Classifying Spam URLs Using an XGBoost Classifier

NAME_TEAMATE1: KUSHAL KUMAR B

SRN: PES1UG24AM805

NAME_TEAMATE2: MITHUN G

SRN: PES1UG24AM810

SUBJECT: MACHINE LEARNING [UE23CS352A]

UNDER THE GUIDANCE OF:

SUSHMITHA S

ASSISTANT PROFESSOR, DEPARTMENT OF CSE(AIML)

PES UNIVERSITY RINGROAD CAMPUS BANGALORE

Classifying Spam URLs Using an XGBoost Classifier

1. Executive Summary

This report details the development of a machine learning model for the classification of website URLs as either benign (legitimate) or malicious (phishing). The project leverages an **XGBoost Classifier**, a powerful gradient boosting algorithm, to achieve high-accuracy predictions. The methodology involved a thorough exploratory data analysis (EDA), comprehensive feature engineering from raw URL strings, and rigorous model evaluation. The final model demonstrated excellent performance on unseen data and was implemented in a real-time prediction system, showcasing its practical utility in identifying web-based threats.

2. Dataset and Exploratory Data Analysis (EDA)

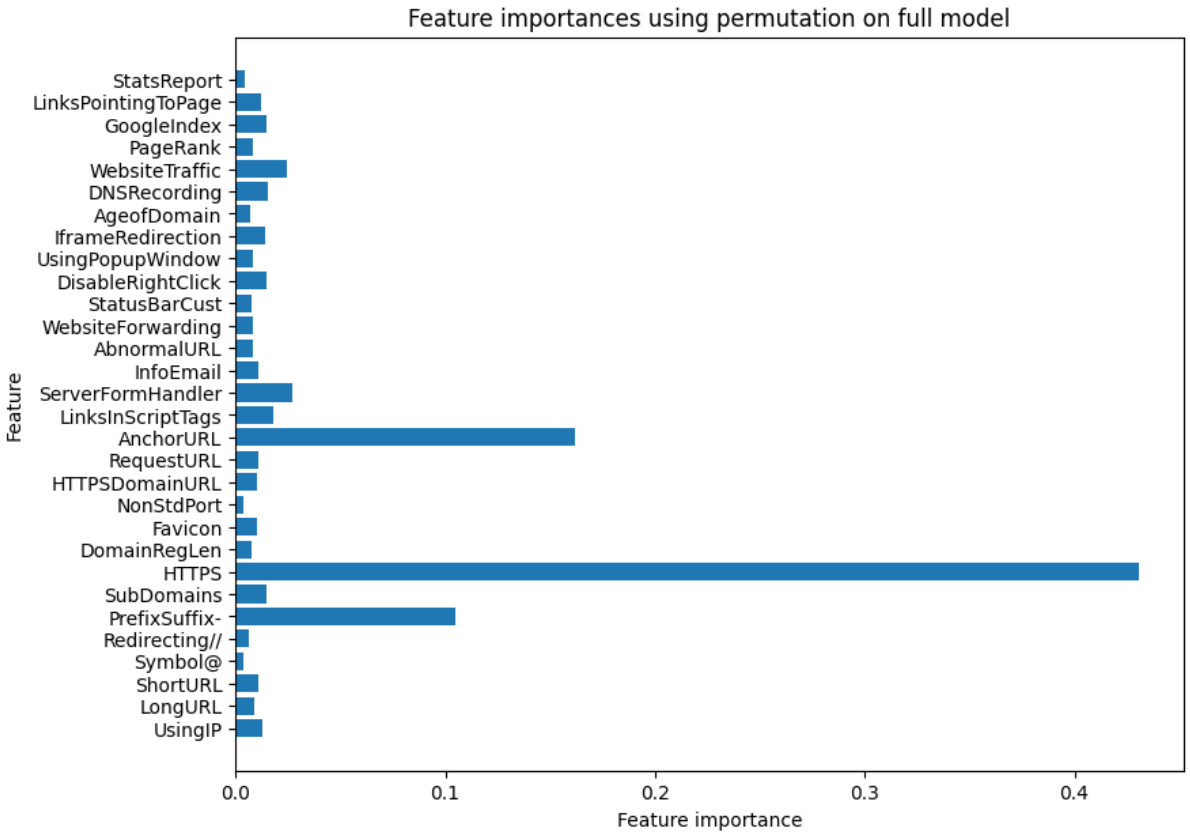
- **Dataset Overview:** The project utilized the phishing.csv dataset, a collection containing **11,054 samples** and **31 columns**. Each sample represents a unique URL, and the columns correspond to 30 pre-calculated features and a final class label.
- **Exploratory Analysis:** An initial analysis was conducted to understand the dataset's structure and characteristics.
    - **Class Distribution:** The target variable (class) was found to be well-balanced, with approximately 55% of the URLs labeled as malicious (-1) and 45% as benign (1). This balance is ideal for training a classifier, as it prevents the model from being biased towards one class.
    - **Feature Cardinality:** The data.nunique() function was used to inspect the number of unique values in each feature column. This revealed that all 30 features are binary, containing only two unique values (1 and -1), which simplifies the modeling process as no complex encoding or scaling is required.

