Data Mining and Machine Learning in Cybersecurity
Lab 5
Network Traffic Classification using KNN, Naive Bayes, and SVM

Karthikeyan G
Roll No: CB.SC.P2.CYS24008

February 12, 2025

1 1. Loading and Preprocessing the Dataset

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the dataset
data = pd.read_csv('android_traffic.csv', delimiter=';')

# Drop non-numeric and duplicate columns
if 'name' in data.columns:
    data.drop(columns=['name'], inplace=True)
if 'source_app_packets.1' in data.columns:
    data.drop(columns=['source_app_packets.1'], inplace=True)

# Convert categorical target variable to numeric
data['type'] = data['type'].astype('category').cat.codes

# Split features and target variable
X = data.drop('type', axis=1)
X = X.apply(pd.to_numeric, errors='coerce').fillna(0)
y = data['type']

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# Standardize the dataset
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print("Dataset preprocessed successfully!")
```

Explanation:

- Loads the dataset and removes unnecessary or duplicate columns.
- Converts categorical target variables into numeric values.

```
        name  tcp_packets  dist_port_tcp  external_ips  vulume_bytes  \
0  AntiVirus           36              6             3          3911
1  AntiVirus          117              0             9         23514
2  AntiVirus          196              0             6         24151
3  AntiVirus            6              0             1           889
4  AntiVirus            6              0             1           882

   udp_packets  tcp_urg_packet  source_app_packets  remote_app_packets  \
0            0               0                  39                  33
1            0               0                 128                 107
2            0               0                 205                 214
3            0               0                   7                   6
4            0               0                   7                   6

   source_app_bytes  remote_app_bytes  duracion  avg_local_pkt_rate  \
0              5100              4140       NaN                 NaN
1             26248             24358       NaN                 NaN
2            163887             24867       NaN                 NaN
3               819               975       NaN                 NaN
4               819               968       NaN                 NaN

   avg_remote_pkt_rate  source_app_packets.1  dns_query_times    type
0                  NaN                    39                3  benign
1                  NaN                   128               11  benign
2                  NaN                   205                9  benign
3                  NaN                     7                1  benign
4                  NaN                     7                1  benign
Preprocessing complete! Dataset is ready for training.
```

Figure 1: Preprocessed Network Traffic dataset, containing various network traffic features with labels representing different types of activity.

- Splits the dataset into training (80%) and testing (20%) sets.
- Standardizes the feature values using ‘StandardScaler‘ for better model performance.

2 2. Implementing K-Nearest Neighbors (KNN)

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score

# Initialize and train KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Make predictions
y_pred_knn = knn.predict(X_test)

# Compute accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn) * 100

# Display results
print("KNN Accuracy:", accuracy_knn)
print(classification_report(y_test, y_pred_knn))
```

Explanation:

- Initializes and trains a KNN classifier with ‘k=5‘ neighbors.
- Predicts the class labels of the test set.
- Computes the model’s accuracy and prints a classification report.

```
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.88      0.88       938
           1       0.82      0.84      0.83       631

    accuracy                           0.86      1569
   macro avg       0.86      0.86      0.86      1569
weighted avg       0.86      0.86      0.86      1569

KNN Confusion Matrix:
 [[825 113]
 [103 528]]
KNN Accuracy: 86.23326959847036
```

