ISSA
Information Systems Security Association
Middle Tennessee
INFOSEC
NASHVILLE

INFOSEC
NASHVILLE 2024

Cybersecurity Crossroads: Securing the Intersection of Innovation and Tradition

September 12th, 2024

ML for Cybersecurity 102

From Theory to Practice

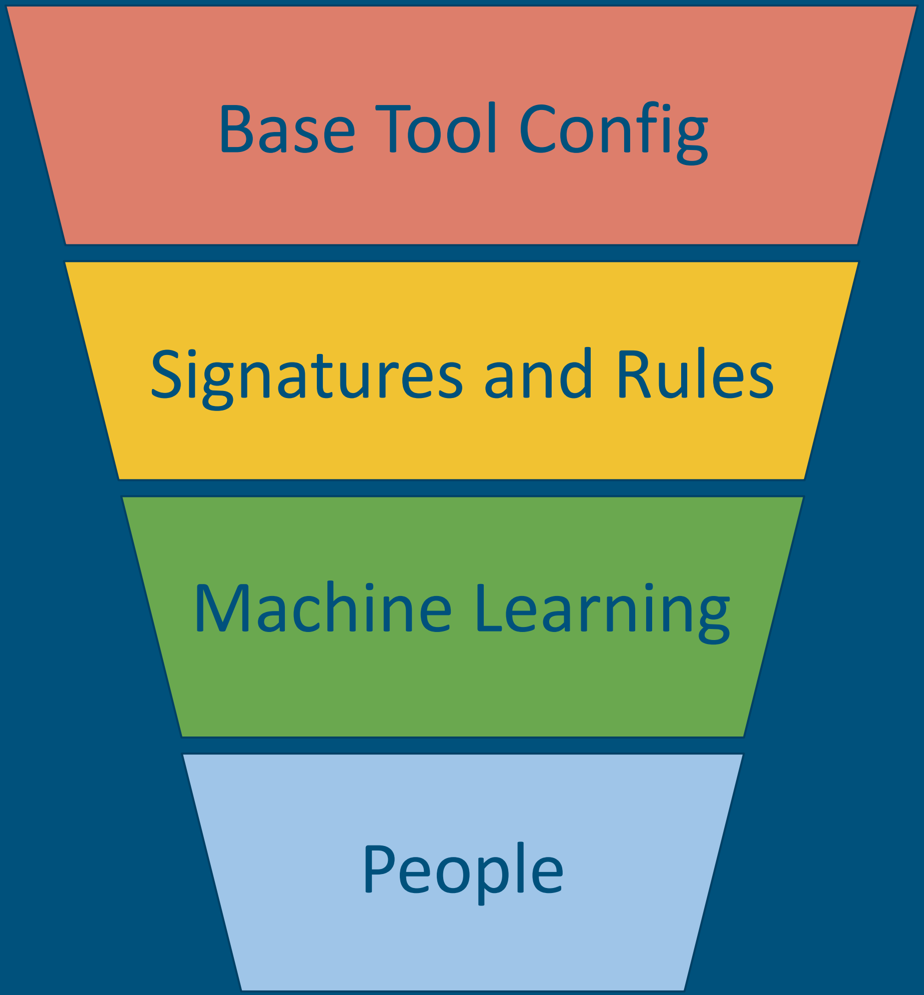
Robert Chapman
Cloud Engineer

Share your thoughts/
photos on LinkedIn!

#InfoSec2024

Imagine being faced with thousands of alerts daily and having to manually triage each one. Now imagine an algorithm that can prioritize those for you, flagging only the most suspicious for review.

Why Cybersecurity Needs Machine Learning



ML in Cybersecurity: Why Now?

1. Investment in upskilling and training engineers/analysts in programming (Python)
2. Square tabular data available in log analytics tools
3. Limited resources for FTE roles on infosec teams
4. A need to prioritize information and provide coverage for gaps in rules based systems
5. A need to reduce manual effort

Types of Machine Learning - Supervised

Definition: In supervised learning, the model is trained on labeled data, meaning the input data is paired with the correct output.

Goal: The objective is to learn a mapping from inputs to outputs so that the model can predict the output for new, unseen inputs. (**Classification**)

Common Algorithms:

- Linear Regression
- **Logistic Regression**
- Decision Trees
- **Random Forests**
- Support Vector Machines (SVM)
- Neural Networks
- Gradient Boosting Machines (e.g., XGBoost, LightGBM)

Types of Machine Learning - Unsupervised

Definition: Unsupervised learning involves training on data that has no labeled responses. The goal is to find hidden patterns or intrinsic structures in the input data.

Goal: The primary objective is clustering, **anomaly detection**, or dimensionality reduction.

Common Algorithms:

- **Isolation Forest**
- K-Means Clustering
- DBSCAN
- Principal Component Analysis (PCA)

Foundational Algorithms

1. Isolation Forest: Anomaly Detection
2. Logistic Regression: Binary Classification
3. Random Forest: Ensemble Method for Classification and Detection

Feature Engineering

Feature engineering is the process of creating new features or modifying existing ones from raw data to improve the performance of machine learning models.

It involves **transforming**, **combining**, or **selecting features** in a way that help the model understand important details or signals from the data that might not be obvious.

Data Preparation! - Example Network Data

	Flow.ID	Source.IP	Source.Port	Destination.IP	Destination.Port	Protocol	Timestamp	Flow.Duration	Total.Fwd.Packets	Total.Backward.Packe
0	172.19.1.46-10.200.7.7-52422-3128-6	172.19.1.46	52422	10.200.7.7	3128	6	26/04/201711:11:17	45523	22	5
1	172.19.1.46-10.200.7.7-52422-3128-6	10.200.7.7	3128	172.19.1.46	52422	6	26/04/201711:11:17	1	2	
2	10.200.7.217-50.31.185.39-38848-80-6	50.31.185.39	80	10.200.7.217	38848	6	26/04/201711:11:17	1	3	
3	10.200.7.217-50.31.185.39-38848-80-6	50.31.185.39	80	10.200.7.217	38848	6	26/04/201711:11:17	217	1	
4	192.168.72.43-10.200.7.7-55961-3128-6	192.168.72.43	55961	10.200.7.7	3128	6	26/04/201711:11:17	78068	5	

5 rows × 87 columns

How should we prepare this data for use in machine learning?

What do we keep? What do we drop? What types of data do ML models support?

https://www.kaggle.com/datasets/jsrojas/ip-network-traffic-flows-labeled-with-87-apps

Feature Engineering an IP Address

192.168.0.1

- **Split the octets**: 192, 168, 0, 1
  - Different octets can signify different network classes or subnets.
- **Create a boolean**: if it’s a private address 192.168.0.1 → True
  - Distinguishing private IPs can help focus on external threats vs. internal traffic.
- **Geolocate**: 8.8.8.8 resolves to Mountain View, CA, USA
  - Helps detect unusual access patterns (e.g., login attempts from unfamiliar locations).
- **Classification**: 192.168.0.1/24 are end user machines
  - Apply known classification details derived from policy

Handling Missing Data

Scenario: You have a dataset of network logs where some entries might be incomplete due to packet loss or system errors.

