SGanguli_Assignment3

May 27, 2024

1 IoT dataset for Intrusion Detection Systems (IDS)

1.1 About Dataset

1.1.1 Source:

Kaggle

1.1.2 Description:

BoTNeTIoT-L01 is a data set integrated all the IoT devices data file from the detection_of_IoT_botnet_attacks_N_BaIoT (BoTNeTIoT) data set. This new version reduced the redundancy of the original dataset by choosing the features of 10 seconds time window only. In the dataset class label, 0 stands for attacks, and 1 stands for normal samples.

1.1.3 Data set details:

The BoTNeTIoT-L01, the most recent dataset, contains nine IoT devices traffic sniffed using Wireshark in a local network using a central switch. It includes two Botnet attacks (Mirai and Gafgyt). The dataset contains twenty-three statistically engineered features extracted from the .pcap files. Seven statistical measures were computed (mean, variance, count, magnitude, radius, covariance, correlation coefficient) over the time window of 10 sec with decay factor equals 0.1. The decay factor value is used in the dataset as well as in our papers below [2],[3],[4], and [5] to refer to its corresponding time window as L0.1. Four features were extracted from the .pcap: packet count, jitter, size of outbound packets only, and outbound and inbound packets together. For each of these four features, three or more statistical measures were computed, resulting in twenty-three features.

1.1.4 About the features

Based on the column names provided, it appears that the dataset includes various features extracted from network traffic or signals, potentially for the purpose of intrusion detection or anomaly detection in an IoT environment. Here's a likely interpretation of what these columns might represent:

1.1.5 MI_dir_L0.1_weight / MI_dir_L0.1_mean / MI_dir_L0.1_variance:

MI: Mutual Information

dir: Directional (indicating that this metric is related to the direction of traffic or data flow)

L0.1: Likely a parameter or level setting, perhaps indicating a specific layer or threshold weight / mean / variance: Statistical measures related to Mutual Information at the specified level and direction

###H_L0.1_weight / H_L0.1_mean / H_L0.1_variance:

H: Entropy

L0.1: Specific level or parameter

weight / mean / variance: Statistical measures related to Entropy at the specified level

1.1.6 HH_L0.1_weight / HH_L0.1_mean / HH_L0.1_std / HH_L0.1_magnitude:

HH: Possibly High-High Entropy or a higher-order statistical measure involving Entropy

L0.1: Specific level or parameter

weight / mean / std (standard deviation) / magnitude: Statistical measures related to this higher-order entropy

1.1.7 HpHp_L0.1_mean / HpHp_L0.1_std / HpHp_L0.1_magnitude / HpHp_L0.1_radius / HpHp_L0.1_covariance / HpHp_L0.1_pcc:

HpHp: This could refer to a second-order statistical measure or a derived metric from entropy or signal processing

L0.1: Specific level or parameter

mean / std (standard deviation) / magnitude / radius / covariance / pcc (Pearson correlation coefficient): Various statistical and mathematical properties of this derived metric

General Interpretation:

MI (Mutual Information): Measures the mutual dependence between two variables. In the context of network traffic, it could indicate the amount of information shared between different traffic patterns or data streams.

Entropy (H): Measures the uncertainty or randomness in the data. High entropy can indicate more randomness, which might be associated with normal behavior, while low entropy might indicate predictable patterns, possibly due to malicious activities.

Higher-order statistical measures (HH, HpHp): These could involve combinations or transformations of the basic entropy measures, providing more nuanced information about the data's structure or anomalies.

Statistical terms (mean, variance, std, magnitude, radius, covariance, pcc): These terms refer to common statistical properties used to describe the distribution, spread, and relationships within the data.

