Report for the AI Lab Project

Classifying Cyber-Attacks in Network Traffic

Members: Taha Abbas Ali
Date: 30 April 2023
Rollnos: P20-0119
Section: A

Executive Summary

The dramatic growth of computer networks and the increase in the number of applications running on top of them have made network security a critical concern for businesses and organizations. In this context, detecting vulnerabilities in network systems has become increasingly important to prevent cyber attacks that could have negative impacts on the economy.

The objective of this project is to apply various classification and clustering algorithms to classify cyber-attacks in network traffic as accurately and in real time as possible. Through this approach, the project aims to enhance network security by identifying potential vulnerabilities and enabling prompt action to mitigate security threats.

The project's results demonstrate that these algorithms can accurately classify network traffic and detect potential cyber threats in real time. The implementation of these algorithms will help organizations to improve their network security by identifying and responding to cyber-attacks more quickly and effectively.

In conclusion, the project highlights the importance of network security and the need for accurate and timely detection of vulnerabilities in the network system. The project's approach of using classification and clustering algorithms to classify cyber-attacks in network traffic provides an effective and efficient solution to enhance network security.

Introduction

The goal of this research is to use multiple classification and clustering techniques to accurately and in real-time categorize cyber-attacks in network traffic. The project's strategy attempts to improve network security by identifying potential vulnerabilities and enabling quick action to minimize security risks.

The research analyses network traffic data using classification and clustering techniques to detect patterns linked with various forms of cyber-attacks. The project employs multilayer perceptron, decision trees, k closest neighbor, and k-means clustering algorithms.

Data-preprocessing

I have focused on the use of data preprocessing techniques in exploratory data analysis (EDA). Data preprocessing is a critical step in data analysis that involves cleaning, transforming, and preparing raw data for further analysis. I have highlighted various data preprocessing techniques, such as *identifying and handling columns containing only zero values, checking for null values, computing value counts, obtaining dataset information, plotting data distributions, and cleaning the data*. I have done various data preprocessing techniques for this real-world dataset. The ipynb notebook demonstrates how these techniques are used to identify and handle *missing data, outliers, and inconsistencies in the data*. Additionally, I have explored how data visualization techniques can be used to gain insights into the data and highlight patterns and trends that might be missed otherwise.

Feature Engineering

In the notebook , the features are already pre-selected through correlation analysis and Profiling report, no additional feature engineering is performed.

Use of the given classification and clustering algorithms

In the notebook, I have used some classification machine learning algorithms such as decision trees, k-nearest neighbors, and deep neural networks like Multilayer Perceptron. Calculated different metrics for each such as accuracy, precision, recall, f1 to measure the output, with that so I have metric output with bar and scatter plots.

Comparison and Performance Evaluation (plots, tables etc.)

For better understanding and comparison I have attached the screenshots of our notebook sequentially.

KNN

In [87]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

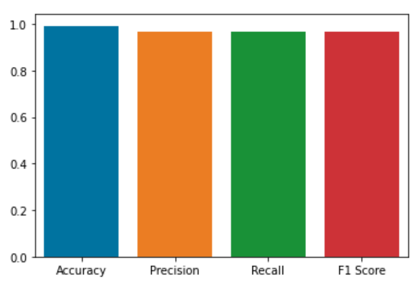
Out[87]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [88]:
```
y_pred = knn.predict(X_test)
```

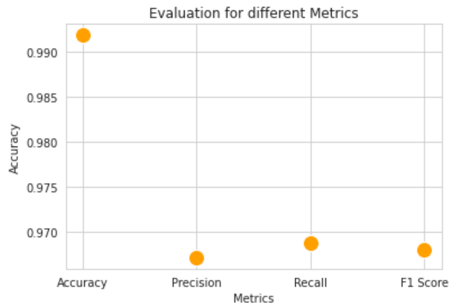
In [94]:
```
print("KNN Object different stats!")
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy: {:.2f}'.format(accuracy))
print('Precision: {:.2f}'.format(precision))
print('Recall: {:.2f}'.format(recall))
print('F1-score: {:.2f}'.format(f1))

sns.barplot(x=['Accuracy', 'Precision', 'Recall', 'F1 Score'], y=[accuracy, precision, recall, f1])
```

```
KNN Object different stats!
Accuracy: 0.99
Precision: 0.97
Recall: 0.97
F1-score: 0.97
```