Techniques:

1. **Imputation**: Fill missing values with a default value, such as 0 for missing packet counts.
2. **Dropping**: Remove logs with missing critical fields, like IP addresses or timestamps.

```
[9]: import pandas as pd
     from sklearn.impute import SimpleImputer
```

```
[54]: # Sample DataFrame
      df = pd.DataFrame({
          'timestamp': ['2024-08-01 00:00:01', '2024-08-01 00:00:02', None, '2024-08-01 00:00:04', '2024-08-01 00:00:05'],
          'src_ip': ['192.168.1.1', '192.168.1.2', '192.168.1.3', None, '192.168.1.5'],
          'packet_count': [100, 150, None, 200, 0]
      })
```

```
[55]: print(df)
```

```
             timestamp       src_ip  packet_count
0  2024-08-01 00:00:01  192.168.1.1         100.0
1  2024-08-01 00:00:02  192.168.1.2         150.0
2                 None  192.168.1.3           NaN
3  2024-08-01 00:00:04         None         200.0
4  2024-08-01 00:00:05  192.168.1.5           0.0
```

```
[56]: # Imputation for packet_count
      imputer = SimpleImputer(strategy='median')
      df['packet_count'] = imputer.fit_transform(df[['packet_count']])
```

```
[57]: print(df)
```

```
             timestamp       src_ip  packet_count
0  2024-08-01 00:00:01  192.168.1.1         100.0
1  2024-08-01 00:00:02  192.168.1.2         150.0
2                 None  192.168.1.3         125.0
3  2024-08-01 00:00:04         None         200.0
4  2024-08-01 00:00:05  192.168.1.5           0.0
```

```
[58]: # Drop rows with missing IPs or timestamps
      df.dropna(subset=['timestamp', 'src_ip'], inplace=True)
```

```
[59]: print(df)
```

```
             timestamp       src_ip  packet_count
0  2024-08-01 00:00:01  192.168.1.1         100.0
1  2024-08-01 00:00:02  192.168.1.2         150.0
4  2024-08-01 00:00:05  192.168.1.5           0.0
```

Feature Scaling

Scenario: You are analyzing network traffic where features like packet size and time intervals have different scales.

Techniques:

1. **Standardization**: makes the features have a mean of 0 and a standard deviation of 1, which centers the data and makes it easier for algorithms like SVM or Logistic Regression to converge.
2. **Normalization**: rescales the data to a fixed range (typically 0 to 1), which is helpful for algorithms like k-NN or neural networks that are sensitive to feature magnitudes.

```
[1]: import pandas as pd
     from sklearn.preprocessing import StandardScaler, MinMaxScaler

[2]: # Sample DataFrame
     df = pd.DataFrame({
         'packet_size': [1500, 600, 2000, 50],
         'time_interval': [0.1, 0.5, 1.0, 0.05]
     })

[3]: # Standardization
     scaler = StandardScaler()
     df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)

[4]: # Normalization
     normalizer = MinMaxScaler()
     df_normalized = pd.DataFrame(normalizer.fit_transform(df), columns=df.columns)

[5]: print(df)

        packet_size  time_interval
     0         1500           0.10
     1          600           0.50
     2         2000           1.00
     3           50           0.05

[6]: print(df_scaled)

        packet_size  time_interval
     0     0.609017      -0.819342
     1    -0.576098       0.229416
     2     1.267415       1.540363
     3    -1.300334      -0.950437

[7]: print(df_normalized)

        packet_size  time_interval
     0     0.743590       0.052632
     1     0.282051       0.473684
     2     1.000000       1.000000
     3     0.000000       0.000000

[ ]:
```

Encoding Categorical Variables

Why It's Important: Machine learning algorithms work with numerical data, so categorical data needs to be converted into a numerical format.

Techniques:

1. **One-Hot Encoding**: Creates binary columns for each category.
   a. **Example**: If a column has three categories: “Red”, “Blue”, and “Green”, one-hot encoding will create three binary columns: “Is_Red”, “Is_Blue”, and “Is_Green”.
   b. Best for when you have a handful of categories that don’t have ordinality
2. **Label Encoding**: Assigns a unique integer to each category.
   a. **Example**: For column with three categories: “low”, “medium”, and “high”, label encoding might assign: low = 0, medium = 1, and high = 2.
   b. Best for when ordinality is a natural part of the category type and should influence the model

```
[7]: import pandas as pd
     from sklearn.preprocessing import OneHotEncoder, LabelEncoder
```

```
[8]: # Sample DataFrame
     df = pd.DataFrame({
         'protocol': ['TCP', 'UDP', 'ICMP', 'TCP'],
         'flag': ['SYN', 'ACK', 'None', 'SYN-ACK']
     })
```

```
[9]: # Display the original DataFrame
     print("Original DataFrame:")
     print(df)
```

```
Original DataFrame:
  protocol     flag
0      TCP      SYN
1      UDP      ACK
2     ICMP     None
3      TCP  SYN-ACK
```

```
[10]: # One-Hot Encoding
      onehot_encoder = OneHotEncoder(sparse_output=False)
      df_onehot_encoded = pd.DataFrame(onehot_encoder.fit_transform(df[['protocol']]),
                                       columns=onehot_encoder.get_feature_names_out(['protocol']))
```

```
[11]: # Label Encoding
      label_encoder = LabelEncoder()
      df['flag_encoded'] = label_encoder.fit_transform(df['flag'])
```

```
[12]: # Display the DataFrame after One-Hot Encoding
      print("\nOne-Hot Encoded DataFrame:")
      print(df_onehot_encoded)
```

```
One-Hot Encoded DataFrame:
   protocol_ICMP  protocol_TCP  protocol_UDP
0            0.0           1.0           0.0
1            0.0           0.0           1.0
2            1.0           0.0           0.0
3            0.0           1.0           0.0
```

```
[13]: # Display the DataFrame after Label Encoding
      print("\nLabel Encoded DataFrame:")
      print(df)
```

```
Label Encoded DataFrame:
  protocol     flag  flag_encoded
0      TCP      SYN             2
1      UDP      ACK             0
2     ICMP     None             1
3      TCP  SYN-ACK             3
```

