Machine Learning Models applied to IoT Threat Classification

1. Introduction

The Internet of Things (IoT) has made everyday life easier by connecting smart devices, improving efficiency, and automating tasks. However, as more devices connect to the internet, security threats also increase. Hackers can exploit vulnerabilities in IoT networks, leading to data breaches, system failures, and cyberattacks. To keep IoT systems secure, we need a reliable threat detection system that can identify and prevent attacks.

In this project, we use machine learning to detect IoT threats by analyzing network traffic data. First, we load and preprocess the dataset, ensuring that the data is clean and well-structured. We then perform feature selection using a method called SelectKBest, which helps us choose the most important factors in identifying threats.

Next, we train and evaluate different machine learning models, including Logistic Regression, Decision Trees, and Neural Networks. Each model is tested for accuracy, precision, and recall to determine its effectiveness. To further improve our detection system, we create an ensemble model, which combines the best-performing models to enhance accuracy and reliability.

By the end of this project, we demonstrate that machine learning can effectively detect IoT threats, and an ensemble approach provides even better results. This study highlights the importance of feature selection, model evaluation, and ensemble techniques in improving security in IoT networks. Future research can explore deep learning methods for even more advanced threat detection.

2. Base Libraries

In this section, the necessary libraries and modules are imported. These range from data manipulation libraries like pandas and numpy to visualization tools such as matplotlib and seaborn. Additionally, various machine learning models and evaluation metrics from the Scikit-learn library are also imported. To ensure a smooth execution, any potential warnings, except for deprecation warnings, are suppressed.

```
# Useful libraries
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

```python
# Feature selection
from sklearn.feature_selection import SelectKBest, f_classif

# Metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Statistic and machine learning models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier, AdaBoostClassifier, StackingClassifier,
ExtraTreesClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier

# Ignore all warnings
warnings.filterwarnings("ignore")

# Re-enable DeprecationWarning
warnings.filterwarnings("default", category=DeprecationWarning)
```

3. Data Loading and Preprocessing

The dataset, sourced from a CSV file, is loaded into a DataFrame for inspection and manipulation. Preliminary exploration includes examining the first few rows, understanding the data's structure, and visualizing the distribution of labels. Any missing values are handled to ensure data integrity.

```python
# Load the CSV file into a DataFrame
df = pd.read_csv('data/DDoS_part-00000-363d1ba3-8ab5-4f96-bc25-4d5862db7cb9-
c000.csv',sep=',')

# Display the first 10 rows of the DataFrame
df.head(10)
```

```
   flow_duration  Header_Length  Protocol Type  Duration       Rate  \
0       0.000000          54.00           6.00     64.00   0.329807
1       0.000000           0.00           1.00     64.00  33.396799
2       1.052463         108.00           6.00     64.00   1.902353
3       0.000000          54.20           6.00     64.00  11.243547
4       0.223192          61.54           6.11     64.64   9.087882
5       0.000000          54.00           6.00     64.00  17.333181
```

```
6       0.000000         0.00          1.00     75.46     0.000000
7       0.000000         0.00          1.00     64.00     1.507148
8       0.088335     29216.00         17.00     64.00  7752.316374
9       0.000000        54.00          6.00     64.00    50.099188

         Srate  Drate  fin_flag_number  syn_flag_number  rst_flag_number  ...
\
0     0.329807    0.0              1.0              0.0              1.0  ...
1    33.396799    0.0              0.0              0.0              0.0  ...
2     1.902353    0.0              0.0              1.0              0.0  ...
3    11.243547    0.0              0.0              1.0              0.0  ...
4     9.087882    0.0              0.0              0.0              0.0  ...
5    17.333181    0.0              0.0              0.0              0.0  ...
6     0.000000    0.0              0.0              0.0              0.0  ...
7     1.507148    0.0              0.0              0.0              0.0  ...
8  7752.316374    0.0              0.0              0.0              0.0  ...
9    50.099188    0.0              0.0              1.0              0.0  ...