Out[94]: <AxesSubplot: >



In [96]:
```
sns.set_style('whitegrid')
sns.scatterplot(x=['Accuracy', 'Precision', 'Recall', 'F1 Score'], y=[accuracy, precision, recall, f1], s=200, colo
plt.xlabel('Metrics')
plt.ylabel('Accuracy')
plt.title('Evaluation for different Metrics')
plt.show()
```



Decision Treee

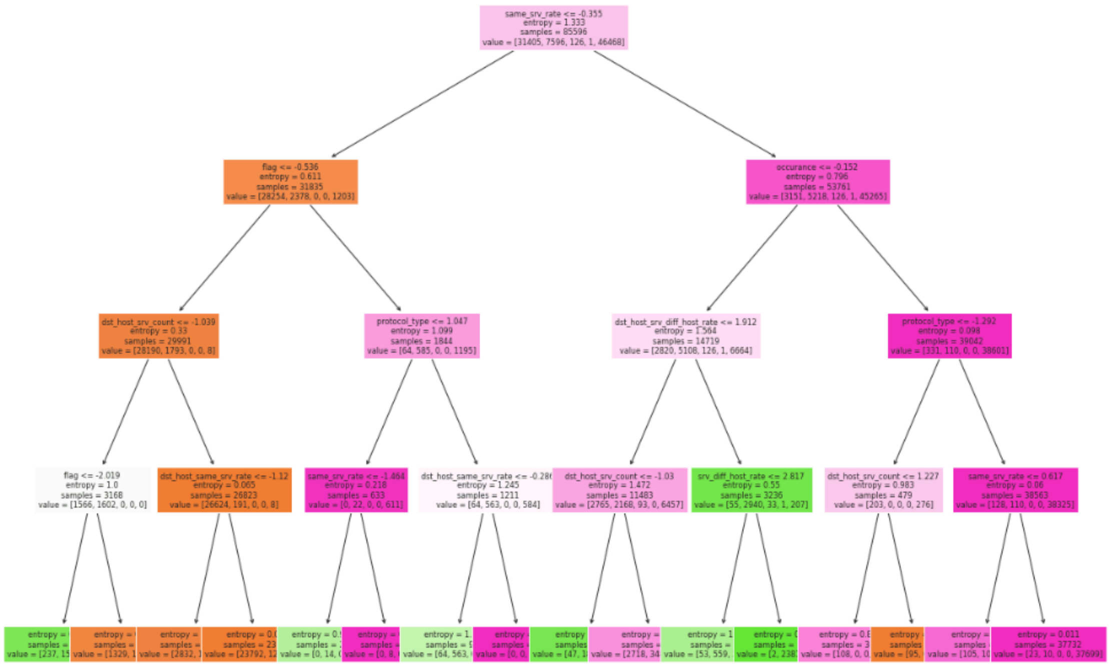
In [97]:
```
DTC_Model_entropy = DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=42)
C_Model_entropy.fit(X_train, y_train)

y_pred_entropy = DTC_Model_entropy.predict(X_test)
```

In [98]:
```
print("Decision Tree Object with different stats!")
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy: {:.2f}'.format(accuracy))
print('Precision: {:.2f}'.format(precision))
print('Recall: {:.2f}'.format(recall))
print('F1-score: {:.2f}'.format(f1))
```

```
Decision Tree Object with different stats!
Accuracy: 0.99
Precision: 0.97
Recall: 0.97
F1-score: 0.97
```

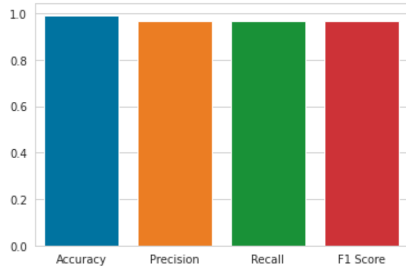
In [100]:

```
plt.figure(figsize=(20, 15))
DTC_tree_entropy = tree.plot_tree(DTC_Model_entropy, filled=True, feature_names=selected_features, fontsize=8)
plt.show()
```