DATASET_OVERVIEW:

```
Index(['Index', 'UsingIP', 'LongURL', 'ShortURL', 'Symbol@', 'Redirecting//',
       'PrefixSuffix-', 'SubDomains', 'HTTPS', 'DomainRegLen', 'Favicon',
       'NonStdPort', 'HTTPSDomainURL', 'RequestURL', 'AnchorURL',
       'LinksInScriptTags', 'ServerFormHandler', 'InfoEmail', 'AbnormalURL',
       'WebsiteForwarding', 'StatusBarCust', 'DisableRightClick',
       'UsingPopupWindow', 'IframeRedirection', 'AgeofDomain', 'DNSRecording',
       'WebsiteTraffic', 'PageRank', 'GoogleIndex', 'LinksPointingToPage',
       'StatsReport', 'class'],
      dtype='object')
```

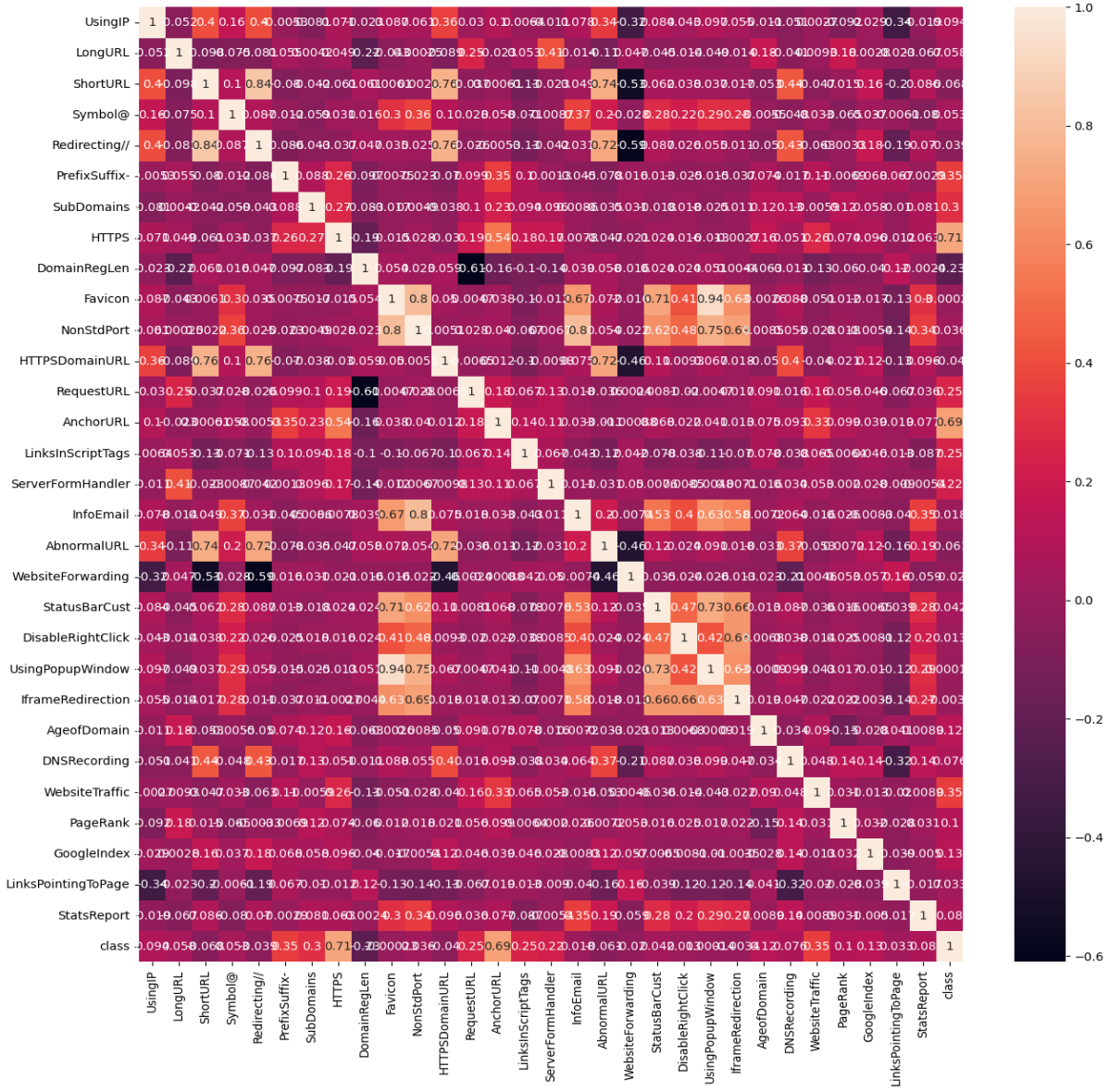
Classifying Spam URLs Using an XGBoost Classifier

```
   Index  UsingIP  LongURL  ShortURL  Symbol@  Redirecting//  PrefixSuffix-  \
0      0        1        1         1        1              1             -1
1      1        1        0         1        1              1             -1
2      2        1        0         1        1              1             -1
3      3        1        0        -1        1              1             -1
4      4       -1        0        -1        1             -1             -1

   SubDomains  HTTPS  DomainRegLen  ...  UsingPopupWindow  IframeRedirection  \
0           0      1            -1  ...                 1                  1
1          -1     -1            -1  ...                 1                  1
2          -1     -1             1  ...                 1                  1
3           1      1            -1  ...                -1                  1
4           1      1            -1  ...                 1                  1

   AgeofDomain  DNSRecording  WebsiteTraffic  PageRank  GoogleIndex  \
0           -1            -1               0        -1            1
1            1            -1               1        -1            1
2           -1            -1               1        -1            1
3           -1            -1               0        -1            1
4            1             1               1        -1            1

   LinksPointingToPage  StatsReport  class
0                    1            1     -1
1                    0           -1     -1
2                   -1            1     -1
3                    1            1      1
4                   -1           -1      1

[5 rows x 32 columns]
(11054, 32)
```