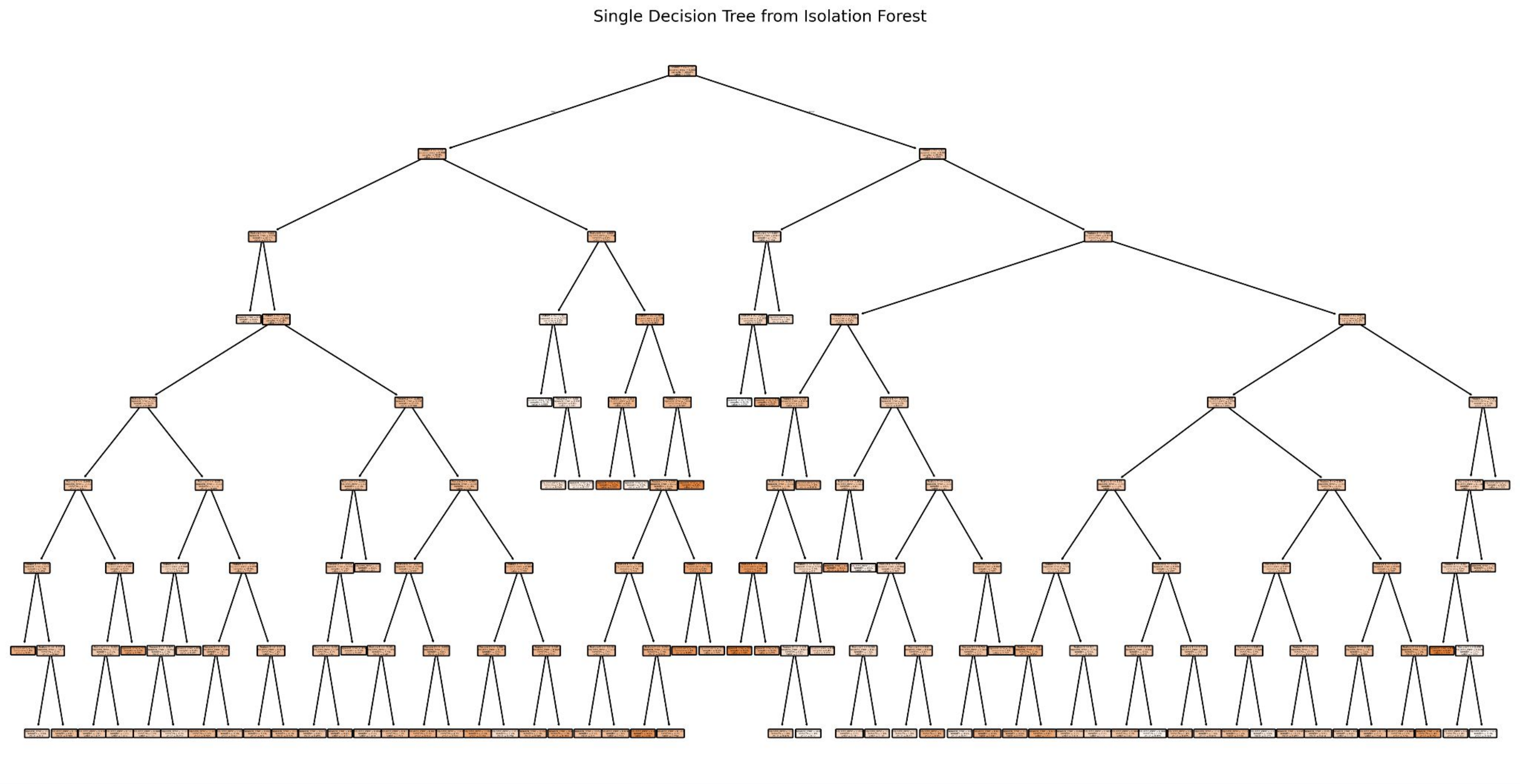
Isolation Forest

Description: Isolation Forest is an unsupervised learning algorithm used primarily for anomaly detection. It operates on the principle that anomalies are few and different, making them easier to isolate.

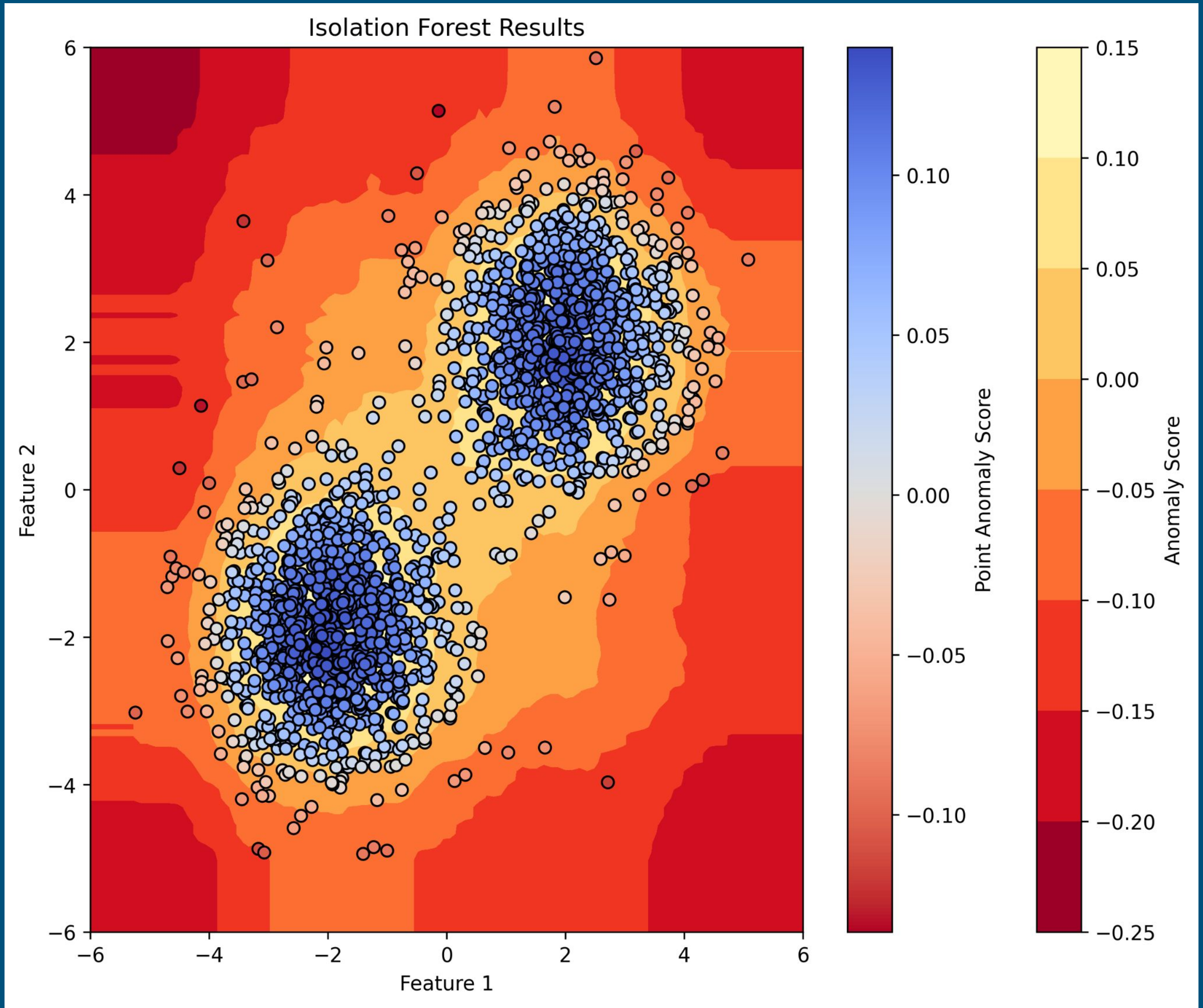
How It Works:

- The algorithm builds decision trees by randomly selecting features and splitting points.
- It isolates observations by creating partitions, and the fewer splits required to isolate an observation, the more likely it is to be an anomaly.
- The path length from the root node to the leaf node represents how easily an observation can be isolated.

Isolation Forest - Decision Tree



Isolation Forest - Plot



Isolation Forest Use Cases

1. Anomaly detection in network traffic: If most traffic is internal and suddenly a large amount of traffic starts flowing to an unusual external IP, Isolation Forest can flag that as potentially malicious.
2. Identifying compromised endpoints: Detect anomalies in system logs or user behavior that might indicate a compromised machine or insider threat.
3. Uncovering rare attack signatures: Isolate rare but potentially harmful security events that might go unnoticed with traditional monitoring.

Isolation Forest

Python Notebook Example

```
[1]: import pandas as pd
     from sklearn.ensemble import IsolationForest
     from sklearn.preprocessing import StandardScaler
     from sklearn.metrics import classification_report, precision_recall_curve, roc_curve, auc, confusion_matrix
     import matplotlib.pyplot as plt
     import seaborn as sns
     import numpy as np
```

```
[3]: data = pd.read_csv('/Users/rob/Documents/Infosec Nashville/dataset_cybersecurity_michelle.csv')
```

```
[4]: # Separate features and labels
     X = data.drop(columns=['phishing'])  # Features
     y = data['phishing']  # Labels
```

```
[5]: # Scale the data
     scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X)
```

```
[6]: # Split into benign (0) and phishing (1)
     X_benign = X_scaled[y == 0]  # Known good (train on this)
     X_malicious = X_scaled[y == 1]  # Known bad (test on this)
```

```
[7]: # Manually adjust parameters for Isolation Forest
     contamination_value = 0.1  # Manually set the contamination value here
     n_estimators_value = 100  # You can manually adjust other parameters as needed
```

```
[8]: # Initialize and fit the Isolation Forest model
     iso_forest = IsolationForest(n_estimators=n_estimators_value, contamination=contamination_value, random_state=42)
     iso_forest.fit(X_benign)
```

```
[8]: IsolationForest
     IsolationForest(contamination=0.1, random_state=42)
```

```
[9]: # Predict anomalies on both benign and malicious data
     y_pred_benign = iso_forest.predict(X_benign)
     y_pred_malicious = iso_forest.predict(X_malicious)
```