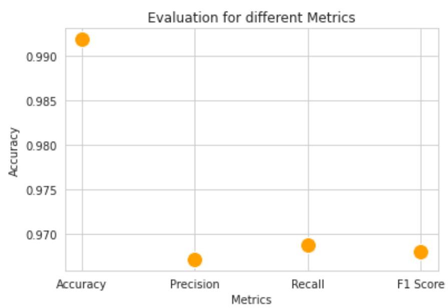


```
In [101]: sns.barplot(x=['Accuracy', 'Precision', 'Recall', 'F1 Score'], y=[accuracy, precision, recall, f1])
```

```
Out[101]: <AxesSubplot: >
```




```
In [102]: sns.set_style('whitegrid')
sns.scatterplot(x=['Accuracy', 'Precision', 'Recall', 'F1 Score'], y=[accuracy, precision, recall, f1], s=200, colo
plt.xlabel('Metrics')
plt.ylabel('Accuracy')
plt.title('Evaluation for different Metrics')
plt.show()
```



MLP- Multi layer Perceptron

```
In [103]: mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='adam', max_iter=300, learning_rate_init=0

mlp.fit(X_train, y_train)

y_pred = mlp.predict(X_test)
```

```
Out[103]: MLPClassifier
MLPClassifier(max_iter=300)
```

```
In [106]: print("MLP Object with different stats!")
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy: {:.2f}'.format(accuracy))
print('Precision: {:.2f}'.format(precision))
print('Recall: {:.2f}'.format(recall))
print('F1-score: {:.2f}'.format(f1))
```

```
MLP Object with different stats!
Accuracy: 0.99
Precision: 0.98
Recall: 0.95
F1-score: 0.97
```

In [110]:
```python
mlp = MLPClassifier(hidden_layer_sizes=(100,50), activation='relu', solver='sgd', max_iter=500, learning_rate_init=(

mlp.fit(X_train, y_train)
```

Out[110]:

```
MLPClassifier
MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=500, solver='sgd')
```

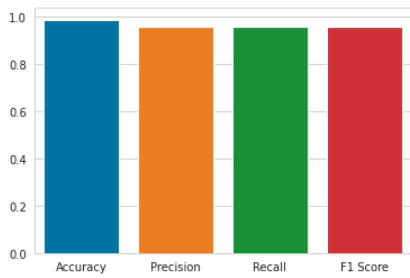
In [112]:
```python
y_pred = mlp.predict(X_test)
```

In [114]:
```python
print("MLP Object with different stats after Hyper tunning!")
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy: {:.2f}'.format(accuracy))
print('Precision: {:.2f}'.format(precision))
print('Recall: {:.2f}'.format(recall))
print('F1-score: {:.2f}'.format(f1))
```

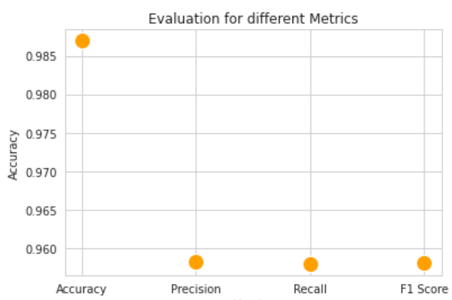
```
MLP Object with different stats after Hyper tunning!
Accuracy: 0.99
Precision: 0.96
Recall: 0.96
F1-score: 0.96
```

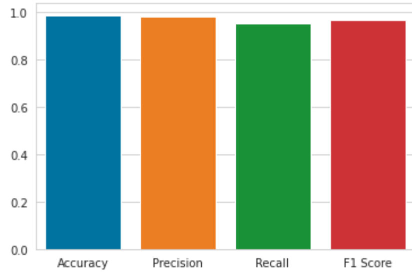
In [115]:
```python
sns.barplot(x=['Accuracy', 'Precision', 'Recall', 'F1 Score'], y=[accuracy, precision, recall, f1])
```

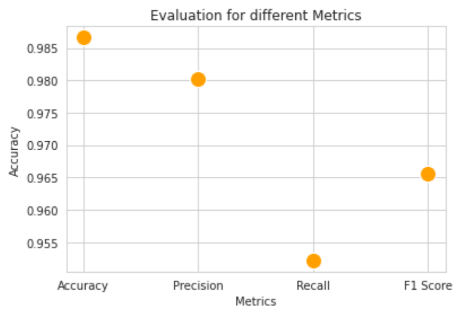
Out[115]: <AxesSubplot: >



In [116]:
```python
sns.set_style('whitegrid')
sns.scatterplot(x=['Accuracy', 'Precision', 'Recall', 'F1 Score'], y=[accuracy, precision, recall, f1], s=200, colo
plt.xlabel('Metrics')
plt.ylabel('Accuracy')
plt.title('Evaluation for different Metrics')
plt.show()
```