[2]:

```
import pandas as pd
```

[3]:

```
data = pd.read_csv('data/BoTNeTIoT-L01-v2.csv')
data.head()
```

```
[3]:    MI_dir_L0.1_weight  MI_dir_L0.1_mean  MI_dir_L0.1_variance  H_L0.1_weight  \
     0            1.000000         98.000000          0.000000e+00       1.000000
     1            1.931640         98.000000          1.818989e-12       1.931640
     2            2.904273         86.981750          2.311822e+02       2.904273
     3            3.902546         83.655268          2.040614e+02       3.902546
     4            4.902545         81.685828          1.775746e+02       4.902545

        H_L0.1_mean  H_L0.1_variance  HH_L0.1_weight  HH_L0.1_mean   HH_L0.1_std  \
     0    98.000000     0.000000e+00         1.00000          98.0  0.000000e+00
     1    98.000000     1.818989e-12         1.93164          98.0  1.348699e-06
     2    86.981750     2.311822e+02         1.00000          66.0  0.000000e+00
     3    83.655268     2.040614e+02         1.00000          74.0  0.000000e+00
     4    81.685828     1.775746e+02         2.00000          74.0  9.536743e-07

        HH_L0.1_magnitude  …  HpHp_L0.1_mean  HpHp_L0.1_std  HpHp_L0.1_magnitude  \
     0          98.000000  …            98.0       0.000000            98.000000
     1         138.592929  …            98.0       0.000001           138.592929
     2         114.856432  …            66.0       0.000000           114.856432
     3          74.000000  …            74.0       0.000000            74.000000
     4          74.000000  …            74.0       0.000000            74.000000

        HpHp_L0.1_radius  HpHp_L0.1_covariance  HpHp_L0.1_pcc       Device_Name  \
     0      0.000000e+00                   0.0            0.0  Danmini_Doorbell
     1      1.818989e-12                   0.0            0.0  Danmini_Doorbell
     2      0.000000e+00                   0.0            0.0  Danmini_Doorbell
     3      0.000000e+00                   0.0            0.0  Danmini_Doorbell
     4      0.000000e+00                   0.0            0.0  Danmini_Doorbell

        Attack  Attack_subType  label
     0  gafgyt           combo      0
     1  gafgyt           combo      0
     2  gafgyt           combo      0
     3  gafgyt           combo      0
     4  gafgyt           combo      0

     [5 rows x 27 columns]
```

1.2 Exploratory Data Analysis

1.3 Dropping columns

We drop some of the columns so that the number of features is 20 or less. Given that one of the features is the target, we will need the second element of the 'data.shape' to be 21 or less

```
[4]: data.columns
```

```
[4]: Index(['MI_dir_L0.1_weight', 'MI_dir_L0.1_mean', 'MI_dir_L0.1_variance',
            'H_L0.1_weight', 'H_L0.1_mean', 'H_L0.1_variance', 'HH_L0.1_weight',
            'HH_L0.1_mean', 'HH_L0.1_std', 'HH_L0.1_magnitude', 'HH_L0.1_radius',
            'HH_L0.1_covariance', 'HH_L0.1_pcc', 'HH_jit_L0.1_weight',
            'HH_jit_L0.1_mean', 'HH_jit_L0.1_variance', 'HpHp_L0.1_weight',
            'HpHp_L0.1_mean', 'HpHp_L0.1_std', 'HpHp_L0.1_magnitude',
            'HpHp_L0.1_radius', 'HpHp_L0.1_covariance', 'HpHp_L0.1_pcc',
            'Device_Name', 'Attack', 'Attack_subType', 'label'],
           dtype='object')
```

[5]:

```
# Drop the specified columns
columns_to_drop = [
       'H_L0.1_weight', 'H_L0.1_mean', 'H_L0.1_variance', 'HH_L0.1_weight',
       'HH_L0.1_mean', 'HH_L0.1_std', 'HH_L0.1_magnitude', 'HH_L0.1_radius',
       'HH_L0.1_covariance', 'HH_L0.1_pcc', 'HH_jit_L0.1_weight',
       'HH_jit_L0.1_mean', 'HH_jit_L0.1_variance', 'HpHp_L0.1_weight',
       'HpHp_L0.1_mean', 'HpHp_L0.1_std', 'HpHp_L0.1_magnitude',
       'HpHp_L0.1_radius', 'HpHp_L0.1_covariance', 'HpHp_L0.1_pcc'
]
print(f"Dropping columns: {columns_to_drop}")
data = data.drop(columns=columns_to_drop)
```

```
Dropping columns: ['H_L0.1_weight', 'H_L0.1_mean', 'H_L0.1_variance',
'HH_L0.1_weight', 'HH_L0.1_mean', 'HH_L0.1_std', 'HH_L0.1_magnitude',
'HH_L0.1_radius', 'HH_L0.1_covariance', 'HH_L0.1_pcc', 'HH_jit_L0.1_weight',
'HH_jit_L0.1_mean', 'HH_jit_L0.1_variance', 'HpHp_L0.1_weight',
'HpHp_L0.1_mean', 'HpHp_L0.1_std', 'HpHp_L0.1_magnitude', 'HpHp_L0.1_radius',
'HpHp_L0.1_covariance', 'HpHp_L0.1_pcc']
```

1.4 Reducing the dataset

If we are using the full dataset then we are getting an accuracy of 1 for the decision tree used later. This is not enable us to see the comparative effect of random forest.