        Std  Tot size           IAT  Number   Magnitue    Radius  Covariance
\
0  0.000000     54.00  8.334383e+07     9.5  10.392305  0.000000    0.000000
1  0.000000     42.00  8.312799e+07     9.5   9.165151  0.000000    0.000000
2  0.000000     54.00  8.336548e+07     9.5  10.392305  0.000000    0.000000
3  0.619849     54.20  8.308906e+07     9.5  10.409168  0.878113    3.254011
4  1.692073     54.77  8.333087e+07     9.5  10.434347  2.398780   32.140680
5  0.000000     54.00  8.307592e+07     9.5  10.392305  0.000000    0.000000
6  0.000000     42.00  8.315005e+07     9.5   9.165151  0.000000    0.000000
7  0.000000     42.00  8.315032e+07     9.5   9.165151  0.000000    0.000000
8  0.000000     50.00  8.310233e+07     9.5  10.000000  0.000000    0.000000
9  0.000000     54.00  8.309398e+07     9.5  10.392305  0.000000    0.000000

  Variance  Weight                     label
0     0.00  141.55          DDoS-RSTFINFlood
1     0.00  141.55           DDoS-ICMP_Flood
2     0.00  141.55  DDoS-SynonymousIP_Flood
3     0.12  141.55            DDoS-SYN_Flood
4     0.09  141.55          DDoS-PSHACK_Flood
5     0.00  141.55            DDoS-TCP_Flood
6     0.00  141.55           DDoS-ICMP_Flood
7     0.00  141.55           DDoS-ICMP_Flood
8     0.00  141.55            DDoS-UDP_Flood
9     0.00  141.55            DDoS-SYN_Flood

[10 rows x 47 columns]
```

```
# Display a concise summary of the DataFrame's columns, non-null values, and
data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 173777 entries, 0 to 173776
Data columns (total 47 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   flow_duration    173777 non-null  float64
 1   Header_Length    173777 non-null  float64
 2   Protocol Type    173777 non-null  float64
 3   Duration         173777 non-null  float64
 4   Rate             173777 non-null  float64
 5   Srate            173777 non-null  float64
 6   Drate            173777 non-null  float64
 7   fin_flag_number  173777 non-null  float64
 8   syn_flag_number  173777 non-null  float64
 9   rst_flag_number  173777 non-null  float64
 10  psh_flag_number  173777 non-null  float64
 11  ack_flag_number  173777 non-null  float64
 12  ece_flag_number  173777 non-null  float64
 13  cwr_flag_number  173777 non-null  float64
 14  ack_count        173777 non-null  float64
 15  syn_count        173777 non-null  float64
 16  fin_count        173777 non-null  float64
 17  urg_count        173777 non-null  float64
 18  rst_count        173777 non-null  float64
 19  HTTP             173777 non-null  float64
 20  HTTPS            173777 non-null  float64
 21  DNS              173777 non-null  float64
 22  Telnet           173777 non-null  float64
 23  SMTP             173777 non-null  float64
 24  SSH              173777 non-null  float64
 25  IRC              173777 non-null  float64
 26  TCP              173777 non-null  float64
 27  UDP              173777 non-null  float64
 28  DHCP             173777 non-null  float64
 29  ARP              173777 non-null  float64
 30  ICMP             173777 non-null  float64
 31  IPv              173777 non-null  float64
 32  LLC              173777 non-null  float64
 33  Tot sum          173777 non-null  float64
 34  Min              173777 non-null  float64
 35  Max              173777 non-null  float64
 36  AVG              173777 non-null  float64
 37  Std              173777 non-null  float64
 38  Tot size         173777 non-null  float64
 39  IAT              173777 non-null  float64
 40  Number           173777 non-null  float64
 41  Magnitue         173777 non-null  float64
 42  Radius           173777 non-null  float64
 43  Covariance       173777 non-null  float64
 44  Variance         173777 non-null  float64
```

```
 45  Weight           173777 non-null  float64
 46  label            173777 non-null  object 
dtypes: float64(46), object(1)
memory usage: 62.3+ MB
```