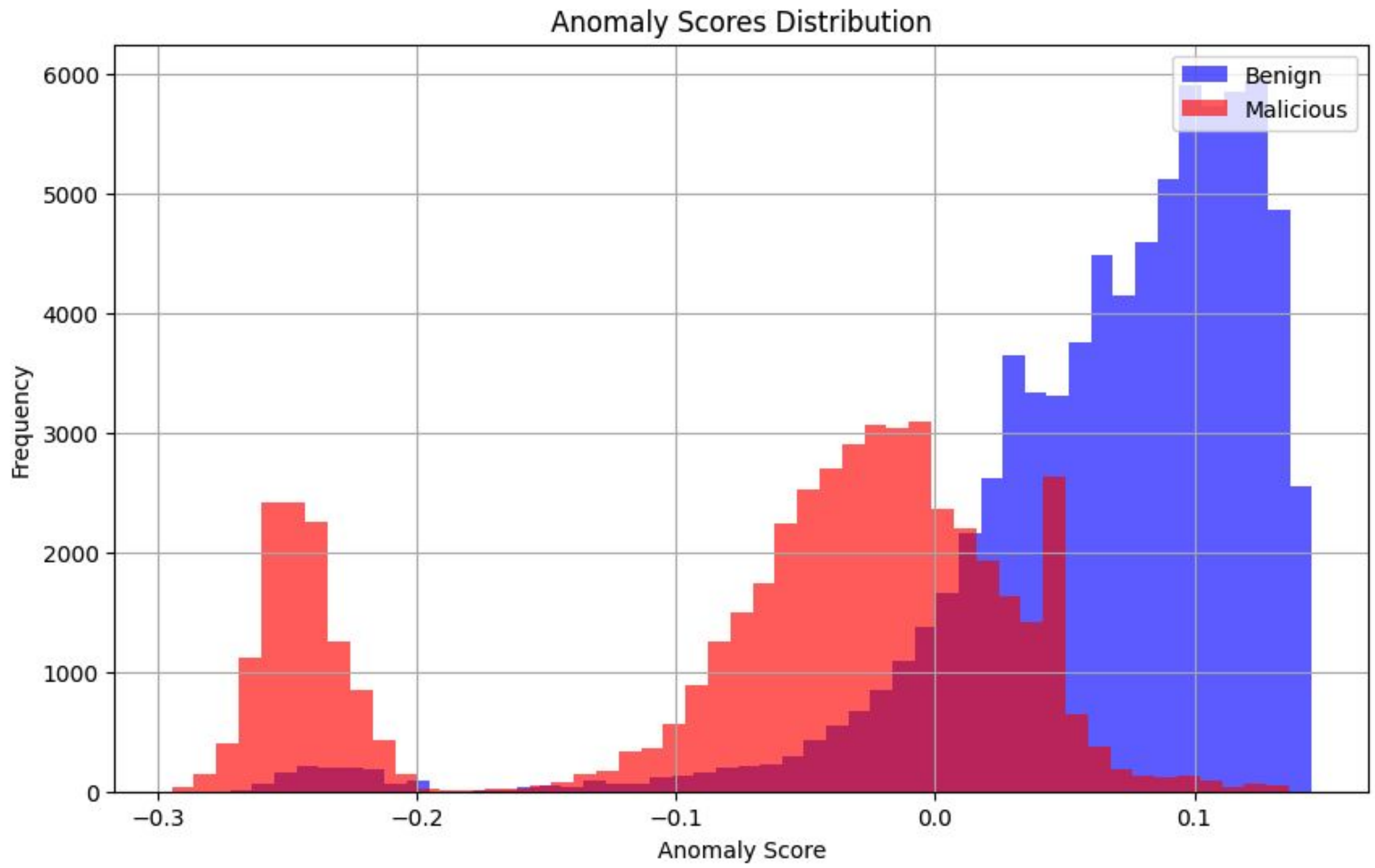
```
[10]: # Convert -1 to 1 for anomalies, and 1 to 0 for normal
      y_pred_benign = [1 if i == -1 else 0 for i in y_pred_benign]
      y_pred_malicious = [1 if i == -1 else 0 for i in y_pred_malicious]
```

```
[11]: # Combine predictions from both datasets for evaluation
      y_pred = y_pred_benign + y_pred_malicious
      y_true = [0] * len(y_pred_benign) + [1] * len(y_pred_malicious)
```

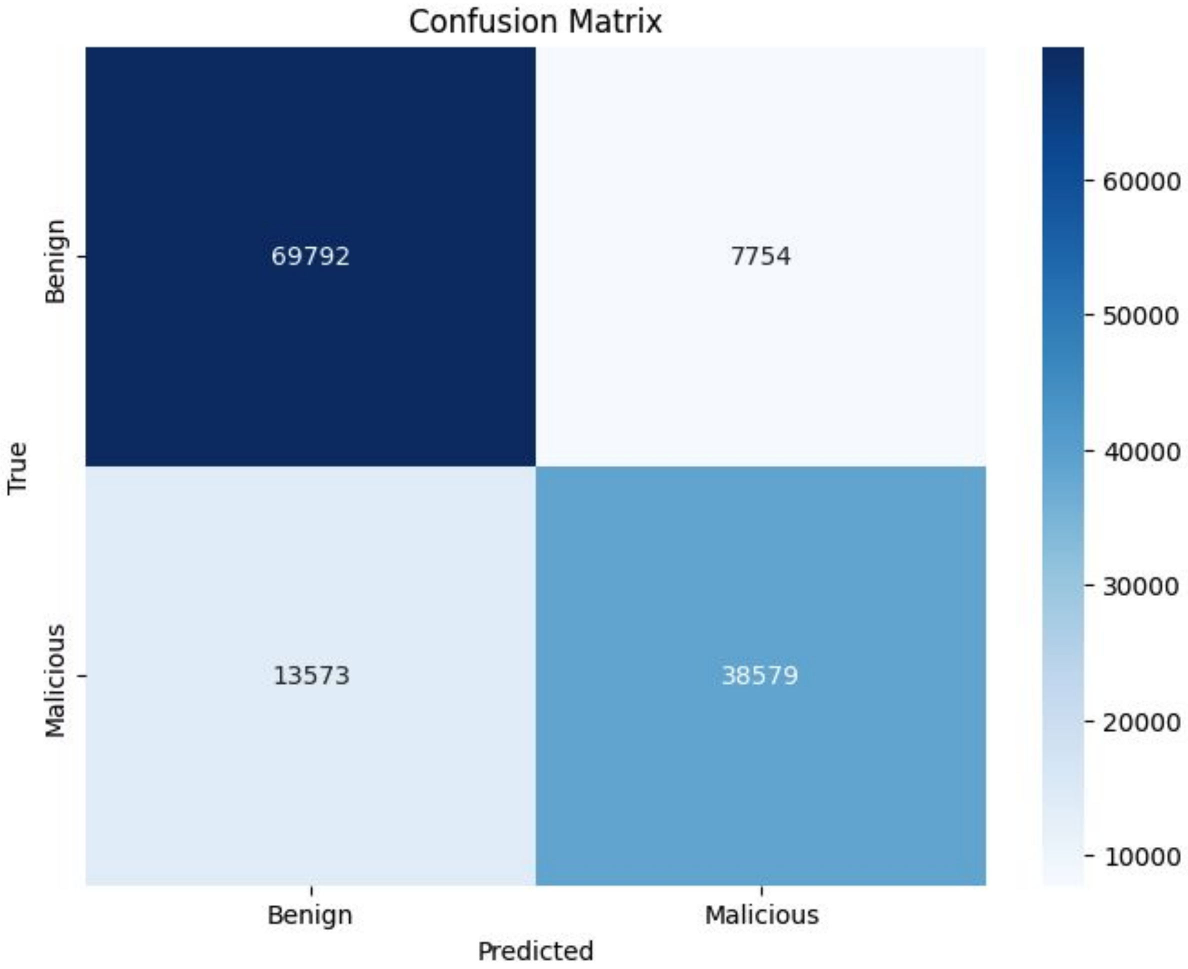
```
[12]: # Evaluate the performance using the classification report
      print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.90      0.87     77546
           1       0.83      0.74      0.78     52152

    accuracy                           0.84    129698
   macro avg       0.83      0.82      0.83    129698
weighted avg       0.84      0.84      0.83    129698
```

[16]:
```python
# Plot 3: Anomaly Scores Distribution
benign_scores = iso_forest.decision_function(X_benign)
malicious_scores = iso_forest.decision_function(X_malicious)
plt.figure(figsize=(10, 6))
plt.hist(benign_scores, bins=50, alpha=0.6, color='blue', label='Benign')
plt.hist(malicious_scores, bins=50, alpha=0.6, color='red', label='Malicious')
plt.title('Anomaly Scores Distribution')
plt.xlabel('Anomaly Score')
plt.ylabel('Frequency')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```



```python
# Plot 4: Confusion Matrix
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Benign', 'Malicious'], yticklabels=['Benign', 'Malicious'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



Measuring Performance

Accuracy: The proportion of correctly predicted instances (both positive and negative) out of the total instances.