[6]:

```
# Shape of data before reduction
print("Shape of data before reduction")
print(data.shape)

# Reduce the dataset
print("Reducing the dataset to 1% of the size...")
data_reduced = data.sample(frac=0.01, random_state=42)

# Shape of data after reduction
print("# Shape of data after reduction")
print(data_reduced.shape)

data = data_reduced
```

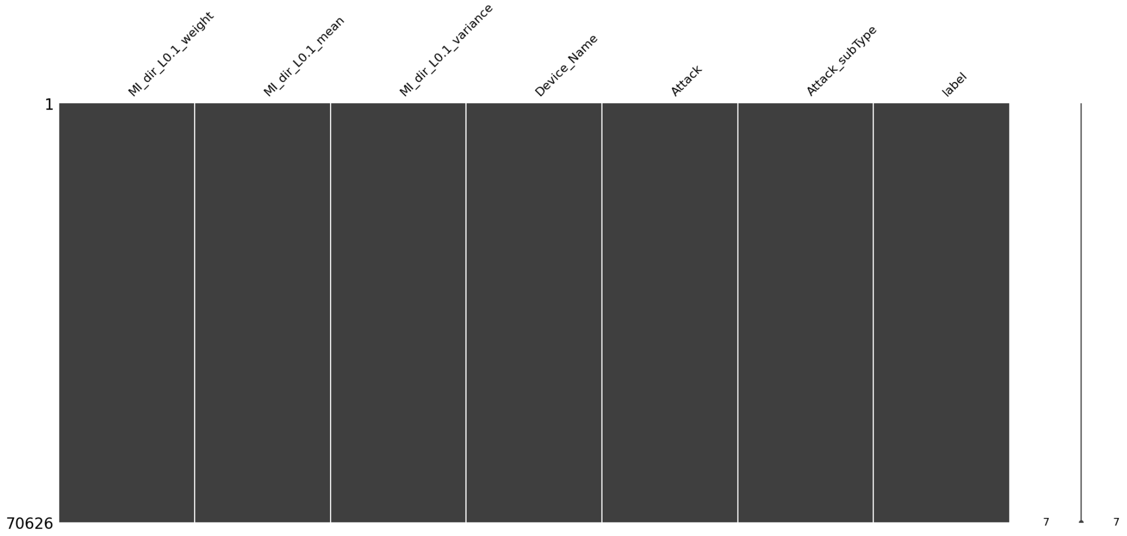
```
Shape of data before reduction
(7062606, 7)
Reducing the dataset to 1% of the size…
```

```
# Shape of data after reduction
(70626, 7)
```