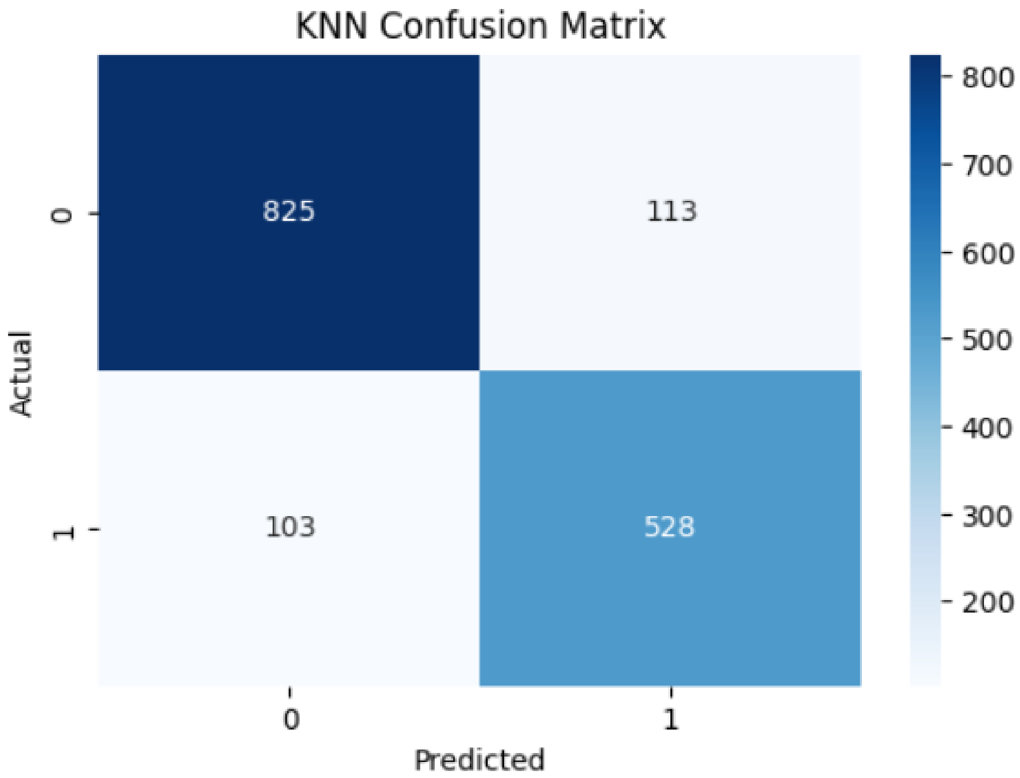

Figure 2: K-Nearest Neighbors (KNN) model predicting network traffic classification based on the majority vote of its closest neighbors.

3 3. Implementing Naïve Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB

# Initialize and train Na ve Bayes model
nb = GaussianNB()
nb.fit(X_train, y_train)

# Make predictions
y_pred_nb = nb.predict(X_test)

# Compute accuracy
accuracy_nb = accuracy_score(y_test, y_pred_nb) * 100

# Display results
print("Na ve Bayes Accuracy:", accuracy_nb)
print(classification_report(y_test, y_pred_nb))
```

Explanation:

- Initializes and trains a Gaussian Naïve Bayes model.
- Predicts the classification of network traffic.
- Computes and prints accuracy and classification metrics.

```
Naive Bayes Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.09      0.16       938
           1       0.42      0.97      0.58       631

    accuracy                           0.44      1569
   macro avg       0.60      0.53      0.37      1569
weighted avg       0.64      0.44      0.33      1569

Naive Bayes Confusion Matrix:
 [[ 81 857]
 [ 22 609]]
Naive Bayes Accuracy: 43.977055449330784
```

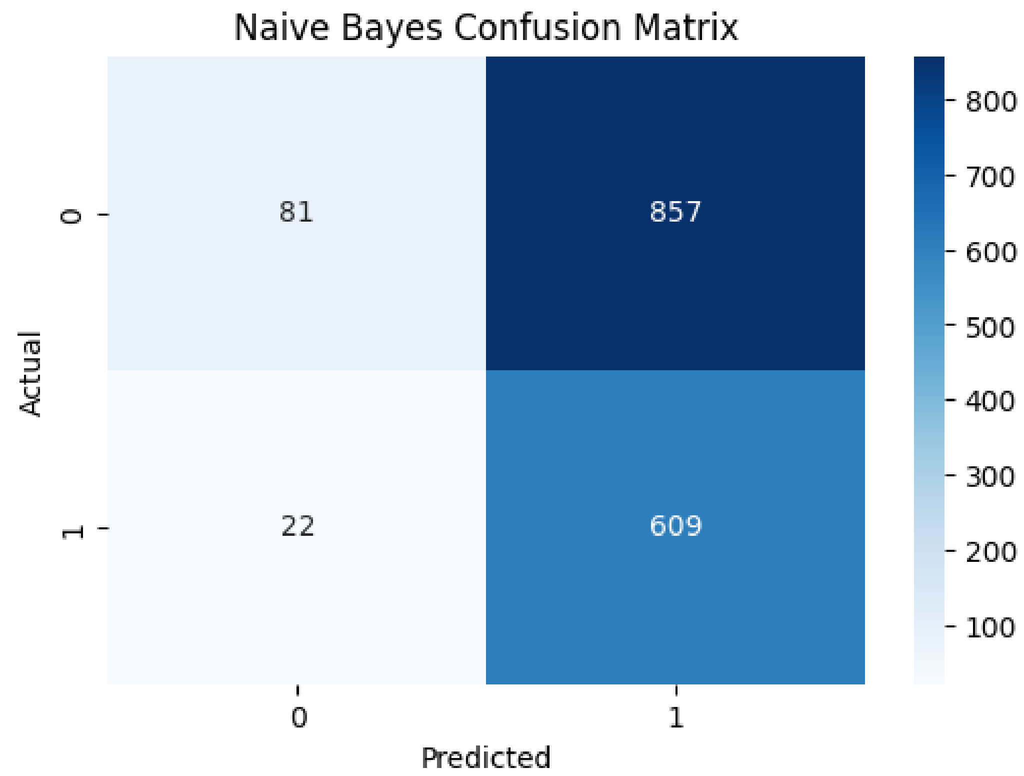

Figure 3: Naïve Bayes classifier applying probability-based classification to network traffic.

4 4. Implementing Support Vector Machine (SVM)