Precision: The proportion of true positive predictions out of all instances predicted as positive.

Recall: The proportion of true positive predictions out of all actual positive instances.

F1-Score: The harmonic mean of precision and recall, balancing the two metrics.

Example: “In a phishing detection system, precision ensures that flagged emails are truly phishing attempts, and recall makes sure we catch as many phishing emails as possible.”

Logistic Regression

Description: Logistic Regression is a supervised learning algorithm used for binary classification. It models the probability that an instance belongs to a particular class by using the logistic function.

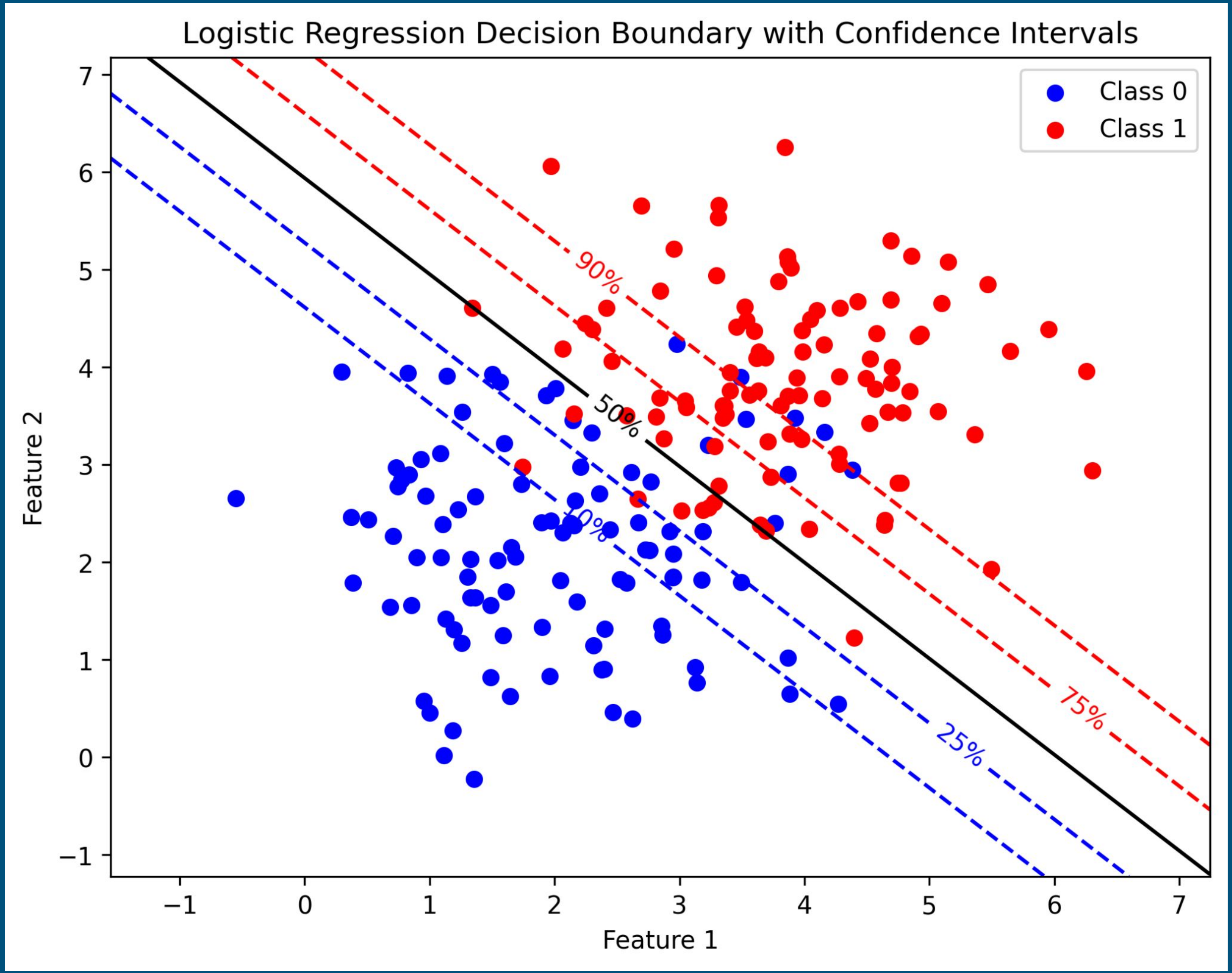
How It Works:

- The algorithm estimates the relationship between input features and the probability of a binary outcome.
- It applies a logistic function to the linear combination of input features to produce a value between 0 and 1, interpreted as the probability of belonging to the positive class.
- The threshold (usually 0.5) is used to classify the instance into one of two categories.

Logistic Regression Use Cases

1. Spam detection: Predict whether an email is spam or not based on various features such as text content, sender details, and metadata.
2. Fraud detection: Classify transactions as fraudulent or legitimate by analyzing transaction patterns and user behavior.
3. Login anomaly detection: Identify unusual login attempts that could signal a brute force attack or account compromise.

Logistic Regression - Plot



Logistic Regression

Python Notebook Example

[22]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

[23]:
```python
df = pd.read_csv('/Users/rob/Documents/Infosec Nashville/dataset_cybersecurity_michelle.csv')
```

[24]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 129698 entries, 0 to 129697
Columns: 112 entries, qty_dot_url to phishing
dtypes: float64(1), int64(111)
memory usage: 110.8 MB
```

[25]:
```python
df.head()
```

[25]:

	qty_dot_url	qty_hyphen_url	qty_underline_url	qty_slash_url	qty_questionmark_url	qty_equal_url	qty_at_url	qty_and_url	qty_exclamation_url	qty_space_
0	1	0	0	1	0	0	0	0	0	
1	2	5	4	2	0	0	0	0	0	
2	2	0	0	0	0	0	0	0	0	
3	1	1	0	2	0	0	0	0	0	
4	2	1	0	0	0	0	0	0	0	

5 rows × 112 columns

[26]:
```python
# Define the features and target variable
X = df.drop('phishing', axis=1)  # Replace 'target_column_name' with the actual name
y = df['phishing']
```

[27]:
```python
# First split: 80% training, 20% remaining (validation + testing)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42)
```

[28]:
```python
# Second split: 50% validation, 50% testing from the remaining 20%
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

[29]:
```python
# Check the sizes of each set
len(X_train), len(X_val), len(X_test)
```

[29]:
```
(103758, 12970, 12970)
```

[30]:
```python
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

[31]:
```python
# Initialize and train the model
model = LogisticRegression(C=1, solver='lbfgs', max_iter=2000, random_state=42)
model.fit(X_train_scaled, y_train)
```

[31]:
```
LogisticRegression
LogisticRegression(C=1, max_iter=2000, random_state=42)
```

[32]:
```python
# Predict on the validation set
y_val_pred = model.predict(X_val_scaled)
```

[33]:
```python
# Predict on the test set
y_test_pred = model.predict(X_test_scaled)
```

[34]:
```python
# Evaluate on validation set
val_accuracy = accuracy_score(y_val, y_val_pred)
val_report = classification_report(y_val, y_val_pred)
val_conf_matrix = confusion_matrix(y_val, y_val_pred)
```

[35]:
```python
# Evaluate on test set
test_accuracy = accuracy_score(y_test, y_test_pred)
test_report = classification_report(y_test, y_test_pred)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)
```

[36]:
```python
# Print the results
print(f"Validation Accuracy: {val_accuracy:.2f}")
print("Validation Classification Report:\n", val_report)
print("Validation Confusion Matrix:\n", val_conf_matrix)
```

```
Validation Accuracy: 0.93
Validation Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.93      0.94      7768
           1       0.90      0.92      0.91      5202

    accuracy                           0.93     12970
   macro avg       0.92      0.92      0.92     12970
weighted avg       0.93      0.93      0.93     12970

Validation Confusion Matrix:
 [[7245  523]
 [ 442 4760]]
```

[37]:
```python
print(f"Test Accuracy: {test_accuracy:.2f}")
print("Test Classification Report:\n", test_report)
print("Test Confusion Matrix:\n", test_conf_matrix)
```

```
Test Accuracy: 0.92
Test Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.93      0.94      7623
           1       0.90      0.92      0.91      5347

    accuracy                           0.92     12970
   macro avg       0.92      0.92      0.92     12970
weighted avg       0.92      0.92      0.92     12970

Test Confusion Matrix:
 [[7094  529]
 [ 451 4896]]
```

[38]:

```python
import numpy as np

# Retrieve the feature names
feature_names = X_train.columns

# Retrieve the coefficients from the model
coefficients = model.coef_[0]

# Create a DataFrame to hold the feature importance information
feature_importance = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients
})

# Calculate the absolute value of coefficients to interpret importance
feature_importance['Absolute Coefficient'] = np.abs(feature_importance['Coefficient'])

# Sort the features by importance
feature_importance = feature_importance.sort_values(by='Absolute Coefficient', ascending=False)

# Display the top features
feature_importance.head(10)
```

[38]:

	Feature	Coefficient	Absolute Coefficient
93	params_length	2.076469	2.076469
18	length_url	1.833763	1.833763
95	qty_params	-1.492762	1.492762
6	qty_at_url	1.441855	1.441855
100	time_domain_activation	-1.361090	1.361090
45	qty_equal_directory	1.209900	1.209900
57	directory_length	1.042548	1.042548
68	qty_tilde_file	-1.015693	1.015693
40	qty_dot_directory	1.011678	1.011678
94	tld_present_params	0.850796	0.850796

[20]:
```python
# Display the bottom features
feature_importance.tail(10)
```

[20]:

	Feature	Coefficient	Absolute Coefficient
32	qty_asterisk_domain	0.0	0.0
29	qty_tilde_domain	0.0	0.0
33	qty_hashtag_domain	0.0	0.0
24	qty_equal_domain	0.0	0.0
35	qty_percent_domain	0.0	0.0
26	qty_and_domain	0.0	0.0
23	qty_questionmark_domain	0.0	0.0
22	qty_slash_domain	0.0	0.0
28	qty_space_domain	0.0	0.0
27	qty_exclamation_domain	0.0	0.0

[41]:
```python
def predict_single_entry(index, model, X_test_scaled, y_test):
    """
    Predict the label for a single entry from the test set.

    Parameters:
    - index: The index of the entry in the test set.
    - model: The trained logistic regression model.
    - X_test_scaled: The scaled test set features.
    - y_test: The test set labels.

    Returns:
    - None. Prints out the prediction details.
    """

    # Step 1: Extract the single entry (scaled features) and the true label
    single_entry_scaled = X_test_scaled[index:index+1]  # Extract the entry (already scaled)
    single_label = y_test.iloc[index]                    # Extract the corresponding label

    # Step 2: Make the prediction
    single_prediction = model.predict(single_entry_scaled)
    single_prediction_proba = model.predict_proba(single_entry_scaled)

    # Step 3: Display the results
    print(f"Test Entry at Index {index} (Scaled Features):\n", single_entry_scaled)
    print(f"Actual Label: {single_label}")
    print(f"Predicted Label: {single_prediction[0]}")
    print(f"Prediction Probabilities (Probability of Class 0 and Class 1):", single_prediction_proba[0])

# Example usage
predict_single_entry(0, model, X_test_scaled, y_test)  # Predicts the first entry in the test set
predict_single_entry(10, model, X_test_scaled, y_test) # Predicts the 11th entry in the test set
predict_single_entry(25, model, X_test_scaled, y_test) # Predicts the 26th entry in the test set
```

```
Test Entry at Index 0 (Scaled Features):
 [[-0.1654897  -0.30722369 -0.18688861  0.26270293 -0.09193443 -0.23467967
  -0.09020384 -0.16572616 -0.03356408 -0.01383338 -0.04339784 -0.03322978
  -0.02632453 -0.01552704 -0.00892116 -0.02003729 -0.06355879 -0.19933905
  -0.14440608  0.20882222 -0.27877486 -0.02010553  0.          0.
   0.         -0.0031045   0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   1.37663191  0.97524586 -0.05145631 -0.06510957  0.23173298  0.22052497
   0.56407859  0.4471084   0.92446901  0.87027123  0.83437224  0.84591792
   0.90916413  0.91294189  0.89819739  0.91629948  0.89646151  0.75251238
   0.92446901  0.90507094  0.22972318 -0.22292744  0.34216348  0.4730884
   0.70463167  0.92446901  0.92446901  0.91251795  0.92230416  0.91482547
   0.91786143  0.91761209  0.9215028   0.9176045   0.90197579  0.79989487
   0.92446901  0.92446901  0.250742   -0.22507649 -0.20956933 -0.20408153
  -0.22764668 -0.21974683 -0.31249785 -0.26690103 -0.30837461 -0.2260317
  -0.32265595 -0.3285704  -0.32885697 -0.32224951 -0.32109964 -0.32826742
  -0.32894675 -0.32634706 -0.14411737 -0.19737529 -0.3084925  -0.27778898
  -0.14670269 -0.22812772 -1.71563634 -0.31440947 -0.65561542 -0.05831436
  -0.14287504  0.89992406 -0.41392625  0.583181   -1.01118458 -0.42029676
  -0.02694152 -0.03524927 -0.07945961]]
Actual Label: 1
Predicted Label: 1
Prediction Probabilities (Probability of Class 0 and Class 1): [0.32688574 0.67311426]
```

Hyperparamter Tuning

This next code block is for hyperparameter tuning. It not necessary to run this step for this model as it's already been run but I'm leaving the code in case you want to reference it.

[20]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Define a refined hyperparameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'solver': ['lbfgs'],  # Using a more stable solver
}

# Initialize the logistic regression model with increased iterations
model = LogisticRegression(max_iter=2000, random_state=42)

# Set up GridSearchCV
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')

# Perform grid search on the training set
grid_search.fit(X_train_scaled, y_train)

# Retrieve the best model from grid search
best_model = grid_search.best_estimator_

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)
```

```
Best Hyperparameters: {'C': 1, 'solver': 'lbfgs'}
```

[ ]:
```
import joblib

# Save the trained model to a file
joblib.dump(model, 'logistic_regression_model.pkl')

# To load the model back into memory
loaded_model = joblib.load('logistic_regression_model.pkl')
```

Random Forest

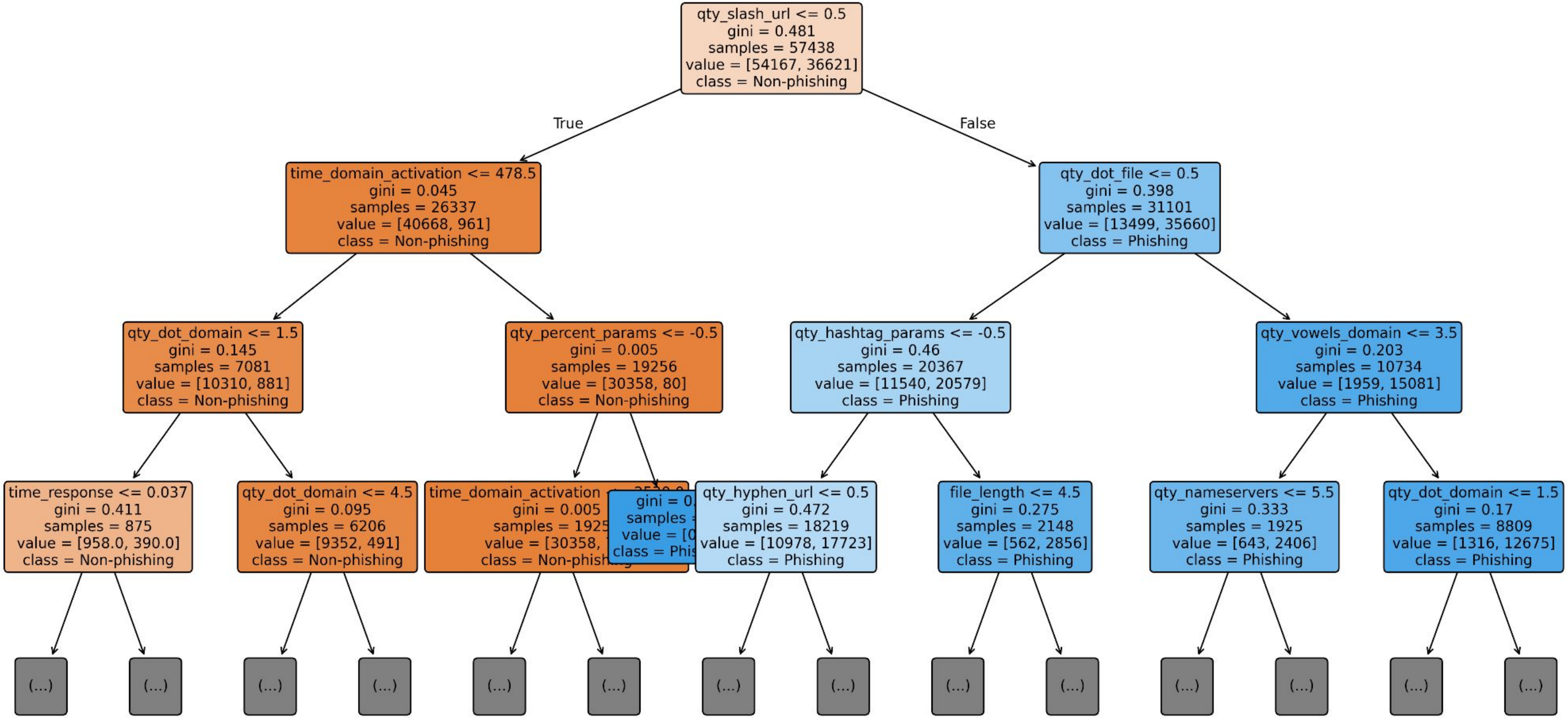
Description: Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes for classification or the mean prediction for regression.

How It Works:

- Multiple decision trees are generated using different subsets of data and features.
- Each tree votes on the outcome, and the majority decision is taken as the final prediction.
- It reduces overfitting by averaging out predictions and increases model robustness.

Random Forest Use Cases

1. Classifying phishing emails: Distinguish between legitimate and phishing emails by learning patterns across multiple features like text, sender behavior, and metadata.
2. Intrusion detection: Classify network activity as benign or malicious by analyzing traffic features.
3. User behavior analytics: Identify potential insider threats by classifying user activities as normal or suspicious.


qty_slash_url <= 0.5
gini = 0.481
samples = 57438
value = [54167, 36621]
class = Non-phishing
True
False
time_domain_activation <= 478.5
gini = 0.045
samples = 26337
value = [40668, 961]
class = Non-phishing
qty_dot_file <= 0.5
gini = 0.398
samples = 31101
value = [13499, 35660]
class = Phishing
qty_dot_domain <= 1.5
gini = 0.145
samples = 7081
value = [10310, 881]
class = Non-phishing
qty_percent_params <= -0.5
gini = 0.005
samples = 19256
value = [30358, 80]
class = Non-phishing
qty_hashtag_params <= -0.5
gini = 0.46
samples = 20367
value = [11540, 20579]
class = Phishing
qty_vowels_domain <= 3.5
gini = 0.203
samples = 10734
value = [1959, 15081]
class = Phishing
time_response <= 0.037
gini = 0.411
samples = 875
value = [958.0, 390.0]
class = Non-phishing
qty_dot_domain <= 4.5
gini = 0.095
samples = 6206
value = [9352, 491]
class = Non-phishing
time_domain_activation
gini = 0.005
class = Non-phishing
qty_hyphen_url <= 0.5
gini = 0.472
samples = 18219
value = [10978, 17723]
class = Phishing
file_length <= 4.5
gini = 0.275
samples = 2148
value = [562, 2856]
class = Phishing
qty_nameservers <= 5.5
gini = 0.333
samples = 1925
value = [643, 2406]
class = Phishing
qty_dot_domain <= 1.5
gini = 0.17
samples = 8809
value = [1316, 12675]
class = Phishing
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)

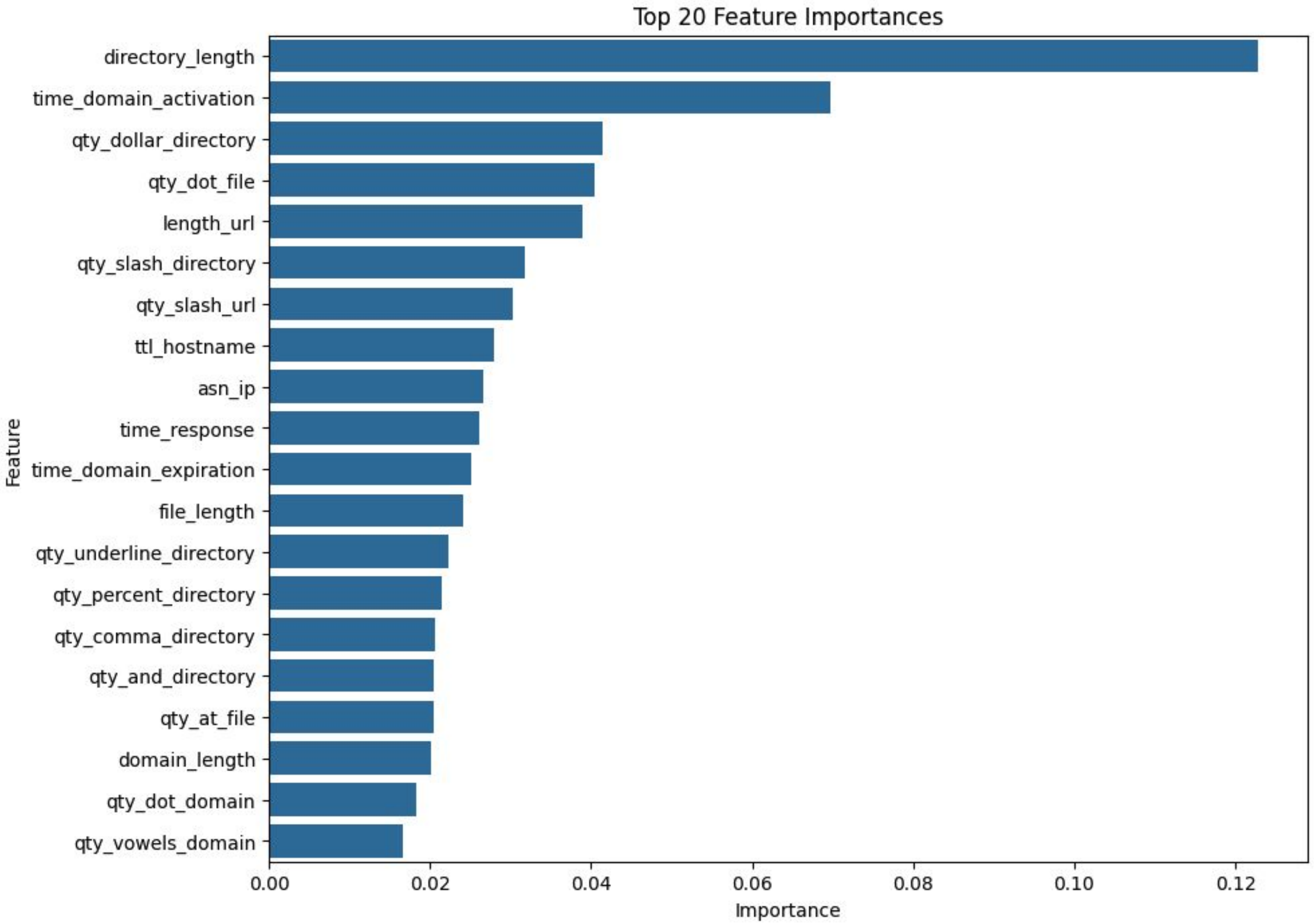
Random Forest

Python Notebook Example

```
[1]: import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
     from sklearn.tree import plot_tree
     import matplotlib.pyplot as plt
     import seaborn as sns

[2]: df = pd.read_csv('/Users/rob/Documents/Infosec Nashville/dataset_cybersecurity_michelle.csv')

[3]: # Step 1: Prepare the Data
     X = df.drop(columns=['phishing'])
     y = df['phishing']

[4]: # Split the data into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

[5]: # Step 2: Train a Random Forest Model
     rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
     rf_model.fit(X_train, y_train)

[5]: RandomForestClassifier
     RandomForestClassifier(random_state=42)

[6]: # Step 3: Evaluate the Model
     y_pred = rf_model.predict(X_test)

[7]: # Calculate accuracy
     accuracy = accuracy_score(y_test, y_pred)
     print(f'Accuracy: {accuracy:.2f}')

     Accuracy: 0.99

[8]: # Confusion matrix
     conf_matrix = confusion_matrix(y_test, y_pred)
     print(f'Confusion Matrix:\n{conf_matrix}')

     Confusion Matrix:
     [[22947   279]
      [  221 15463]]

[9]: # Classification report
     class_report = classification_report(y_test, y_pred)
     print(f'Classification Report:\n{class_report}')

     Classification Report:
                   precision    recall  f1-score   support

                0       0.99      0.99      0.99     23226
                1       0.98      0.99      0.98     15684

         accuracy                           0.99     38910
        macro avg       0.99      0.99      0.99     38910
     weighted avg       0.99      0.99      0.99     38910
```

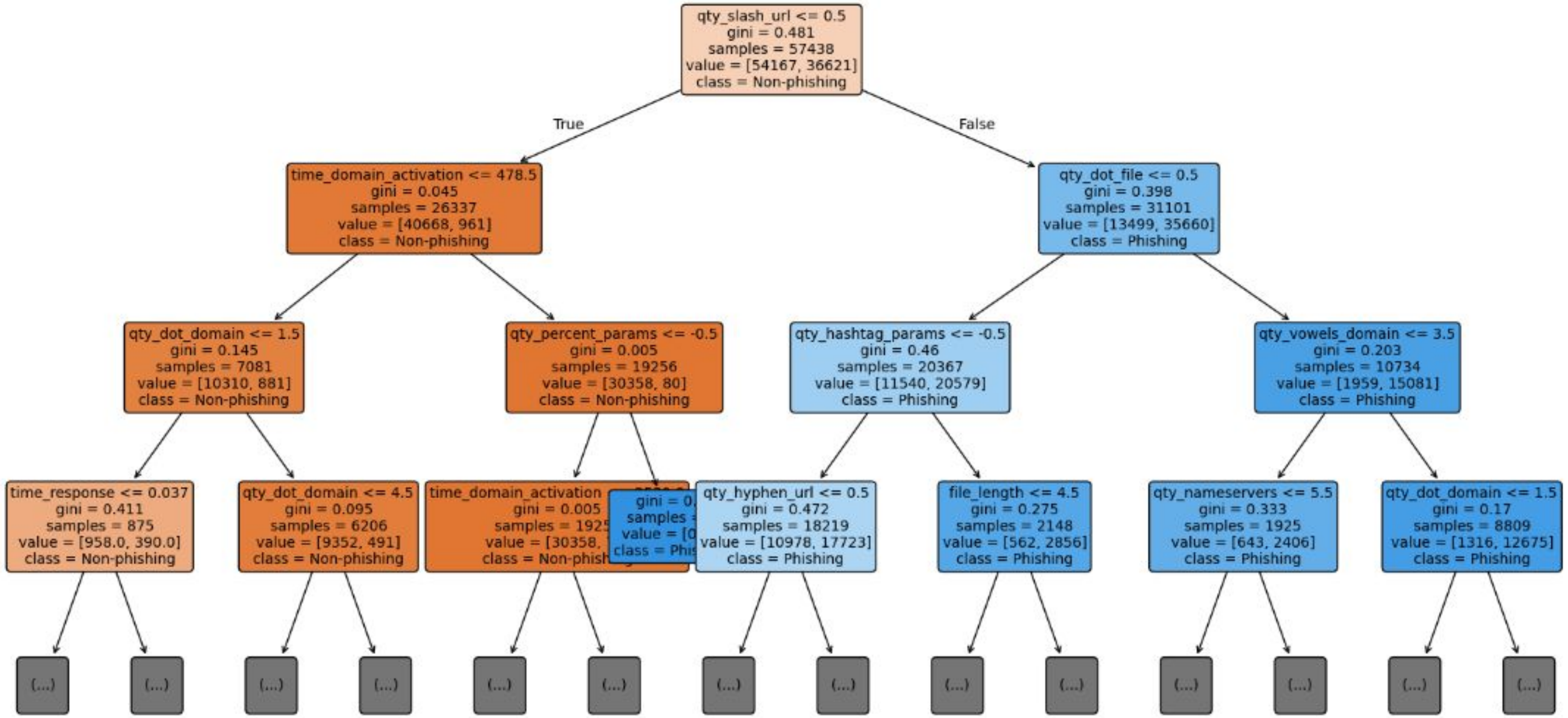
[10]:
```python
# Step 4: Feature Importance
feature_importances = rf_model.feature_importances_
feature_names = X.columns
importances_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances})
importances_df = importances_df.sort_values(by='Importance', ascending=False)
```

[11]:
```python
# Plot the feature importances
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=importances_df.head(20))
plt.title('Top 20 Feature Importances')
plt.show()
```



```
[12]: # Extract one tree from the Random Forest (e.g., the first tree)
estimator = rf_model.estimators_[0]
```

```
[13]: # Plot the tree using matplotlib
plt.figure(figsize=(20, 10))
plot_tree(estimator,
          feature_names=X.columns,
          class_names=['Non-phishing', 'Phishing'],
          filled=True,
          rounded=True,
          max_depth=3,  # Limiting depth for better visualization
          fontsize=10)
plt.savefig('random_forest_tree.png', dpi=300, bbox_inches='tight')
plt.show()
```



Resources

Book: The StatQuest Illustrated Guide To Machine Learning

YouTube: StatQuest with Josh Starmer

Datasets: Kaggle (https://www.kaggle.com/datasets/michellevp/dataset-phishing-domain-detection-cybersecurity)

Code Examples from Others: https://github.com/jivoi/awesome-ml-for-cybersecurity

Thank You

Robert Chapman
rschapman.com