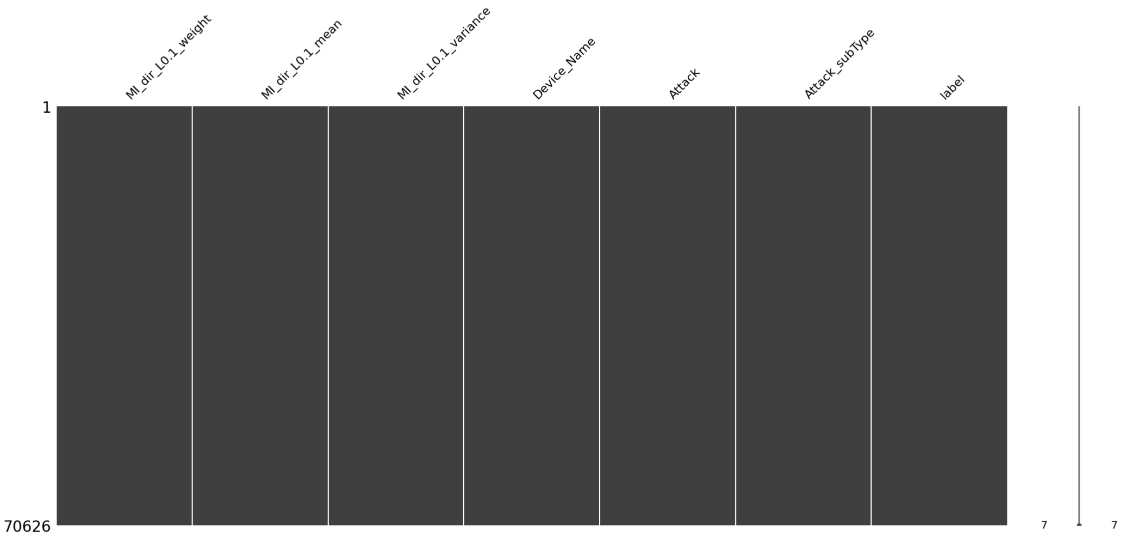
1.5 Check for missing data

```
[7]: import missingno as msno
import matplotlib.pyplot as plt

# Check for missing data using missingno
print("Checking for missing data before cleanup...")
msno.matrix(data)
plt.show()

# Drop rows with missing data
print("Dropping rows with missing data...")
data_clean = data.dropna()

print("Checking for missing data after cleanup...")
msno.matrix(data_clean)
plt.show()

data = data_clean
```

```
Checking for missing data before cleanup…
```

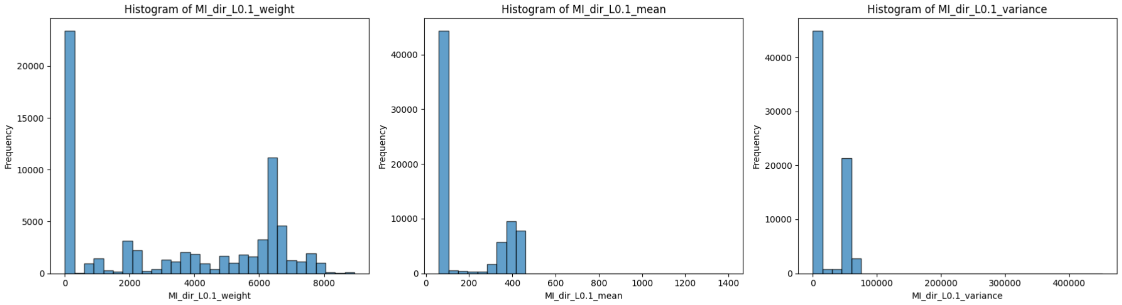


```
Dropping rows with missing data…
Checking for missing data after cleanup…
```



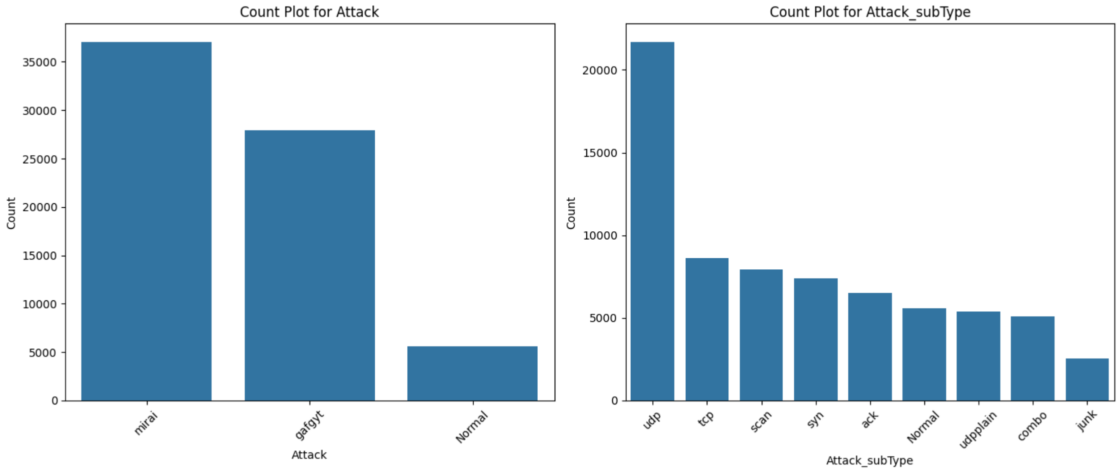
[8]:

```
# Check if the columns exist in the dataset
columns_to_plot = ['MI_dir_L0.1_weight', 'MI_dir_L0.1_mean', 'MI_dir_L0.
 ↪1_variance']
for column in columns_to_plot:
    if column not in data.columns:
        raise KeyError(f"Column {column} is not present in the dataset.")

# Create histograms for the specified columns
plt.figure(figsize=(18, 5))

for i, column in enumerate(columns_to_plot, 1):
    plt.subplot(1, 3, i)
    plt.hist(data[column].dropna(), bins=30, edgecolor='k', alpha=0.7)
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



[10]:

```
import seaborn as sns

# Check if the columns exist in the dataset
columns_to_plot = ['Attack', 'Attack_subType']
for column in columns_to_plot:
    if column not in data.columns:
        raise KeyError(f"Column {column} is not present in the dataset.")

# Create count plots for 'Attack' and 'Attack_subType'
plt.figure(figsize=(14, 6))

# Count plot for 'Attack'
plt.subplot(1, 2, 1)
sns.countplot(data=data, x='Attack', order=data['Attack'].value_counts().index)
plt.title('Count Plot for Attack')
plt.xlabel('Attack')
plt.ylabel('Count')
plt.xticks(rotation=45)

# Count plot for 'Attack_subType'
plt.subplot(1, 2, 2)
sns.countplot(data=data, x='Attack_subType', order=data['Attack_subType'].
 ↪value_counts().index)
plt.title('Count Plot for Attack_subType')
plt.xlabel('Attack_subType')
plt.ylabel('Count')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```



1.5.1 Select Target and Features

[35]:

```python
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.metrics import classification_report
from sklearn.compose import ColumnTransformer


# Drop the columns "Attack" and "Attack_subType" from the features
print("Dropping columns 'Attack' and 'Attack_subType' from the features...")
X = data.drop(columns=['Attack', 'Attack_subType'])
Attack_name = data['Attack']
Attack_subType = data['Attack_subType']

# Encode target labels
print("Encoding target labels...")
Attack_name_encoder = LabelEncoder()
Attack_subType_encoder = LabelEncoder()
y_Attack_name = Attack_name_encoder.fit_transform(Attack_name)
y_Attack_subType = Attack_subType_encoder.fit_transform(Attack_subType)

# Combine the two target labels into a DataFrame
y = pd.DataFrame({'Attack': y_Attack_name, 'Attack_subType': y_Attack_subType})

# Identify categorical features
categorical_features = X.select_dtypes(include=['object']).columns

# Preprocess the features (encode categorical features and standardize␣
 ↪numerical features)
print("Preprocessing the features...")
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
        ('num', StandardScaler(), X.select_dtypes(include=['float64', 'int64']).
 ↪columns)
    ])

X_processed = preprocessor.fit_transform(X)

X = X_processed
```

```
Dropping columns 'Attack' and 'Attack_subType' from the features…
Encoding target labels…
Preprocessing the features…
```

1.6 Classifiers

[37]:
```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
print("Splitting data 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 features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
Splitting data into training and testing sets…
```

1.7 Decision Tree

[38]:
```
# Initialize and train the MultiOutputClassifier with DecisionTreeClassifier
print("Initializing and training the MultiOutputClassifier...")
base_classifier = DecisionTreeClassifier(random_state=42)
multi_target_classifier = MultiOutputClassifier(base_classifier, n_jobs=-1)
multi_target_classifier.fit(X_train, y_train)

# Make predictions
print("Making predictions...")
y_pred = multi_target_classifier.predict(X_test)

# Decode the predictions
y_pred_df = pd.DataFrame(y_pred, columns=['Attack', 'Attack_subType'])
y_pred_df['Attack'] = Attack_name_encoder.inverse_transform(y_pred_df['Attack'])
y_pred_df['Attack_subType'] = Attack_subType_encoder.
 ↪inverse_transform(y_pred_df['Attack_subType'])

y_test_decoded = y_test.copy()
y_test_decoded['Attack'] = Attack_name_encoder.
 ↪inverse_transform(y_test['Attack'])
y_test_decoded['Attack_subType'] = Attack_subType_encoder.
 ↪inverse_transform(y_test['Attack_subType'])

# Evaluate the model
print("Classification Report for Attack Name:")
print(classification_report(y_test_decoded['Attack'], y_pred_df['Attack']))

print("Classification Report for Attack SubType:")
print(classification_report(y_test_decoded['Attack_subType'],␣
 ↪y_pred_df['Attack_subType']))
```

```
print("Done.")
```

```
Initializing and training the MultiOutputClassifier…
Making predictions…
Classification Report for Attack Name:
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00      1114
      gafgyt       1.00      1.00      1.00      5548
       mirai       1.00      1.00      1.00      7464

    accuracy                           1.00     14126
   macro avg       1.00      1.00      1.00     14126
weighted avg       1.00      1.00      1.00     14126

Classification Report for Attack SubType:
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00      1114
         ack       1.00      1.00      1.00      1308
       combo       1.00      1.00      1.00       988
        junk       0.99      0.99      0.99       493
        scan       1.00      1.00      1.00      1585
         syn       1.00      1.00      1.00      1505
         tcp       1.00      0.00      0.00      1687
         udp       0.72      1.00      0.84      4412
    udpplain       0.99      0.98      0.98      1034

    accuracy                           0.88     14126
   macro avg       0.97      0.89      0.87     14126
weighted avg       0.91      0.88      0.83     14126

Done.
```