```python
# Get the number of rows and columns in the DataFrame
df.shape
```

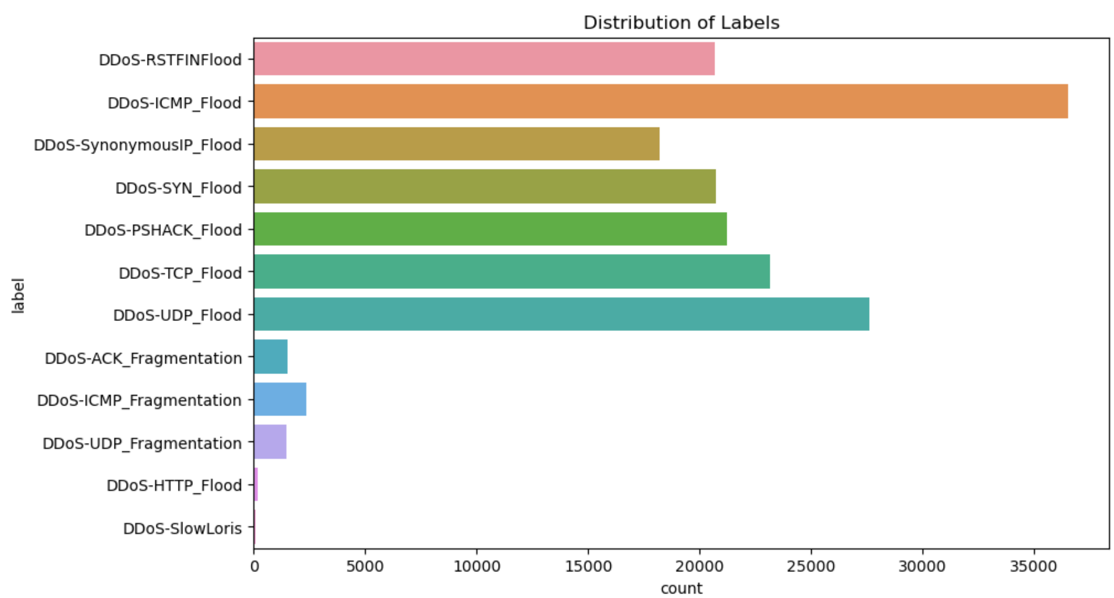
```
(173777, 47)
```

```python
# Display statistical summary of the DataFrame's columns
df.describe()
```

```
       flow_duration  Header_Length  Protocol Type       Duration  \
count  173777.000000   1.737770e+05  173777.000000  173777.000000   
mean        0.341774   6.937612e+03       6.723544      64.287490   
std        13.149956   4.107415e+04       4.992523       2.582139   
min         0.000000   0.000000e+00       0.770000      19.210000   
25%         0.000000   5.375000e+01       5.940000      64.000000   
50%         0.000000   5.400000e+01       6.000000      64.000000   
75%         0.044044   9.612000e+01       6.000000      64.000000   
max      3990.334709   3.188817e+06      17.000000     211.070000   

               Rate         Srate          Drate  fin_flag_number  \
count  1.737770e+05  1.737770e+05  173777.000000     173777.00000   
mean   9.493362e+03  9.493362e+03       0.000007          0.11883   
std    1.038008e+05  1.038008e+05       0.002069          0.32359   
min    0.000000e+00  0.000000e+00       0.000000          0.00000   
25%    2.077204e+00  2.077204e+00       0.000000          0.00000   
50%    1.356388e+01  1.356388e+01       0.000000          0.00000   
75%    7.288358e+01  7.288358e+01       0.000000          0.00000   
max    5.242880e+06  5.242880e+06       0.848465          1.00000   

       syn_flag_number  rst_flag_number  ...            AVG            Std  \
count    173777.000000    173777.000000  ...  173777.000000  173777.000000   
mean          0.223453         0.121443  ...      78.098595      18.339989   
std           0.416561         0.326642  ...     147.318503      95.785662   
min           0.000000         0.000000  ...      42.000000       0.000000   
25%           0.000000         0.000000  ...      50.000000       0.000000   
50%           0.000000         0.000000  ...      54.000000       0.000000   
75%           0.000000         0.000000  ...      54.000000       0.000000   
max           1.000000         1.000000  ...    2279.491224    1350.157783   

            Tot size           IAT         Number       Magnitue  \
count  173777.000000  1.737770e+05  173777.000000  173777.000000   
mean       78.138102  8.319127e+07       9.498836      11.126329   
std       147.257117  1.340093e+06       0.068870       5.578303   
```

```
min        42.000000  0.000000e+00       1.000000       9.165151
25%        50.000000  8.309760e+07       9.500000      10.000000
50%        54.000000  8.312886e+07       9.500000      10.392305
75%        54.000000  8.334092e+07       9.500000      10.392305
max      2338.910000  1.001943e+08      10.000000      63.519115