```
from sklearn.svm import SVC

# Initialize and train SVM model
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)

# Make predictions
y_pred_svm = svm.predict(X_test)

# Compute accuracy
accuracy_svm = accuracy_score(y_test, y_pred_svm) * 100

# Display results
print("SVM Accuracy:", accuracy_svm)
print(classification_report(y_test, y_pred_svm))
```

Explanation:

- Initializes and trains an SVM classifier with a linear kernel.
- Finds the optimal decision boundary (hyperplane) to separate classes.
- Predicts and evaluates the accuracy of network traffic classification.

```
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.60      1.00      0.75       938
           1       0.60      0.00      0.01       631

    accuracy                           0.60      1569
   macro avg       0.60      0.50      0.38      1569
weighted avg       0.60      0.60      0.45      1569

SVM Confusion Matrix:
 [[936   2]
 [628   3]]
SVM Accuracy: 59.847036328871894
```

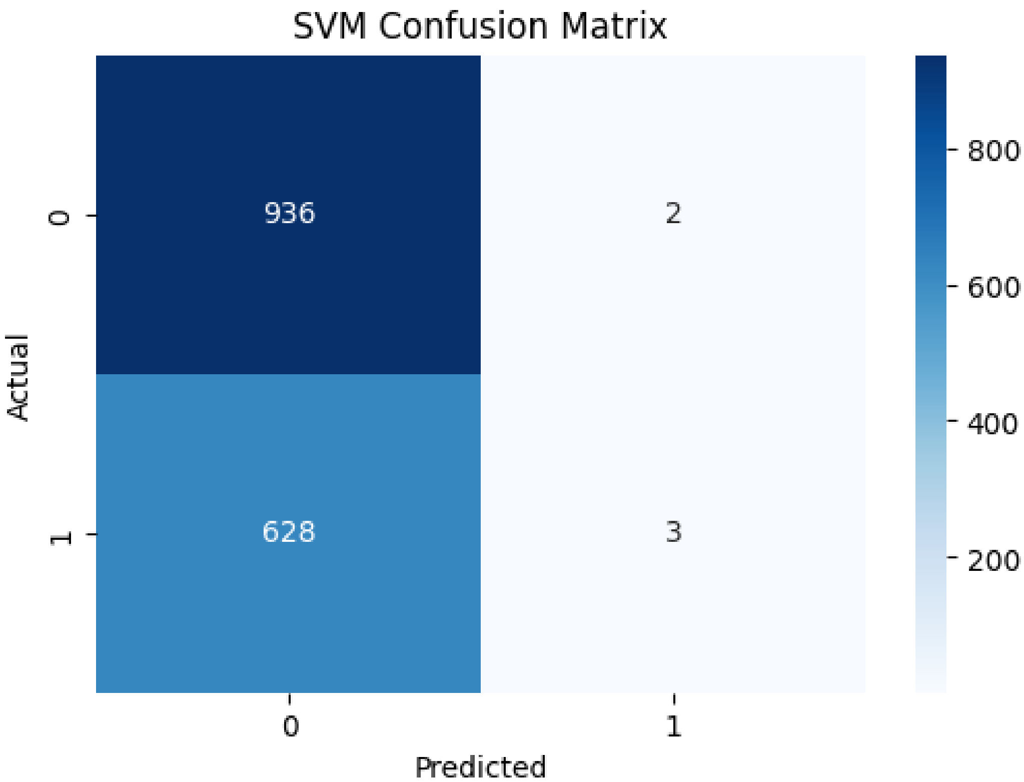

Figure 4: Support Vector Machine (SVM) classifier, optimizing a decision boundary for accurate network traffic classification.

5 5. Performance Comparison of Classifiers

