

[6]:

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	...	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	ct_ftp_cmd	ct_flw_http_mthd	ct_src_lt
0	1	0.121478	tcp	-	FIN	6	4	258	172	74.087490	...	1	1	0	0	0	
1	2	0.649902	tcp	-	FIN	14	38	734	42014	78.473372	...	1	2	0	0	0	
2	3	1.623129	tcp	-	FIN	8	16	364	13186	14.170161	...	1	3	0	0	0	
3	4	1.681642	tcp	ftp	FIN	12	12	628	770	13.677108	...	1	3	1	1	0	
4	5	0.449454	tcp	-	FIN	10	6	534	268	33.373826	...	1	40	0	0	0	

5 rows × 45 columns

FIGURE 2: NETWORK INTRUSION DATASET

3.2 Before using machine learning models, exploratory dataset analysis (EDA) is a crucial step in comprehending the dataset. It entails analyzing the structure of the dataset, finding trends, and spotting possible problems.

```
[12]: import seaborn as sns
      import matplotlib.pyplot as plt

      # Assuming 'service' is the column you want to visualize
      service_counts = df_train['service'].value_counts()

      # Creating the bar plot using Seaborn
      fig = plt.figure(figsize=(8, 8))  # Create a new figure object
      sns.barplot(x=service_counts.index, y=service_counts.values, palette="Set3")
      plt.title('Distribution of Services')
      plt.xlabel('Service')
      plt.ylabel('Count')
      plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability
      plt.tight_layout()  # Adjust layout to prevent overlapping labels
      plt.show()
```

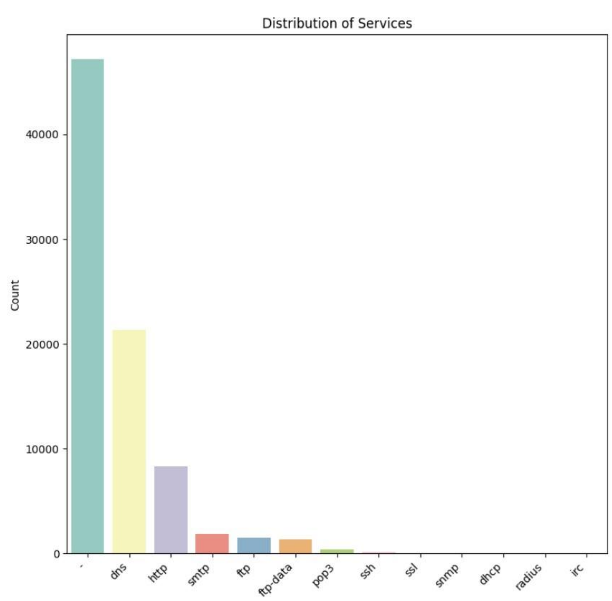


FIGURE 3: SERVICE DISTRIBUTION: VISUALIZED WITH BAR PLOTS.

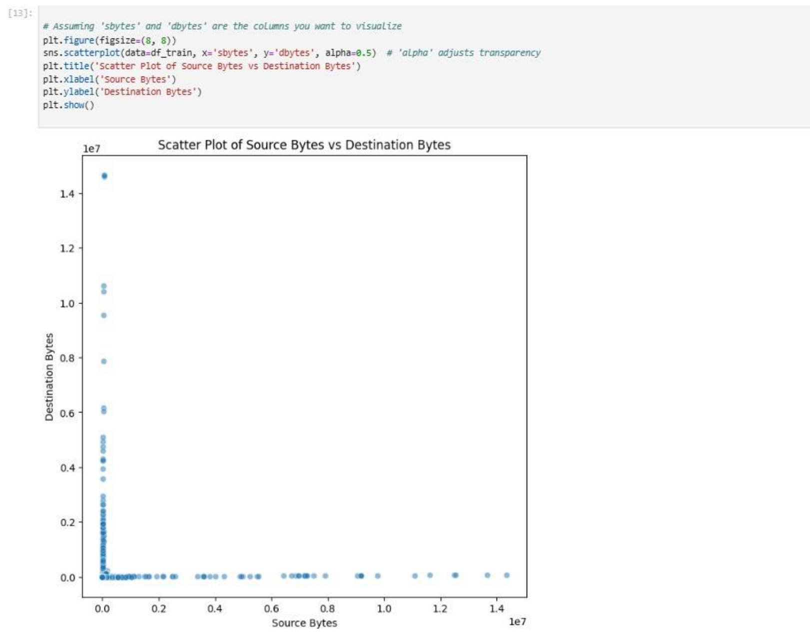


FIGURE 4: TRAFFIC ANALYSIS: SCATTER PLOTS OF SOURCE BYTES VS. DESTINATION BYTES.

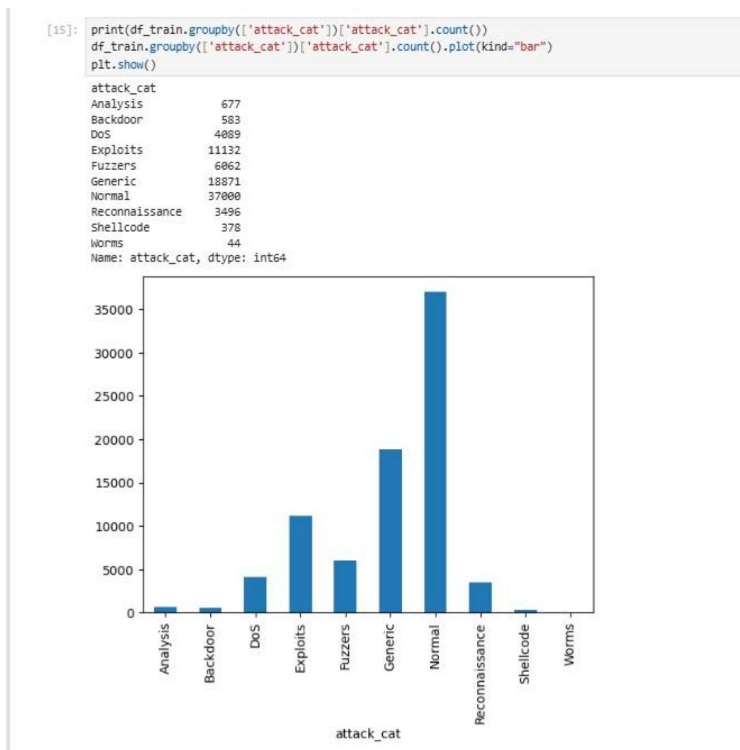


FIGURE 5: BARCHART SHOWING THE NETWORK INTRUSION

3.3 TARGET VARIABLE

The dependent variable, often known as the target variable, shows whether network attacks are occurring or not.

An attack network is indicated by a value of 1, while an assault network is represented by a value of 0.

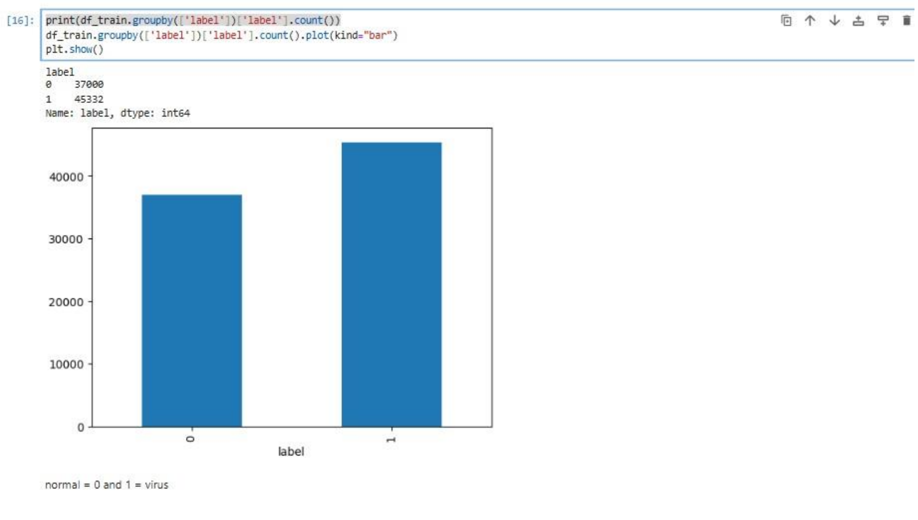


FIGURE 6: TARGET VARIABLE DISTRIBUTION

3.4 FEATURE ENGINEERING

Feature engineering involves selecting and transforming raw data into meaningful attributes that can improve the performance of machine learning models.

3.4.1 CONCATENATION

Concatenation is the process of merging two or more datasets into one. When working with distinct training and testing datasets or combining several feature sets for analysis, machine learning operations frequently employ this technique.

To make preparation more efficient, the training (df_train) and testing (df_test) datasets were combined. This guarantees consistent application of operations such as scaling, encoding, and managing missing values across the two datasets.

```
[18]: df_train['label']='df_train'
      df_test['label']='df_test'

[19]: data=pd.concat([df_train,df_test])
      data.shape

[19]: (257673, 45)

[20]: X=data.drop(['label','state'],axis=1)
      y= data['label']

[21]: X=X.select_dtypes(np.number)
      X
```

[21]:

	id	dur	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	sload	...	ct_dst_ltm	ct_src_dport_ltm	ct_dst_sport_ltm	ct_dst_src_ltm	is_ft
0	1	0.000011	2	0	496	0	90909.090200	254	0	1.803636e+08	...	1	1	1	2	
1	2	0.000008	2	0	1762	0	125000.000300	254	0	8.810000e+08	...	1	1	1	2	
2	3	0.000005	2	0	1068	0	200000.005100	254	0	8.544000e+08	...	1	1	1	3	
3	4	0.000006	2	0	900	0	166666.660800	254	0	6.000000e+08	...	2	2	1	3	
4	5	0.000010	2	0	2126	0	100000.002500	254	0	8.504000e+08	...	2	2	1	3	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
175336	175337	0.000009	2	0	114	0	111111.107200	254	0	5.066666e+07	...	24	24	13	24	
175337	175338	0.505762	10	8	620	354	33.612649	254	252	8.826286e+03	...	1	1	1	2	
175338	175339	0.000009	2	0	114	0	111111.107200	254	0	5.066666e+07	...	3	3	3	13	
175339	175340	0.000009	2	0	114	0	111111.107200	254	0	5.066666e+07	...	30	30	14	30	
175340	175341	0.000009	2	0	114	0	111111.107200	254	0	5.066666e+07	...	30	30	16	30	

257673 rows × 40 columns

FIGURE 7: NEW SET OF DATASETS GENERATED AFTER CONCATENATION

3.4.2 LABEL ENCODING

One preprocessing method for turning categorical data into numerical values is label encoding. Label encoding converts text labels (categories) into numbers, which are frequently needed for machine learning models.

An integer value is assigned to each distinct category inside a feature. For instance:

["normal", "attack"] were the original categories.

Labels encoded: [0, 1]

In machine learning algorithms that are unable to directly handle categorical inputs, this procedure guarantees that the data can be utilized.

```
[22]: from sklearn import preprocessing as pp
      label_encoder=pp.LabelEncoder()
      label_encoder

[22]: ▾ LabelEncoder
      LabelEncoder()

[23]: for column in data:
          data['label']=label_encoder.fit_transform(data['label'])

[24]: y=data['label']
```

FIGURE 8: LABEL ENCODER.

3.5 DATA PREPROCESSING

Splitting Data: The dataset is split into 80% for training and 20% for testing.

Feature Scaling: Standardization of features for models sensitive to scale

```
[27]: from sklearn.model_selection import train_test_split

      X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=101)
      from sklearn.metrics import confusion_matrix, classification_report
      from sklearn.metrics import accuracy_score,confusion_matrix
```

FIGURE 9: SPLITTING DATA INTO TEST (20%) AND TRAIN (80%)

4. MACHINE LEARNING TECHNIQUES

4.1 SUPERVISED MACHINE LEARNING MODEL

- **Logistic Regression (LR):** A simple linear model used for binary classification tasks, offering good interpretability and efficiency in predicting outcomes based on input features.

```
[28]: from sklearn.linear_model import LogisticRegression

      # Initialize the logistic regression model
      log_reg = LogisticRegression()

      # Train the logistic regression model on the training data
      log_reg.fit(X_train, y_train)

      # Predict the labels for the training set
      y_pred_train_log_reg = log_reg.predict(X_train)

      # Calculate the accuracy on the training set
      train_accuracy_log_reg = accuracy_score(y_train, y_pred_train_log_reg)

      # Predict the labels for the test set
      y_pred_log_reg = log_reg.predict(X_test)

      # Calculate the accuracy on the test set
      accuracy_log_reg = accuracy_score(y_test, y_pred_log_reg)

      # Print the accuracy of logistic regression
      print('Accuracy of logistic regression:', accuracy_log_reg)
      print('==================================================')

      C:\Users\SBMCODED\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed t
      converge (status=1):
      STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

      Increase the number of iterations (max_iter) or scale the data as shown in:
          https://scikit-learn.org/stable/modules/preprocessing.html
      Please also refer to the documentation for alternative solver options:
          https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
        n_iter_i = _check_optimize_result(
      Accuracy of logistic regression: 0.701057533714951
      ==================================================
```

FIGURE 10: LOGISTIC REGRESSION MODEL.

- **Decision Tree** is a supervised machine learning model used for classification and regression tasks. In the context of network intrusion detection, it helps classify network activities into normal or intrusive behavior based on certain features.

```
from sklearn.tree import DecisionTreeClassifier

# Initialize the decision tree classifier
decision_tree = DecisionTreeClassifier()

# Train the decision tree classifier on the training data
decision_tree.fit(X_train, y_train)

# Predict the labels for the training set
y_pred_train_decision_tree = decision_tree.predict(X_train)

# Calculate the accuracy on the training set
train_accuracy_decision_tree = accuracy_score(y_train, y_pred_train_decision_tree)

# Predict the labels for the test set
y_pred_decision_tree = decision_tree.predict(X_test)

# Calculate the accuracy on the test set
accuracy_decision_tree = accuracy_score(y_test, y_pred_decision_tree)

# Print the accuracy of decision tree classifier
print('Accuracy of decision tree classifier:', accuracy_decision_tree)
print('==================================================')

Accuracy of decision tree classifier: 0.9555835839720578
==================================================
```

FIGURE 11: DECISION TREE MODEL

- **Gaussian Naive Bayes (GNB):** Based on Bayes' theorem, this model assumes feature independence and uses Gaussian distributions to predict outcomes. It is computationally efficient but may struggle when features are correlated.

```
[32]: from sklearn.naive_bayes import GaussianNB

# Initialize the Naive Bayes classifier
naive_bayes = GaussianNB()

# Train the Naive Bayes classifier on the training data
naive_bayes.fit(X_train, y_train)

# Predict the labels for the training set
y_pred_train_naive_bayes = naive_bayes.predict(X_train)

# Calculate the accuracy on the training set
train_accuracy_naive_bayes = accuracy_score(y_train, y_pred_train_naive_bayes)

# Predict the labels for the test set
y_pred_naive_bayes = naive_bayes.predict(X_test)

# Calculate the accuracy on the test set
accuracy_naive_bayes = accuracy_score(y_test, y_pred_naive_bayes)

# Print the accuracy of Naive Bayes classifier
print('Accuracy of Naive Bayes classifier:', accuracy_naive_bayes)
print('==================================================')

Accuracy of Naive Bayes classifier: 0.6336082274182594
==================================================
```

FIGURE 12: NAÏVE BAYES

- **K-Nearest Neighbors** (KNN) is a simple and widely used machine learning algorithm that can be applied to network intrusion detection. It classifies data points (network traffic) based on the majority class among the "k" closest neighbors in the feature space.

```
[34]: from sklearn.neighbors import KNeighborsClassifier

# Initialize the KNN classifier
knn = KNeighborsClassifier()

# Train the KNN classifier on the training data
knn.fit(X_train, y_train)

# Predict the labels for the training set
y_pred_train_knn = knn.predict(X_train)

# Calculate the accuracy on the training set
train_accuracy_knn = accuracy_score(y_train, y_pred_train_knn)

# Predict the labels for the test set
y_pred_knn = knn.predict(X_test)

# Calculate the accuracy on the test set
accuracy_knn = accuracy_score(y_test, y_pred_knn)

# Print the accuracy of KNN classifier
print('Accuracy of KNN classifier:', accuracy_knn)
print('==================================================')

Accuracy of KNN classifier: 0.7817793732414864
==================================================
```

FIGURE 13: KNN MODEL

4.2 EMSEMBLE MACHINE LEARNING MODEL.

4.2.1 BASED MODEL

A base model is a basic machine learning model that serves as the cornerstone of an ensemble learning strategy like stacking, bagging, or boosting. Utilizing a base model is based on the theory that, although a single model may not function at its best, combining multiple base models might improve performance overall.

• One machine learning method that is frequently applied to classification and regression problems is gradient boosting. Using network traffic data, it is used in the context of network intrusion detection to find malicious activity or network intrusions.

```
[37]: from sklearn.ensemble import GradientBoostingClassifier

[38]: gbc=GradientBoostingClassifier()
gbc.fit(X_train,y_train)
y_pred_train_gbc=gbc.predict(X_train)
train_accuracy_gbc=accuracy_score(y_train,y_pred_train_gbc)
y_pred_gbc=gbc.predict(X_test)
accuracy=accuracy_score(y_test,y_pred_gbc)
print('Accuracy of gradient boosting',accuracy)
print('==================================================')

Accuracy of gradient boosting 0.908256524691957
==================================================
```

FIGURE 14: GRAIDENT BOOSTING MODEL

• **A bagging classifier**, also known as bootstrap aggregating, is an ensemble learning technique that enhances machine learning models' performance by merging several weak models (usually decision trees) to produce a more robust and accurate prediction. Through the classification of network traffic data, it can be applied to the identification of harmful activity, including assaults or breaches, in the context of network intrusion detection.

```
[40]: from sklearn.ensemble import BaggingClassifier

[41]:
bc= BaggingClassifier()
bc.fit(X_train,y_train)
y_pred_train_bc=bc.predict(X_train)
train_accuracy_bc=accuracy_score(y_train,y_pred_train_bc)
y_pred_bc=bc.predict(X_test)
accuracy1=accuracy_score(y_test,y_pred_bc)
print(' Accuracy of bagging',accuracy1)
print('==================================================')

 Accuracy of bagging 0.9635005336179295
==================================================
```

FIGURE 15: BAGGING CLASSIFIER

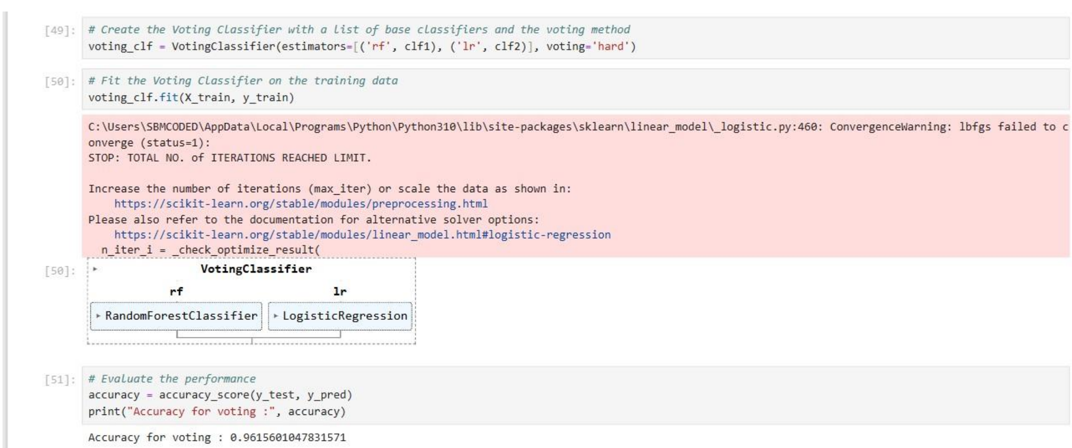
- **A stacking classifier** is a method of ensemble learning that enhances performance by combining the predictions of several basic models, or learners. The basic idea is to employ multiple base models, train them separately on the same data, and then use a meta-model, also called a stacking model, to combine their predictions. The objective is to increase prediction accuracy by utilising the advantages of several models.

In the context of network intrusion detection, a stacking classifier can be used to identify malicious network activity by combining the predictions of different base models, such as Random Forest and Logistic Regression.

```
[43]: from sklearn.metrics import roc_auc_score as roc
      from sklearn.ensemble import RandomForestClassifier
      from sklearn.linear_model import LogisticRegression
      from mlxtend.classifier import StackingClassifier

      # Create individual classifiers
      clf1 = RandomForestClassifier()
      clf2 = LogisticRegression()

      # Create the stacking classifier with a list of classifiers and a meta-classifier
      stacking_clf = StackingClassifier(classifiers=[clf1, clf2], meta_classifier=clf2)

[44]: # Fit the stacking classifier on the training data
      stacking_clf.fit(X_train, y_train)

      C:\Users\SBMCODED\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to c
      onverge (status=1):
      STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

      Increase the number of iterations (max_iter) or scale the data as shown in:
          https://scikit-learn.org/stable/modules/preprocessing.html
      Please also refer to the documentation for alternative solver options:
          https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
        n_iter_i = _check_optimize_result(
[44]:        StackingClassifier
      meta_classifier: LogisticRegression
             LogisticRegression

[45]: # Make predictions on the testing data
      y_pred = stacking_clf.predict(X_test)

[46]: # Evaluate the performance
      accuracy = accuracy_score(y_test, y_pred)
      print("Accuracy of stacking classifier:", accuracy)

      Accuracy of stacking classifier: 0.9615601047831571
```

FIGURE 16: STACKING CLASSIFIER MODEL

- **A voting classifier** is an ensemble learning technique that combines several separate base models, sometimes referred to as classifiers, to produce a prediction. The final prediction is determined by a majority vote (in classification tasks) or the average of predictions (in regression tasks), after the voting classifier has aggregated the predictions of base models.



```
[49]: # Create the Voting Classifier with a list of base classifiers and the voting method
      voting_clf = VotingClassifier(estimators=[('rf', clf1), ('lr', clf2)], voting='hard')

[50]: # Fit the Voting Classifier on the training data
      voting_clf.fit(X_train, y_train)

      C:\Users\SBMCODED\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to c
      onverge (status=1):
      STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

      Increase the number of iterations (max_iter) or scale the data as shown in:
          https://scikit-learn.org/stable/modules/preprocessing.html
      Please also refer to the documentation for alternative solver options:
          https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
        n_iter_i = _check_optimize_result(
[50]:             VotingClassifier
                rf                        lr
      RandomForestClassifier    LogisticRegression

[51]: # Evaluate the performance
      accuracy = accuracy_score(y_test, y_pred)
      print("Accuracy for voting :", accuracy)

      Accuracy for voting : 0.9615601047831571
```

FIGURE 17: VOTING CLASSIFIER MODEL

5. PERFORMANCE EVALUATION

The practice of evaluating a machine learning model's performance using a variety of indicators to ascertain how effectively it **generalizes** to unknown data is known as performance evaluation. This aids in determining if the model is operating at its best or overfitting or **under fitting**.

5.1 CONFUSION MATRIX

A confusion matrix is a table that compares the expected and actual values of a classification model to assess how well it performs. True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) are the four values used in binary classification. Metrics like Accuracy, Precision, Recall, and F1-Score are computed with the use of these variables. Recall stresses false negatives, Precision concentrates on false positives, and Accuracy gauges' overall correctness. The F1Score strikes a compromise between recall and precision. When evaluating model performance in unbalanced datasets, where accuracy by itself can be deceptive, the confusion matrix is especially helpful. It assists in locating a model's error areas.

```
26]: from sklearn import metrics
def plot_confusion_metrix(y_test,model_test):
    cm = metrics.confusion_matrix(y_test, model_test)
    plt.figure(1)
    plt.clf()
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
    classNames = ['normal','virus']
    plt.title('Confusion Matrix')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    tick_marks = np.arange(len(classNames))
    plt.xticks(tick_marks, classNames)
    plt.yticks(tick_marks, classNames)
    s = [['TN','FP'], ['FN', 'TP']]
    for i in range(2):
        for j in range(2):
            plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j]))
    plt.show()
```

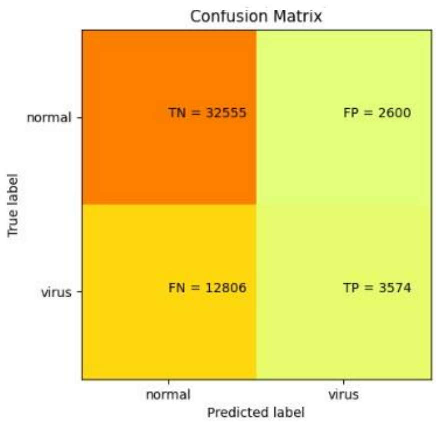
FIGURE 18: CONFUSION MATRIX CODE.

5.2 A graphical depiction called ROC (Receiver Operating Characteristic) is used to assess how well a binary classification model performs. At different categorisation thresholds, it compares the True Positive Rate (TPR), often referred to as Recall, against the False Positive Rate (FPR).

```
[29]: log_reg.fit(X_train,y_train)
      pred_log=log_reg.predict(X_test)
      report_performance(log_reg)
      roc_curves(log_reg)
```

```
C:\Users\SBMCODED\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to c
onverge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Confusion Matrix:
[[32555  2600]
 [12806  3574]]


Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.93      0.81     35155
           1       0.58      0.22      0.32     16380

    accuracy                           0.70     51535
   macro avg       0.65      0.57      0.56     51535
weighted avg       0.67      0.70      0.65     51535
```



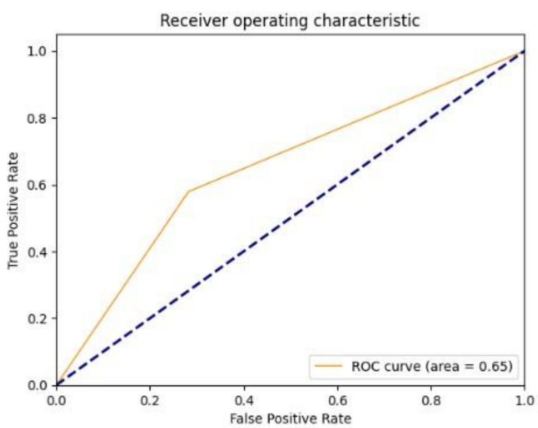
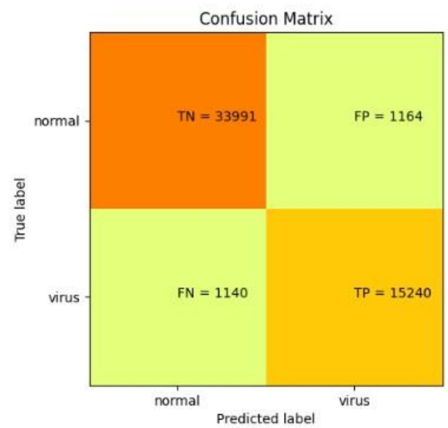


FIGURE 19: PERFORMANCE EVALUATION FOR LOGISTIC REGRESSION (CLASSIFICATION REPORT, CONFUSION MATRIC AND ROC CURVE)

```
[31]: decision_tree.fit(X_train,y_train)
      pred_dec=decision_tree.predict(X_test)
      report_performance(decision_tree)
      roc_curves(decision_tree)
```

```
Confusion Matrix:
[[33991  1164]
 [ 1140 15240]]


Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97     35155
           1       0.93      0.93      0.93     16380

    accuracy                           0.96     51535
   macro avg       0.95      0.95      0.95     51535
weighted avg       0.96      0.96      0.96     51535
```



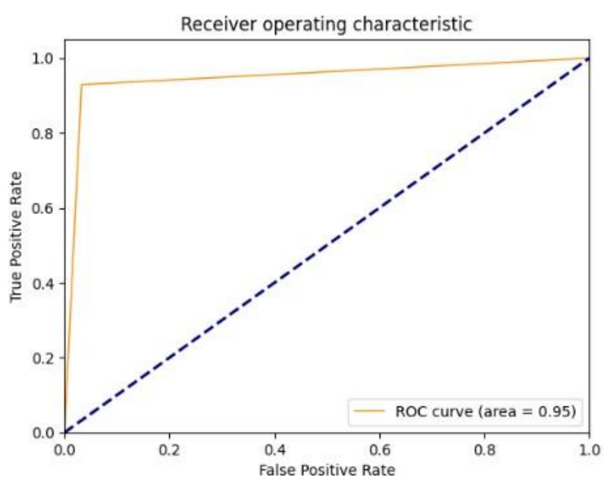
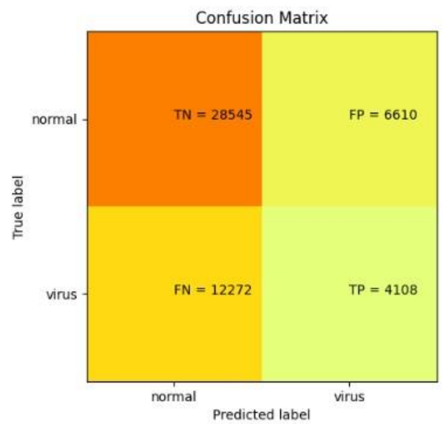


FIGURE 20: PERFORMANCE EVALUATION FOR DECISION TREE (CLASSIFICATION REPORT, CONFUSION MATRIC AND ROC CURVE)

```
[33]: naive_bayes.fit(X_train,y_train)
      pred_nai=naive_bayes.predict(X_test)
      report_performance(naive_bayes)
      roc_curves(naive_bayes)
```

```
Confusion Matrix:
[[28545  6610]
 [12272  4108]]


Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.81      0.75     35155
           1       0.38      0.25      0.30     16380

    accuracy                           0.63     51535
   macro avg       0.54      0.53      0.53     51535
weighted avg       0.60      0.63      0.61     51535
```



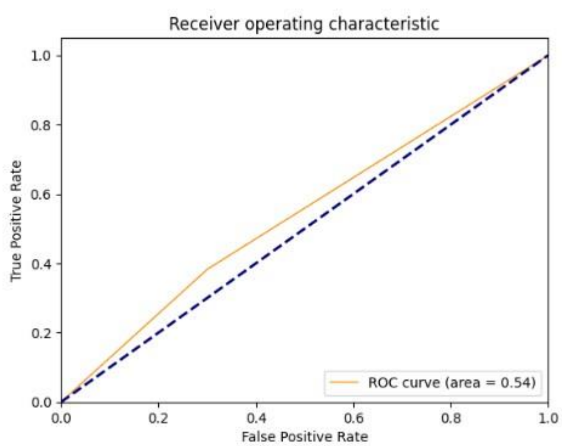
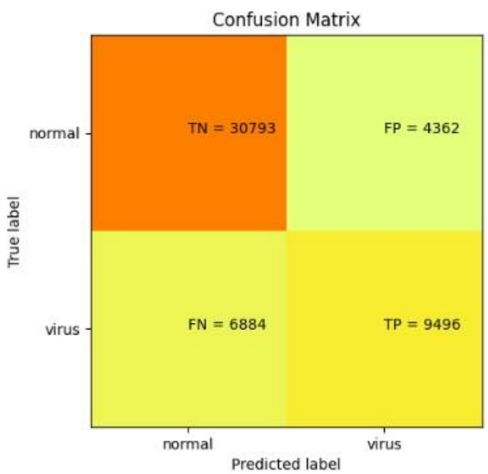


FIGURE 21: PERFORMANCE EVALUATION FOR NAÏVE BAYES

(CLASSIFICATION REPORT, CONFUSION MATRIC AND ROC CURVE)

```
[35]: knn.fit(X_train,y_train)
      pred_knn=knn.predict(X_test)
      report_performance(knn)
      roc_curves(knn)
```

```
Confusion Matrix:
[[30793  4362]
 [ 6884  9496]]


Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.88      0.85     35155
           1       0.69      0.58      0.63     16380

    accuracy                           0.78     51535
   macro avg       0.75      0.73      0.74     51535
weighted avg       0.78      0.78      0.78     51535
```



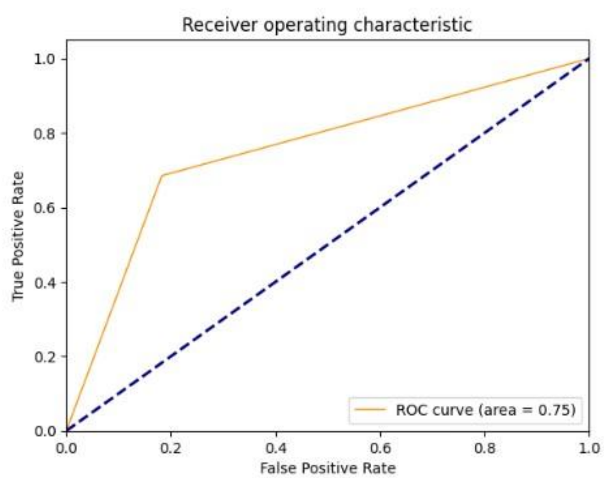


FIGURE 22: PERFORMANCE EVALUATION FOR KNN

(CLASSIFICATION REPORT, CONFUSION MATRIC AND ROC CURVE)

```python
[36]: # Sample data for algorithm performance comparison
algorithms = ['logistic regression', 'decision tree', 'naive bayes', 'KNN']
accuracy_scores = [0.70, 0.95, 0.63, 0.78]

# Set custom colors
colors = sns.color_palette('pastel')

# Plotting the comparison using seaborn
plt.figure(figsize=(8, 5))
sns.barplot(x=algorithms, y=accuracy_scores, palette=colors)
plt.xlabel('Algorithms')
plt.ylabel('Accuracy Score')
plt.title('Comparison of supervised Machine Learning Algorithms')
plt.ylim(0, 1)  # Set the y-axis limits between 0 and 1

# Display the accuracy scores on top of the bars
for i, score in enumerate(accuracy_scores):
    plt.text(i, score, str(score), ha='center', va='bottom')

plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust the layout to prevent overlapping elements
plt.show()
```

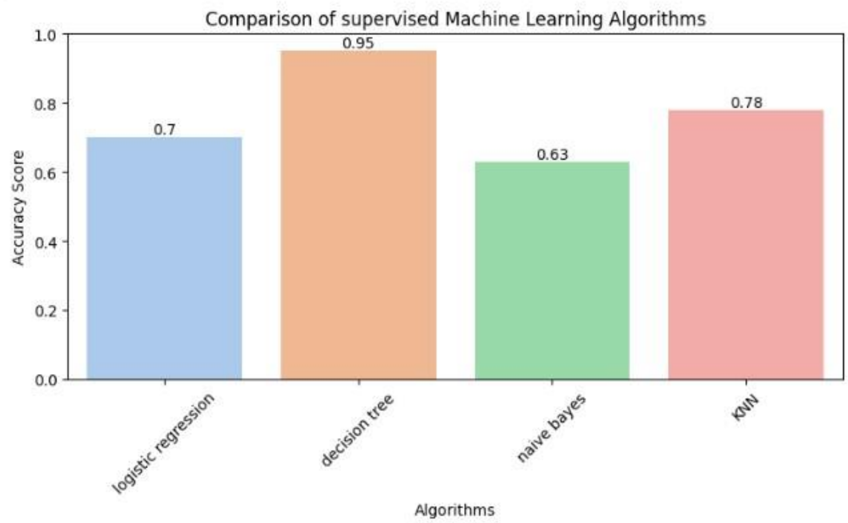
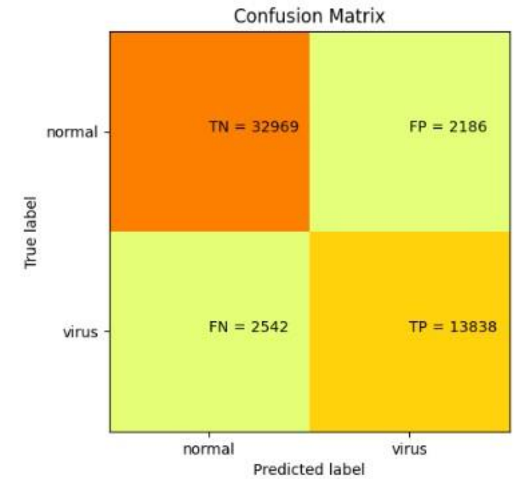


FIGURE 23: ACCURACY FOR THE SUPERVISED MACHINE LEARNING MODEL USED FOR NETWORK INTRUSION DETECTION

```
[39]: gbc.fit(X_train,y_train)
      pred_gbc=gbc.predict(X_test)
      report_performance(gbc)
      roc_curves(gbc)
```

```
Confusion Matrix:
[[32969  2186]
 [ 2542 13838]]


Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.93     35155
           1       0.86      0.84      0.85     16380

    accuracy                           0.91     51535
   macro avg       0.90      0.89      0.89     51535
weighted avg       0.91      0.91      0.91     51535
```



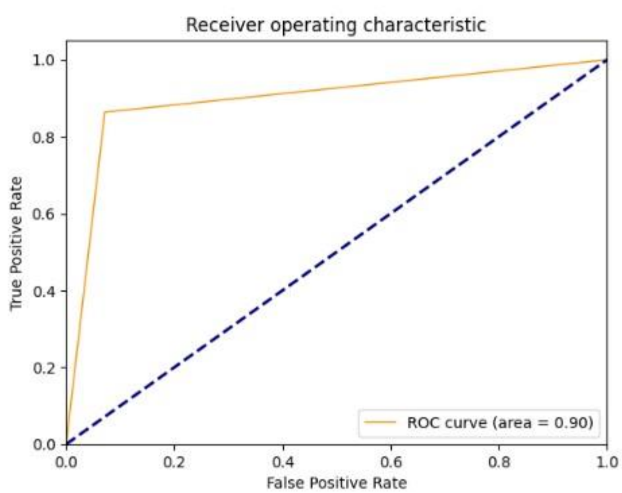
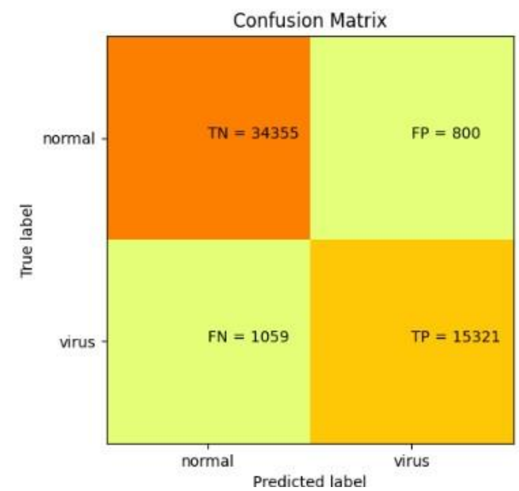


FIGURE 24: PERFORMANCE EVALUATION FOR GRADIENT BOOSTING

(CLASSIFICATION REPORT, CONFUSION MATRIC AND ROC CURVE)

[42]:
```
bc.fit(X_train,y_train)
pred_bc=bc.predict(X_test)
report_performance(bc)
roc_curves(bc)
```

```
Confusion Matrix:
[[34355   800]
 [ 1059 15321]]


Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.97     35155
           1       0.95      0.94      0.94     16380

    accuracy                           0.96     51535
   macro avg       0.96      0.96      0.96     51535
weighted avg       0.96      0.96      0.96     51535
```



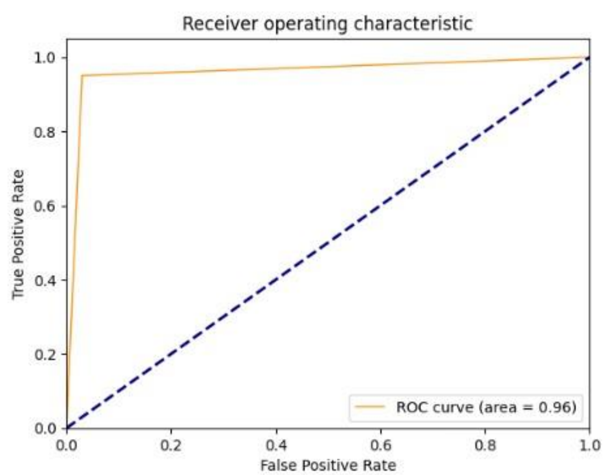


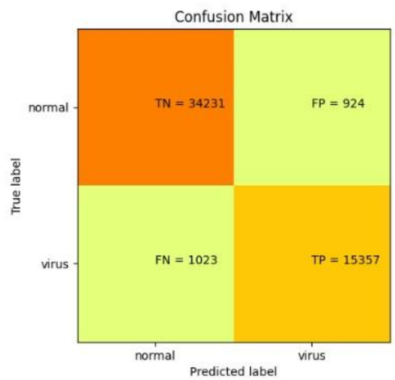
FIGURE 25: PERFORMANCE EVALUATION FOR BAGGING CLASSIFIER MODEL

(CLASSIFICATION REPORT, CONFUSION MATRIC AND ROC CURVE)

```
[47]: stacking_clf.fit(X_train,y_train)
      pred_stc=stacking_clf.predict(X_test)
      report_performance(stacking_clf)
      roc_curves(stacking_clf)
```

```
C:\Users\SBMCODED\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Confusion Matrix:
[[34231   924]
 [ 1023 15357]]


Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97     35155
           1       0.94      0.94      0.94     16380

    accuracy                           0.96     51535
   macro avg       0.96      0.96      0.96     51535
weighted avg       0.96      0.96      0.96     51535
```



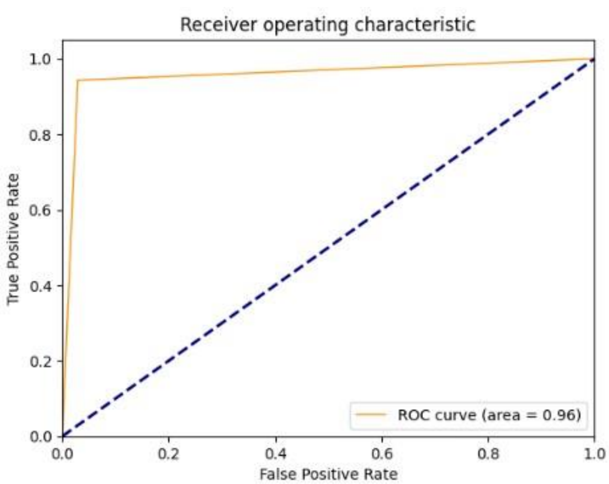


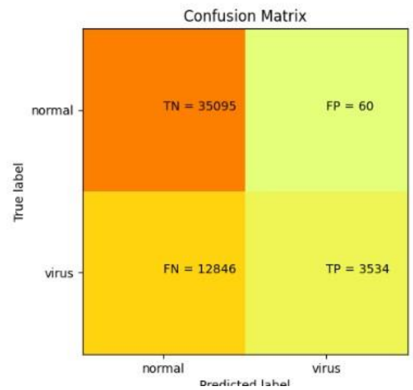
FIGURE 26: PERFORMANCE EVALUATION FOR STACKING CLASSIFIER MODEL

(CLASSIFICATION REPORT, CONFUSION MATRIC AND ROC CURVE)

```
[52]: voting_clf.fit(X_train,y_train)
      pred_vtc=voting_clf.predict(X_test)
      report_performance(voting_clf)
      roc_curves(voting_clf)
```

```
C:\Users\SBMCODED\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbf
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Confusion Matrix:
[[35095    60]
 [12846  3534]]


Classification Report:
              precision    recall  f1-score   support

           0       0.73      1.00      0.84     35155
           1       0.98      0.22      0.35     16380

    accuracy                           0.75     51535
   macro avg       0.86      0.61      0.60     51535
weighted avg       0.81      0.75      0.69     51535
```



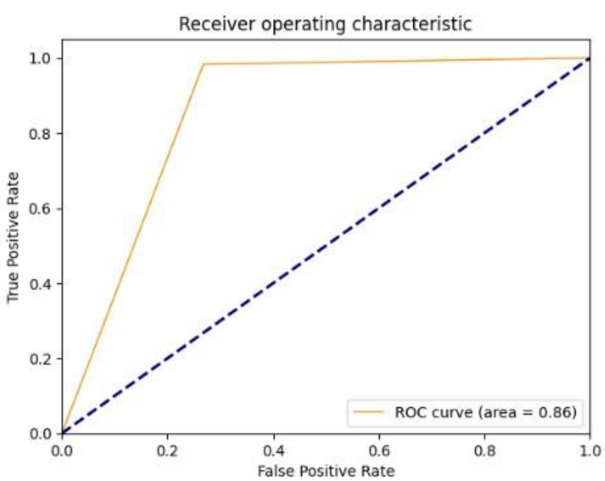


FIGURE 27: PERFORMANCE EVALUATION FOR VOTING CLASSIFIER

(CLASSIFICATION REPORT, CONFUSION MATRIC AND ROC CURVE)

```python
[53]: import seaborn as sns

# Sample data for algorithm performance comparison
algorithms = ['Gradient Boosting', 'Bagging classifier', 'Stacking classifier', 'voting classifier']
accuracy_scores = [0.91, 0.96, 0.96, 0.96]

# Set custom colors
colors = sns.color_palette('pastel')

# Plotting the comparison using seaborn
plt.figure(figsize=(8, 5))
sns.barplot(x=algorithms, y=accuracy_scores, palette=colors)
plt.xlabel('Algorithms')
plt.ylabel('Accuracy Score')
plt.title('Comparison of Ensemble Machine Learning Algorithms')
plt.ylim(0, 1)  # Set the y-axis limits between 0 and 1

# Display the accuracy scores on top of the bars
for i, score in enumerate(accuracy_scores):
    plt.text(i, score, str(score), ha='center', va='bottom')

plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust the layout to prevent overlapping elements
plt.show()
```

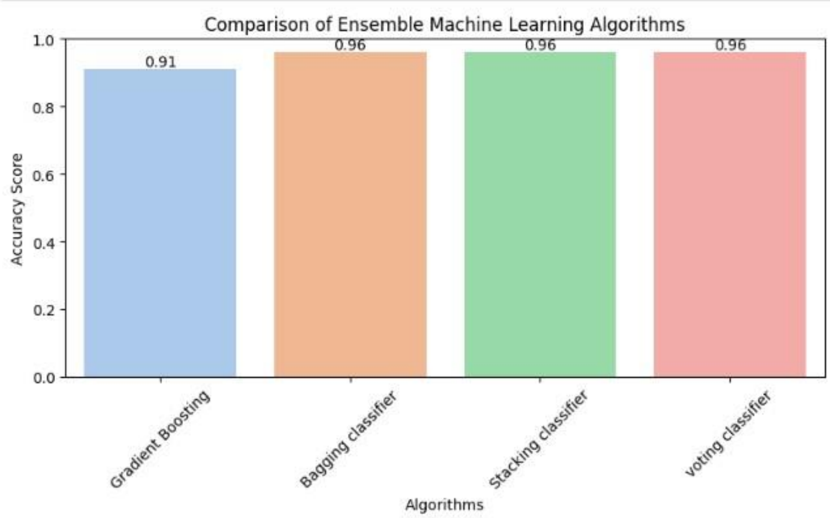


FIGURE 28: ACCURACY FOR ENSEMBLE MODEL USED FOR NETWORK INTRUSION

6. COMPARATIVE ANALYSIS

In network intrusion detection, ensemble models typically perform better than supervised models because they can mix many classifiers, which lowers overfitting and increases accuracy. Ensemble models are more resilient and manage these issues better than supervised models, which need a lot of labelled data and may have trouble with noise or imbalance. By combining the advantages of several models, they become less error-prone and more efficient at challenging tasks. Ensemble approaches are the recommended option for network intrusion detection due to their improved performance, despite their higher computing cost. Although simpler, supervised models are not as accurate or resilient.

FIGURE 29: COMPARATIVE ANALYSIS BETWEEN ENSEMBLE AND SUPERVISED MACHINE LEARNING MODEL.

7. DISCUSSION

By successfully lowering bias and variance, ensemble models in particular, **stacking** performed better than conventional supervised techniques. Significant gains were also seen in bagging and voting, demonstrating the value of combining various models. Real-world implementation may be hampered by issues with interpretability and processing complexity.

7.1 CONCLUSION

The superiority of ensemble learning techniques in network intrusion detection is demonstrated in this paper. Ensembles offer a reliable, scalable, and accurate IDS solution by fusing the advantages of several models. These results provide credence to the use of group approaches to tackle changing cybersecurity issues.

REFERENCES

Machine Leaming based Intrustion Detection. (2023).

Pathmanaban, J., James, P. G., Ashok, P., Ragesh, B., Aakash, S., & Kaushik, N. (2024). Phishing Website Detection Using Machine Learning. Proceedings of the 2nd IEEE International Conference on Networking and Communications 2024, ICNWC 2024, 2024(30). https://doi.org/10.1109/ICNWC60771.2024.10537279

Walsh, I., Titma, T., Psomopoulos, F. E., & Tosatto, S. (2020). Recommendations for machine learning validation in biology. ArXiv, June, 23. https://arxiv.org/vc/arxiv/papers/2006/2006.16189v1.pdf

LINKS

- https://www.researchgate.net/publication/386179895_Phishing_Website_Detection_using_Machine_Learning
- https://www.researchgate.net/publication/372337587_Machine_Learning_based_Intrustion_Detection_System_for_IoT_Applications_using_Explainable_AI
- https://www.researchgate.net/publication/385671415_An_Efficient_Network_Intrusion_Detection_and_Classification_System_using_Machine_Learning