              Radius    Covariance       Variance         Weight
count  173777.000000  1.737770e+05  173777.000000  173777.000000
mean       25.931840  1.079894e+04       0.059744     141.521179
std       135.451819  6.465186e+04       0.177614       1.464649
min         0.000000  0.000000e+00       0.000000       1.000000
25%         0.000000  0.000000e+00       0.000000     141.550000
50%         0.000000  0.000000e+00       0.000000     141.550000
75%         0.000000  0.000000e+00       0.000000     141.550000
max      1912.134518  9.226184e+06       0.950000     141.550000

[8 rows x 46 columns]
```

```python
# Plotting the distribution of the 'label' column
plt.figure(figsize=(10, 6))
sns.countplot(y=df['label'])
plt.title('Distribution of Labels')
plt.show()
```




```python
# Check for null values
print(df.isnull().sum())
```

```
# Drop rows with null values (you can also fill them if you prefer)
df.dropna(inplace=True)
```

```
flow_duration      0
Header_Length      0
Protocol Type      0
Duration           0
Rate               0
Srate              0
Drate              0
fin_flag_number    0
syn_flag_number    0
rst_flag_number    0
psh_flag_number    0
ack_flag_number    0
ece_flag_number    0
cwr_flag_number    0
ack_count          0
syn_count          0
fin_count          0
urg_count          0
rst_count          0
HTTP               0
HTTPS              0
DNS                0
Telnet             0
SMTP               0
SSH                0
IRC                0
TCP                0
UDP                0
DHCP               0
ARP                0
ICMP               0
IPv                0
LLC                0
Tot sum            0
Min                0
Max                0
AVG                0
Std                0
Tot size           0
IAT                0
Number             0
Magnitue           0
Radius             0
Covariance         0
Variance           0
Weight             0
```

```
label              0
dtype: int64
```

4. Feature Selection

To improve the efficiency and accuracy of our models, we employ a feature selection method to determine and retain only the most relevant features. This process is essential to reduce the dimensionality of the dataset and improve model training times.

```
# Create a new DataFrame 'X' by dropping the 'label' column
X = df.drop('label', axis=1)

# Assign the 'label' column values to the variable 'y'
y = df['label']

# Initialize a SelectKBest object to select the top 10 features based on the
ANOVA F-value
selector = SelectKBest(f_classif, k=10)

# Fit the selector to the data and transform 'X' to retain only the top 10
features
X_new = selector.fit_transform(X, y)

# Get the column names of the selected features from 'X'
selected_features = X.columns[selector.get_support()]
```

```
# Print the names of the selected features
print("Selected Features:", selected_features)
```

```
Selected Features: Index(['Protocol Type', 'fin_flag_number',
'syn_flag_number',
       'rst_flag_number', 'psh_flag_number', 'ack_count', 'fin_count', 'TCP',
       'UDP', 'ICMP'],
      dtype='object')
```

```
# Create a new DataFrame with the selected features and the label
df = pd.concat([pd.DataFrame(X_new, columns=selected_features),
y.reset_index(drop=True)], axis=1)
```

5. Model Training and Validation

Here, we dive into the heart of our analysis. Each machine learning model is trained on a subset of the data and then validated on a separate set. By comparing the models' performances based on accuracy, precision, and recall, we identify the top-performing models. These models are then combined into an ensemble model, aiming to leverage their combined strengths for optimal threat detection.

```
# Create a new DataFrame 'X' by dropping the 'label' column
X = df.drop('label', axis=1)

# Assign the 'label' column values to the variable 'y'
y = df['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

This code block sets up, trains, and evaluates a variety of machine learning models on a given dataset. Initially, a dictionary named "models" is defined, where each key is the name of a machine learning algorithm and its associated value is an instance of that algorithm's classifier. Algorithms such as Logistic Regression, Decision Trees, Random Forest, and KNN, among others, are included. The classifiers are initialized with specific parameters, like a maximum number of iterations for the Logistic Regression and Neural Network models.

In the subsequent portion of the code, each model from the "models" dictionary is trained on a training dataset, X_train and y_train. After training, predictions are made on a test dataset, X_test. The accuracy, precision, and recall of these predictions, relative to the true values y_test, are then computed. The results for each model, including its name and evaluation metrics, are stored in the "results" list. This approach provides a straightforward way to compare the performance of different machine learning models on the same dataset.

```
# Define the models
models = {
    'Logistic Regression': LogisticRegression(max_iter=500),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
}

results = []

# Train, test and evaluate each model
for name, model in models.items():
```

```
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')

    results.append({
        'Model': name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall
    })
```

The part performs model ensembling, a technique where multiple machine learning models are combined to achieve better predictive performance than any individual model. First, it evaluates the performance of various standalone models on a dataset and identifies the top three models based on accuracy.

Once these top models are determined, a stacking ensemble is constructed. In stacking, predictions from individual models are used as inputs to another model (the meta-model), which then makes the final predictions. After training this ensemble model on the training data, it is evaluated on the test data, and its performance metrics (accuracy, precision, and recall) are computed. The aim is to capitalize on the strengths of the top models and possibly achieve superior results through their combined predictions.

```
# Convert the results to a DataFrame and sort by Accuracy
results_df = pd.DataFrame(results).sort_values(by='Accuracy',
ascending=False)

# Get the top 3 models
top_models = results_df.head(3)['Model'].tolist()

# Print the names of the models used in the ensemble
print(f"Models used in the ensemble: {', '.join(top_models)}")

# Create the ensemble model using the top 3 models
ensemble_model = StackingClassifier(estimators=[(name, models[name]) for name
in top_models])
ensemble_model.fit(X_train, y_train)
y_pred_ensemble = ensemble_model.predict(X_test)

# Evaluate the ensemble model
accuracy_ensemble = accuracy_score(y_test, y_pred_ensemble)
precision_ensemble = precision_score(y_test, y_pred_ensemble,
average='weighted')
```

```
recall_ensemble = recall_score(y_test, y_pred_ensemble, average='weighted')

# Add the ensemble model's results to the results list
results.append({
    'Model': 'Ensemble Model',
    'Accuracy': accuracy_ensemble,
    'Precision': precision_ensemble,
    'Recall': recall_ensemble
})
```

```
Models used in the ensemble: K-Nearest Neighbors, Neural Network, Gradient Boosting
```

```
# Convert the results to a DataFrame and sort by Accuracy
final_results_df = pd.DataFrame(results).sort_values(by='Accuracy', ascending=False)

# Display the results
final_results_df.head(10)
```

	Model	Accuracy	Precision	Recall
5	Ensemble Model	0.877230	0.873828	0.877230
2	Random Forest	0.877028	0.856051	0.877028
3	Gradient Boosting	0.876913	0.927239	0.876913
1	Decision Tree	0.876511	0.858782	0.876511
0	Logistic Regression	0.874727	0.817484	0.874727
4	K-Nearest Neighbors	0.872799	0.860573	0.872799

Ensemble Model: This model tops the list with an accuracy of approximately 87.72%. The ensemble technique, which combines predictions from multiple models to improve overall performance, seems to have paid off. Its precision score is slightly higher than its accuracy, suggesting that the model's predictions are quite reliable.

Random Forest: Random Forest comes in a close second, with an accuracy almost matching the ensemble model. Its precision is notably less than its accuracy, which means that among the instances it predicted positively, a high percentage were incorrect.

Gradient Boosting: This model's precision stands out, being the highest among the top models. Although its accuracy is slightly lower than the Random Forest, its high precision suggests that its positive predictions are very reliable.

Decision Tree: The Decision Tree model's performance metrics are closely aligned with the Gradient Boosting. This is expected since Random Forests are essentially an ensemble of Decision Trees.

Logistic Regression: This model has similar accuracy scores, but their precision is lower than the models ranked above them. This might imply that these models are making more false positive predictions compared to the top-performing models.

K-Nearest Neighbors (KNN): This model has the lowest accuracy score among the models. It performs well in terms of accuracy, but its precision score suggest there's room for improvement in the reliability of its positive predictions.

6. Conclusions

In conclusion, as IoT continues to evolve and integrate deeper into our daily lives, ensuring the security of these devices becomes paramount. Through machine learning models and ensemble techniques, this notebook showcases a potential approach to enhancing IoT threat detection capabilities.