```
import numpy as np
import matplotlib.pyplot as plt

# Prepare data
labels = ['KNN', 'Na ve Bayes', 'SVM']
accuracies = [accuracy_knn, accuracy_nb, accuracy_svm]

x = np.arange(len(labels))
width = 0.4

fig, ax = plt.subplots()
bars = ax.bar(x, accuracies, width, color=['blue', 'green', 'red'])

ax.set_ylabel('Accuracy (%)')
ax.set_title('Comparison of Classification Accuracy')
ax.set_xticks(x)
ax.set_xticklabels(labels)

# Annotate bars with values
for bar in bars:
    height = bar.get_height()
    ax.annotate(f'{height:.2f}%', xy=(bar.get_x() + bar.get_width() / 2, height)
        ,
                xytext=(0, 3), textcoords="offset points",
                ha='center', va='bottom')

plt.show()
```

Explanation:

- Compares the accuracy of KNN, Naïve Bayes, and SVM classifiers using a bar chart.
- Highlights the model that performs best on network traffic classification.

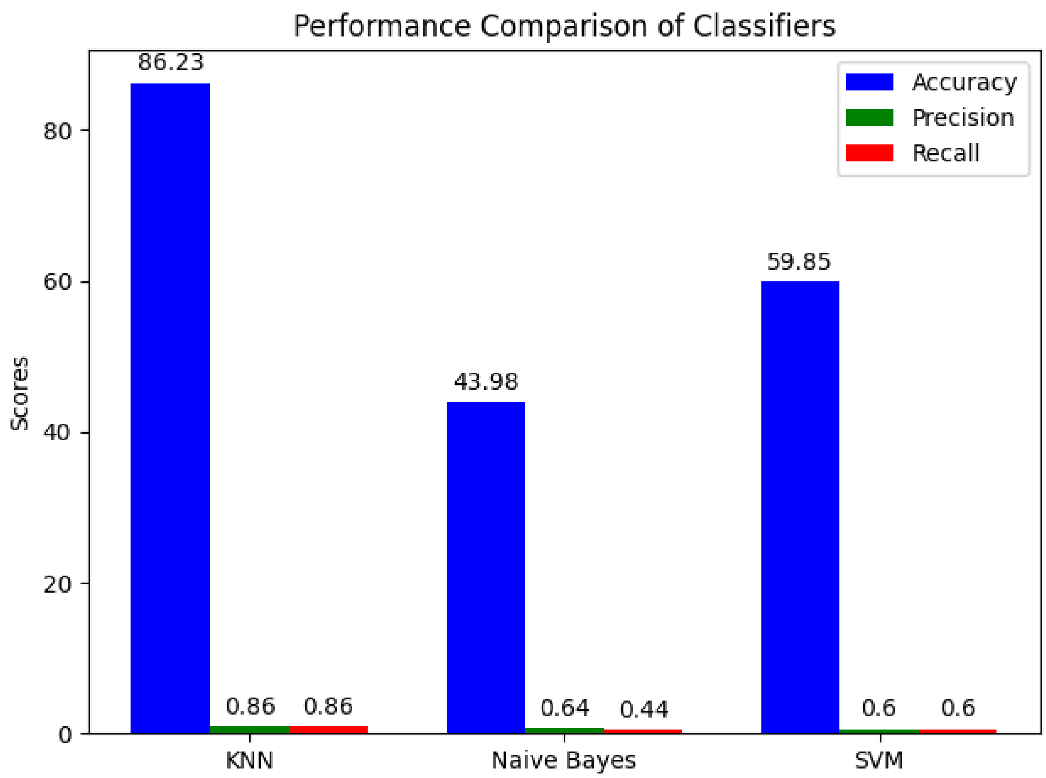

Figure 5: Comparison of accuracy for KNN, Naïve Bayes, and SVM classifiers in network traffic classification.