Lab 2: IoT Device Detection Using Machine Learning (Logistic regression and K-nearest neighbors)

By : Djallel DILMI

Overview

In this lab, you will explore the process of dataset creation, feature extraction, detecting and classifying objects with a hands-on experience on IoT devices based on their network traffic patterns. The lab is divided into two main parts:

1. **Feature Extraction from pcap Files:**
    - Students will learn how to handle pcap files, which capture network traffic data. They will write a Python script to extract specific features from these files, such as IP addresses, protocols, and packet lengths. These features are crucial for representing the network behavior of IoT devices in a structured format that can be used for machine learning.
2. **Building and Training Machine Learning Models:**
    - In the second part of the lab, students will use the features extracted from the pcap files to train machine learning models (especially Logistic regression and K-nearest neighbors). These models will classify different IoT devices based on their network traffic patterns. you will explore the two classifiers, and evaluate their performance using metrics like accuracy, precision, recall, and F1 score.

1. Dataset Presentation

In this lab, you will be provided with five pcap (Packet Capture) files in "filtered_pcap" folder, each corresponding to network traffic generated by a specific IoT device. These pcap files contain raw network traffic data captured over a period of time, and your task is to extract meaningful features from this data for further analysis and classification.

1.1. Dataset Characteristics:

- **Source:** The dataset consists of five separate pcap files, each representing the network activity of a different IoT device:
    1. **Assistant:** Network traffic captured from a smart assistant device.
    2. **Camera:** Network traffic captured from a security or surveillance camera.
    3. **Miscellaneous:** Network traffic captured from a variety of other IoT devices.

4. **Mobile:** Network traffic captured from a mobile phone or similar device.
5. **Outlet:** Network traffic captured from a smart power outlet.

- **Structure:** each pcap file contains a series of network packets. Each packet in these files includes various details such as:
  - **Timestamp:** The time at which the packet was captured.
  - **Source IP Address:** The IP address of the device that sent the packet.
  - **Destination IP Address:** The IP address to which the packet was sent.
  - **Protocol:** The network protocol used (e.g., TCP, UDP).
  - **Packet Length:** The size of the packet in bytes.
  - **Port Numbers:** Source and destination ports used for communication.
  - **Additional Features:** Other protocol-specific features like sequence numbers, acknowledgment numbers, and checksums,...

- **Class Labels:** each pcap file corresponds to one class label representing a particular IoT device type:

  - **Assistant:** pcap file for smart assistant devices.
  - **Camera:** pcap file for security cameras.
  - **Miscellaneous:** pcap file for miscellaneous IoT devices.
  - **Mobile:** pcap file for mobile devices.
  - **Outlet:** pcap file for smart power outlets.

Part 1: Feature Extraction from pcap Files

In this section, you will focus on extracting features from pcap files that contain network traffic data. These features are essential for training machine learning models to classify different IoT devices based on their network behavior.

Lab Tasks:

1. Introduction to pcap Files:

- **What are pcap files?**
  - Pcap (Packet Capture) files store network traffic data, capturing packets as they travel across a network. Each packet contains various details like the timestamp, source IP, destination IP, protocol, and length.
- **Key details captured in pcap files:**
  - **Timestamp:** When the packet was captured.
  - **Source IP:** The IP address of the device that sent the packet.
  - **Destination IP:** The IP address where the packet is being sent.
  - **Protocol:** The protocol used for communication (e.g., TCP, UDP).
  - **Packet Length:** The size of the packet.

2. Python Script for Feature Extraction:

Task: read and understand the given Python script to extract 18 specific features from pcap files using tshark.

Steps:

1. **Understand the Features to be Extracted:**

- The features include IP lengths, protocols, source and destination IPs, TCP/UDP port numbers, and other packet-specific data.

Key Points to Remember:

- Replace './pcap_files/' with the directory path where your pcap files are stored.
- The script assumes that the pcap files follow a specific naming convention (e.g., TCP_Camera.pcap). Adjust as needed.

3. Store Extracted Features:

- The extracted features will be stored in a CSV file named label_feature_IOT.csv.
- The first column of the CSV will represent the class label (type of device), and the subsequent columns will represent the extracted features.

Example Output:

Label	Feature1	Feature2	...	Feature18
Assistant	...	...	...	...
Camera	...	...	...	...
Miscellaneous	...	...	...	...
Mobile	...	...	...	...
Outlet	...	...	...	...

Part 2: Exploring the Dataset Using ydata_profiling

In this section, you will explore the dataset created in Part 1 using ydata_profiling (formerly known as pandas_profiling). This tool automatically generates a detailed report that includes statistics, visualizations, and summaries for each feature in the dataset. Students will analyze this report to gain insights into the dataset and identify potential issues or interesting patterns.

Good to know:

- **What is ydata_profiling?**
  - ydata_profiling is a Python package that generates comprehensive reports from a pandas DataFrame. The report includes various metrics like missing values, distributions, correlations, and interactions between features.
- **Why use ydata_profiling?**
  - It provides a quick and easy way to perform an initial exploration of the dataset, helping to understand the underlying data, detect anomalies, and identify important features.

Task: Write a Python script to load the CSV file created in Part 1 and generate a profiling report using ydata_profiling.

Steps:

1. **Import the Required Libraries:** Start by importing pandas and ProfileReport from ydata_profiling.

```
import pandas as pd
from ydata_profiling import ProfileReport
```

2. **Load the CSV Dataset:** Load the dataset extracted in Part 1 into a panda DataFrame.

```
df = pd.read_csv('label_feature_IOT.csv')
```

3. **Generate the Profiling Report:** use ydata_profiling to create a profile report for the dataset.

```
profile = ProfileReport(df, title="IoT Device Dataset Profiling Report", \
explorative=True)
profile.to_file("IOT_dataset_profile_report.html")
```

The explorative=True argument enables more interactive and detailed analysis in the report.

4. **Review the Generated Report:**
    - The report will be saved as an HTML file named IOT_dataset_profile_report.html. Open this file in a web browser to review the report.

Analyzing and Commenting on the Report:

Task: After generating the profiling report, students should thoroughly analyze the report and provide detailed comments on their findings.

Steps:

1. **Examine Key Sections of the Report:**
    - **Overview:** Look at the overall statistics of the dataset, including the number of variables, observations, missing cells, and memory usage.
    - **Variables:** Review the summary of each feature, including data type, unique values, missing values, and distribution.
    - **Interactions:** Analyze interactions between different features to understand relationships and correlations.
    - **Missing Values:** Identify any missing data and consider how it might impact further analysis or modeling.
    - **Correlations:** Examine the correlation matrix to see how features relate to one another.
    - **Warnings:** Pay attention to any warnings about potential issues like high cardinality, high correlations, or skewness.
2. **Comment on the Report:**
    - **General Observations:** Summarize the overall structure and characteristics of the dataset.
    - **Anomalies or Issues:** Identify and comment on any anomalies, such as missing data, outliers, or highly correlated features.
    - **Feature Insights:** Discuss any interesting patterns or relationships discovered between features.

- **Actionable Steps:** Suggest potential actions to address any issues found, such as handling missing values, removing or transforming features, or considering feature engineering.

Part 3: Classification of IoT Devices Using Logistic Regression and K-Nearest Neighbors

In this section, you will apply two popular machine learning algorithms—Logistic Regression and K-Nearest Neighbors (KNN)—to classify IoT devices based on the network traffic features extracted in Part 1.
The goal is to train, evaluate, and compare the performance of these classifiers in predicting the type of IoT device that generated the traffic.

Lab Tasks:

1. **Load the Dataset:** Load the CSV file (label_feature_IOT.csv) created in Part 1 into a pandas DataFrame.

```
import pandas as pd
df = pd.read_csv('label_feature_IOT.csv')
```

2. Data Preprocessing:

1. **Label Encoding:** Convert the categorical class labels (e.g., 'Assistant', 'Camera') into numerical labels that can be used by the machine learning algorithms.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Label'] = le.fit_transform(df['Label'])
```

2. **Split the Data:** Split the dataset into features (X) and labels (y).

```
X = df.drop('Label', axis=1)
y = df['Label']
```

3. **Train-Test Split:** Split the dataset into a training set and a testing set. Typically, 70-80% of the data is used for training, and 20-30% is used for testing.

```
... To DO
```

4. **Feature Scaling:** Scale the features to ensure that they have similar magnitudes, which is important for algorithms like KNN.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

3.1. Logistic Regression:

1. **Train the Model:** Use the training data to train a Logistic Regression model.

```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train, y_train)
```

2. **Make Predictions:** Use the trained model to make predictions on the test data.

```
y_pred_lr = lr_model.predict(X_test)
```

3. **Evaluate the Model:** Evaluate the performance of the Logistic Regression model using metrics like accuracy, precision, recall, and F1 score.

```
from sklearn.metrics import classification_report, confusion_matrix
print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_lr, target_names=le.classes_))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_lr))
```

3.2. K-Nearest Neighbors (KNN):

1. **Train the Model:** Use the training data to train a K-Nearest Neighbors classifier.

```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
```

2. **Make Predictions: (similar to 3.1)**
3. **Evaluate the Model: (similar to 3.1)**

3.3. Comparing the Models:

1. **Accuracy Comparison:** Compare the accuracy of both models and determine which one performs better on the given dataset.

```
lr_accuracy = lr_model.score(X_test, y_test)
knn_accuracy = knn_model.score(X_test, y_test)
print(f"Logistic Regression Accuracy: {lr_accuracy:.2f}")
print(f"K-Nearest Neighbors Accuracy: {knn_accuracy:.2f}")
```

2. **Confusion Matrix Visualization:** Visualize the confusion matrices of both models using a heatmap.

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
sns.heatmap(confusion_matrix(y_test, y_pred_lr), annot=True, fmt='d', cmap='Blues')
plt.title('Logistic Regression Confusion Matrix')
plt.subplot(1, 2, 2)
sns.heatmap(confusion_matrix(y_test, y_pred_knn), annot=True, fmt='d', \
cmap='Blues')
plt.title('K-Nearest Neighbors Confusion Matrix')
plt.show()
```

3. Discuss the difference between LR and KNN:

- Draw the SWOT matrices for the two techniques.
- In your opinion what is the best techniques in this use case? Why?
- What are the best scenarios for using both techniques

Part 4. Report and Submit Findings:

- **Write a Report:**
  - Summarize the steps you took in this lab, including data preprocessing, model training, and evaluation.
  - Compare the results of the Logistic Regression and K-Nearest Neighbors models, discussing their performance based on the accuracy, precision, recall, and F1 score.
  - Include visualizations such as the confusion matrices and any other relevant charts.
- **Submit the Report:**
  - Name your report as Lab_IOT_classification_yourname.doc.
  - Ensure that you include all code snippets, outputs, and your analysis.