In [107]:
```
sns.barplot(x=['Accuracy', 'Precision', 'Recall', 'F1 Score'], y=[accuracy, precision, recall, f1])
```
Out[107]:
```
<AxesSubplot: >
```


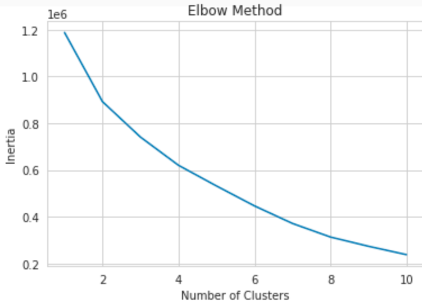
In [108]:
```
sns.set_style('whitegrid')
sns.scatterplot(x=['Accuracy', 'Precision', 'Recall', 'F1 Score'], y=[accuracy, precision, recall, f1], s=200, colo
plt.xlabel('Metrics')
plt.ylabel('Accuracy')
plt.title('Evaluation for different Metrics')
plt.show()
```


In [ ]:

K-means Clustering

In [117]:
```
X_train.info()
```
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 85596 entries, 65182 to 121958
Data columns (total 15 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   same_srv_rate               85596 non-null  float64
 1   dst_host_srv_count          85596 non-null  float64
 2   dst_host_same_srv_rate      85596 non-null  float64
 3   logged_in                   85596 non-null  float64
 4   flag                        85596 non-null  float64
 5   occurance                   85596 non-null  float64
 6   protocol_type               85596 non-null  float64
 7   srv_diff_host_rate          85596 non-null  float64
 8   is_guest_login              85596 non-null  float64
 9   hot                         85596 non-null  float64
 10  root_shell                  85596 non-null  float64
 11  num_failed_logins           85596 non-null  float64
 12  num_root                    85596 non-null  float64
 13  num_compromised             85596 non-null  float64
 14  dst_host_srv_diff_host_rate 85596 non-null  float64
dtypes: float64(15)
memory usage: 10.4 MB
```

In [118]:
```
# Use elbow method to find optimal number of clusters
SSE = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, n_init=20, random_state=42)
    kmeans.fit(X_train)
    SSE.append(kmeans.inertia_)
plt.plot(range(1, 11), SSE)
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.show()
```



In [ ]:

In [119]:
```
# Based on elbow method, choose optimal number of clusters (e.g., 5)
kmeans = KMeans(n_clusters=2, n_init=20, random_state=42)
kmeans.fit(X_train)
```

Out[119]:
```
KMeans
KMeans(n_clusters=2, n_init=20, random_state=42)
```

In [120]:
```
# Getting the cluster labels
cluster_labels = kmeans.labels_

# Printing the cluster labels
print(cluster_labels)
```

```
[1 0 1 ... 1 0 0]
```

In [121]:
```
unique_values = np.unique(cluster_labels)
print(unique_values)
```

```
[0 1]
```

In [122]:
```
freq = np.count_nonzero(cluster_labels == 0)
freq
```

Out[122]: 38315

```
In [123]: freq = np.count_nonzero(cluster_labels == 1)
          freq
```

```
Out[123]: 47281
```

```
In [124]: # Assume kmeans is already fit and has transformed the data
          cluster_centers = kmeans.cluster_centers_
          X_transformed = kmeans.transform(X_train)

          # Create scatter plot using transformed data
          sns.scatterplot(x=X_transformed[:, 0], y=X_transformed[:, 1], hue=kmeans.labels_, style=kmeans.labels_,
                          palette=['red', 'blue'], data=X_train)
          plt.scatter(cluster_centers[:, 6], cluster_centers[:, 7], marker='o', s=20, linewidth=3, color='black')
          plt.title('Cluster Plot for Network Security')
          plt.xlabel('Occurance')
          plt.ylabel('Protocol type')
          plt.show()
```


```
In [125]: X_transformed
```

```
Out[125]: array([[4.46240256, 1.15425457],
                 [0.91518479, 4.16260748],
                 [4.51884785, 1.05607499],
                 ...,
                 [4.43049888, 1.30163567],
                 [4.11240118, 5.65610078],
                 [1.41482751, 3.73815965]])
```

```
In [ ]:
```

Conclusions

It should be considered that the critical component of feature selection has been done well quite enough, and the model has been trained under good conditions, with an accuracy exceeding 96 percent. However, I should test the model with a different subset of features to see how it compares.