[41]:

```
from sklearn.ensemble import RandomForestClassifier

# Initialize and train the MultiOutputClassifier with DecisionTreeClassifier
print("Initializing and training the MultiOutputClassifier...")
base_classifier = RandomForestClassifier(
    random_state=42,
    n_jobs=-1,
    verbose=3
)
multi_target_classifier = MultiOutputClassifier(base_classifier, n_jobs=-1)
multi_target_classifier.fit(X_train, y_train)

# Make predictions
```

```python
print("Making predictions...")
y_pred = multi_target_classifier.predict(X_test)

# Decode the predictions
y_pred_df = pd.DataFrame(y_pred, columns=['Attack', 'Attack_subType'])
y_pred_df['Attack'] = Attack_name_encoder.inverse_transform(y_pred_df['Attack'])
y_pred_df['Attack_subType'] = Attack_subType_encoder.
 ↪inverse_transform(y_pred_df['Attack_subType'])

y_test_decoded = y_test.copy()
y_test_decoded['Attack'] = Attack_name_encoder.
 ↪inverse_transform(y_test['Attack'])
y_test_decoded['Attack_subType'] = Attack_subType_encoder.
 ↪inverse_transform(y_test['Attack_subType'])

# Evaluate the model
print("Classification Report for Attack Name:")
print(classification_report(y_test_decoded['Attack'], y_pred_df['Attack']))

print("Classification Report for Attack SubType:")
print(classification_report(y_test_decoded['Attack_subType'],␣
 ↪y_pred_df['Attack_subType']))

print("Done.")
```

```
Initializing and training the MultiOutputClassifier…
Making predictions…
Classification Report for Attack Name:
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00      1114
      gafgyt       1.00      1.00      1.00      5548
       mirai       1.00      1.00      1.00      7464

    accuracy                           1.00     14126
   macro avg       1.00      1.00      1.00     14126
weighted avg       1.00      1.00      1.00     14126

Classification Report for Attack SubType:
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00      1114
         ack       1.00      1.00      1.00      1308
       combo       0.99      1.00      0.99       988
        junk       0.99      0.98      0.98       493
        scan       1.00      1.00      1.00      1585
         syn       1.00      1.00      1.00      1505
         tcp       0.75      0.00      0.00      1687
         udp       0.72      1.00      0.84      4412
    udpplain       0.99      0.98      0.99      1034

    accuracy                           0.88     14126
   macro avg       0.94      0.88      0.87     14126
weighted avg       0.88      0.88      0.83     14126

Done.
```

1.8 Conclusion

The performance of the Decision Tree classifier and the Random Forest classifier are approximately the same. There is a minor difference in a couple of the ‘Attack subType’ - but one increase while the other decreases by 0.01 with the RandomForest classifier.

We observe this even after we (a) reduced the dataset to 1% and (b) removed a lot of the columns.

This means that there is a very good correlation between the features used and the target feature. Also, it appears that the dataset is relatively simple and the relationships within the data are easily captured by a single decision tree. Hence, adding more trees (as done in Random Forest) might not significantly improve performance. A simple dataset might already be well-classified by a single decision tree.

[ ]: