Malware Detection Using Five Algorithm

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix, accuracy_score, classificatio
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
import matplotlib.pyplot as plt
```

Lode the Dataset

In [2]:
```
Train_data = pd.read_csv('UNSW_2018_IoT_Botnet_Final_10_best_Training.csv')
Test_data = pd.read_csv('UNSW_2018_IoT_Botnet_Final_10_best_Testing.csv')
```

In [3]:
```
column_names = [
    "proto", "saddr", "sport", "daddr", "dport", "seq", "stddev", "N_IN_Con
    "min", "state_number", "mean", "N_IN_Conn_P_DstIP", "drate", "srate", "
]
```

In [4]:
```
X_train_def = pd.DataFrame(data=Train_data, columns=column_names)
y_train_def = pd.DataFrame(data=Train_data, columns=["attack"])
```

PreprocessThe Data(Standardization The Data && Handel The Catagorical Data By Label Encoding)

In [5]:
```
columns_to_label_encode = ["proto", "saddr", "sport", "daddr", "dport", "st
label_encoder = LabelEncoder()
for column in columns_to_label_encode:
    X_train_def[column] = label_encoder.fit_transform(X_train_def[column])
```

In [6]:
```
X_test = pd.DataFrame(data=Test_data, columns=column_names)
y_test = pd.DataFrame(data=Test_data, columns=["attack"])
for column in columns_to_label_encode:
    X_test[column] = label_encoder.fit_transform(X_test[column])
```

In [7]:
```
scaler = StandardScaler()
X_train_def = scaler.fit_transform(X_train_def)
X_test = scaler.transform(X_test)
```

Decision Tree Algorithm

In [8]:
```
clf_gini = DecisionTreeClassifier(criterion="gini", random_state=100, max_d
clf_gini.fit(X_train_def, y_train_def)
y_pred_tree = clf_gini.predict(X_test)
```

SVM Algorithm

In [9]:
```
svm_poly = SVC(kernel='poly',degree=4)
svm_poly.fit(X_train_def, y_train_def.values.ravel())
y_pred_svm_poly = svm_poly.predict(X_test)
```

In [10]:
```
svm_rbf = SVC(kernel='rbf',gamma=0.1,C=25,random_state=0)
svm_rbf.fit(X_train_def, y_train_def.values.ravel())
y_pred_svm_rbf = svm_rbf.predict(X_test)
```

In [11]:
```
classifier=SVC(kernel='linear',random_state=0)
classifier.fit(X_train_def,y_train_def.values.ravel())
Y_pred_L=classifier.predict(X_test)
```

Adaboosting On Decision Tree

In [12]:
```
estimator=DecisionTreeClassifier(max_depth=5, min_samples_leaf=64)
adaboost_clf = AdaBoostClassifier(estimator=estimator,n_estimators=50, rand
adaboost_clf.fit(X_train_def, y_train_def.values.ravel())
y_pred_adaboost = adaboost_clf.predict(X_test)
```

KNN Algorithm

In [13]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_def, y_train_def.values.ravel())
y_pred_knn = knn.predict(X_test)
```

Naive Bias Algorithm

In [14]:
```
gnb = GaussianNB()
gnb.fit(X_train_def, y_train_def.values.ravel())
y_pred_gnb = gnb.predict(X_test)
```

In [15]:
```
bnb = BernoulliNB()
bnb.fit(X_train_def, y_train_def.values.ravel())
y_pred_bnb = bnb.predict(X_test)
```

All Report

In [16]:
```
print("Decision Tree Classifier:")
print("Confusion Matrix: ", confusion_matrix(y_test, y_pred_tree))
print("Accuracy : ", accuracy_score(y_test, y_pred_tree) * 100)
print("Report : ", classification_report(y_test, y_pred_tree))
roc_auc_tree = roc_auc_score(y_test, y_pred_tree)
print("ROC-AUC Score: ", roc_auc_tree)
```

```
Decision Tree Classifier:
Confusion Matrix:  [[    88     19]
 [    50 733548]]
Accuracy :  99.99059567537361
Report :                precision    recall  f1-score   support

           0       0.64      0.82      0.72       107
           1       1.00      1.00      1.00    733598

    accuracy                           1.00    733705
   macro avg       0.82      0.91      0.86    733705
weighted avg       1.00      1.00      1.00    733705

ROC-AUC Score:  0.9111808746612172
```

In [17]:
```
print("\nSupport Vector Machine - Polynomial Kernel:")
print("Confusion Matrix: ", confusion_matrix(y_test, y_pred_svm_poly))
print("Accuracy : ", accuracy_score(y_test, y_pred_svm_poly) * 100)
print("Report : ", classification_report(y_test, y_pred_svm_poly))
roc_auc_svm_poly = roc_auc_score(y_test, y_pred_svm_poly)
print("ROC-AUC Score: ", roc_auc_svm_poly)
```

```
Support Vector Machine - Polynomial Kernel:
Confusion Matrix:  [[   105      2]
 [     7 733591]]
Accuracy :  99.99877334896178
Report :                precision    recall  f1-score   support

           0       0.94      0.98      0.96       107
           1       1.00      1.00      1.00    733598

    accuracy                           1.00    733705
   macro avg       0.97      0.99      0.98    733705
weighted avg       1.00      1.00      1.00    733705

ROC-AUC Score:  0.9906494346021032
```

In [18]:
```
print("\nSupport Vector Machine - RBF Kernel:")
print("Confusion Matrix: ", confusion_matrix(y_test, y_pred_svm_rbf))
print("Accuracy : ", accuracy_score(y_test, y_pred_svm_rbf) * 100)
print("Report : ", classification_report(y_test, y_pred_svm_rbf))
roc_auc_svm_rbf = roc_auc_score(y_test, y_pred_svm_rbf)
print("ROC-AUC Score: ", roc_auc_svm_rbf)
```

```
Support Vector Machine - RBF Kernel:
Confusion Matrix:  [[    17     90]
 [     0 733598]]
Accuracy :  99.98773348961775
Report :                precision    recall  f1-score   support

           0       1.00      0.16      0.27       107
           1       1.00      1.00      1.00    733598

    accuracy                           1.00    733705
   macro avg       1.00      0.58      0.64    733705
weighted avg       1.00      1.00      1.00    733705

ROC-AUC Score:  0.5794392523364487
```

In [19]:
```
print("\nSupport Vector Machine - Linear Kernel:")
print("Confusion Matrix: ", confusion_matrix(y_test, Y_pred_L))
print("Accuracy : ", accuracy_score(y_test,Y_pred_L) * 100)
print("Report : ", classification_report(y_test,Y_pred_L))
roc_auc_svm_linear = roc_auc_score(y_test, Y_pred_L)
print("ROC-AUC Score: ", roc_auc_svm_linear)
```

```
Support Vector Machine - Linear Kernel:
Confusion Matrix:  [[    97     10]
 [    22 733576]]
Accuracy :  99.99563857408631
Report :                precision    recall  f1-score   support

           0       0.82      0.91      0.86       107
           1       1.00      1.00      1.00    733598

    accuracy                           1.00    733705
   macro avg       0.91      0.95      0.93    733705
weighted avg       1.00      1.00      1.00    733705

ROC-AUC Score:  0.9532560334490664
```

In [20]:
```
print("\nK-Nearest Neighbors:")
print("Confusion Matrix: ", confusion_matrix(y_test, y_pred_knn))
print("Accuracy : ", accuracy_score(y_test, y_pred_knn) * 100)
print("Report : ", classification_report(y_test, y_pred_knn))
roc_auc_knn = roc_auc_score(y_test, y_pred_knn)
print("ROC-AUC Score: ", roc_auc_knn)
```

```
K-Nearest Neighbors:
Confusion Matrix:  [[   107      0]
 [     3 733595]]
Accuracy :  99.9995911163206
Report :                precision    recall  f1-score   support

           0       0.97      1.00      0.99       107
           1       1.00      1.00      1.00    733598

    accuracy                           1.00    733705
   macro avg       0.99      1.00      0.99    733705
weighted avg       1.00      1.00      1.00    733705

ROC-AUC Score:  0.9999979552834113
```

In [21]:
```
print("\nGaussian Naive Bayes:")
print("Confusion Matrix: ", confusion_matrix(y_test, y_pred_gnb))
print("Accuracy : ", accuracy_score(y_test, y_pred_gnb) * 100)
print("Report : ", classification_report(y_test, y_pred_gnb,zero_division=1
roc_auc_gnb = roc_auc_score(y_test, y_pred_gnb)
print("ROC-AUC Score: ", roc_auc_gnb)
```

```
Gaussian Naive Bayes:
Confusion Matrix:  [[     0    107]
 [     0 733598]]
Accuracy :  99.98541648210112
Report :                precision    recall  f1-score   support

           0       1.00      0.00      0.00       107
           1       1.00      1.00      1.00    733598

    accuracy                           1.00    733705
   macro avg       1.00      0.50      0.50    733705
weighted avg       1.00      1.00      1.00    733705

ROC-AUC Score:  0.5
```

In [22]:
```
print("\nBernoulli Naive Bayes:")
print("Confusion Matrix: ", confusion_matrix(y_test, y_pred_bnb))
print("Accuracy : ", accuracy_score(y_test, y_pred_bnb) * 100)
print("Report : ", classification_report(y_test, y_pred_bnb))
roc_auc_bnb = roc_auc_score(y_test, y_pred_bnb)
print("ROC-AUC Score: ", roc_auc_bnb)
```

```
Bernoulli Naive Bayes:
Confusion Matrix:  [[    91     16]
 [ 20832 712766]]
Accuracy :  97.15853101723445
Report :                precision    recall  f1-score   support

           0       0.00      0.85      0.01       107
           1       1.00      0.97      0.99    733598

    accuracy                           0.97    733705
   macro avg       0.50      0.91      0.50    733705
weighted avg       1.00      0.97      0.99    733705

ROC-AUC Score:  0.9110351328682318
```

In [23]:
```
print("\nAdaBoost with Decision Tree:")
print("Confusion Matrix: ", confusion_matrix(y_test, y_pred_adaboost))
print("Accuracy : ", accuracy_score(y_test, y_pred_adaboost) * 100)
print("Report : ", classification_report(y_test, y_pred_adaboost))
roc_auc_adaboost = roc_auc_score(y_test, y_pred_adaboost)
print("ROC-AUC Score: ", roc_auc_adaboost)
```

```
AdaBoost with Decision Tree:
Confusion Matrix:  [[   107      0]
 [     0 733598]]
Accuracy :  100.0
Report :                precision    recall  f1-score   support

           0       1.00      1.00      1.00       107
           1       1.00      1.00      1.00    733598

    accuracy                           1.00    733705
   macro avg       1.00      1.00      1.00    733705
weighted avg       1.00      1.00      1.00    733705

ROC-AUC Score:  1.0
```

ROC-AUC Curve

In [24]:
```
fpr_tree, tpr_tree, thresholds_tree = roc_curve(y_test, y_pred_tree)
fpr_svm_poly, tpr_svm_poly, thresholds_svm_poly = roc_curve(y_test, y_pred_
fpr_svm_rbf, tpr_svm_rbf, thresholds_svm_rbf = roc_curve(y_test, y_pred_svm
fpr_svm_linear, tpr_svm_linear, thresholds_svm_linear = roc_curve(y_test, Y
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, y_pred_knn)
fpr_gnb, tpr_gnb, thresholds_gnb = roc_curve(y_test, y_pred_gnb)
fpr_bnb, tpr_bnb, thresholds_bnb = roc_curve(y_test, y_pred_bnb)
fpr_adaboost, tpr_adaboost, thresholds_adaboost = roc_curve(y_test, y_pred_
plt.figure(figsize=(10, 8))

plt.plot(fpr_tree, tpr_tree, color='darkorange', lw=2, label='Decision Tree
plt.plot(fpr_svm_poly, tpr_svm_poly, color='green', lw=2, label='SVM - Polyn
plt.plot(fpr_svm_rbf, tpr_svm_rbf, color='blue', lw=2, label='SVM - RBF Kern
plt.plot(fpr_svm_linear, tpr_svm_linear, color='red', lw=2, label='SVM - Lin
plt.plot(fpr_knn, tpr_knn, color='purple', lw=2, label='K-Nearest Neighbors
plt.plot(fpr_gnb, tpr_gnb, color='brown', lw=2, label='Gaussian Naive Bayes
plt.plot(fpr_bnb, tpr_bnb, color='yellow', lw=2, label='Bernoulli Naive Bay
plt.plot(fpr_adaboost, tpr_adaboost, color='pink', lw=2, label='AdaBoost wi

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```