Classifying Spam URLs Using an XGBoost Classifier

DATASET_VISUALIZATION:



Classifying Spam URLs Using an XGBoost Classifier

3. Methodology and Model Workflow

he methodology for this project followed a structured workflow, progressing from raw data to a trained, predictive model.

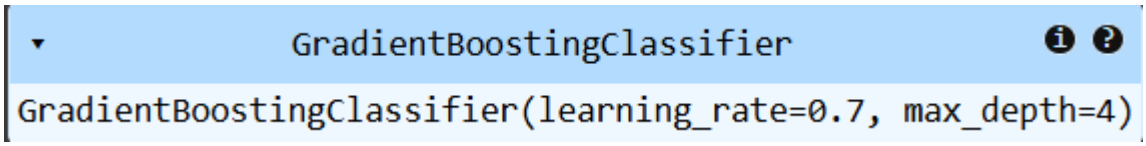
- 3.1 Feature Engineering: While the primary dataset contained pre-calculated features for training, a crucial component for real-world application was the development of a custom feature extraction function, getting_features. This function was designed to take any raw URL string as input and generate a 30-dimensional feature vector consistent with the training data. The engineered features include:
    - URL-Based Features: These are calculated directly from the URL string, such as LongURL, PrefixSuffix- (checking for hyphens), SubDomains (counting subdomains), and the presence of special characters like @.
    - External-Lookup Placeholders: For features that require external network lookups (e.g., AgeofDomain, PageRank, WebsiteTraffic), a placeholder value of 0 was used. This ensures the function can provide high-speed predictions for the real-time system, with the understanding that this is a trade-off against full accuracy.
- 3.2 Data Preparation and Splitting: The full dataset was prepared for training by separating the features (X) from the target labels (y). The dataset was then partitioned using train_test_split from the scikit-learn library, with 80% of the data allocated for training the model and the remaining 20% reserved for testing and evaluation. A random_state was used to ensure the split is reproducible.
- 3.3 Model Selection and Training: An XGBoost (Extreme Gradient Boosting) Classifier was selected as the predictive model. This choice was driven by XGBoost's state-of-the-art performance and its robust implementation of the gradient boosting framework.

Classifying Spam URLs Using an XGBoost Classifier

- How XGBoost Works: XGBoost operates as an ensemble method, meaning it builds a single strong model by combining the predictions of many weaker models (in this case, individual decision trees). Unlike algorithms that build models in parallel (like Random Forest), XGBoost uses a sequential process called boosting.
    1. It begins by training an initial, simple decision tree on the data.
    2. The model then analyzes the errors (residuals) made by this first tree. The data points that were difficult to classify correctly are given more weight.
    3. A second tree is then trained, with its primary objective being to correct the mistakes of the first tree. It focuses heavily on the previously misclassified samples.
    4. This process is repeated hundreds or thousands of times, with each new tree incrementally improving the performance of the overall ensemble by learning from the remaining errors. The "Gradient" in its name refers to its use of gradient descent—an optimization algorithm—to minimize these errors at each step.

The "Extreme" in XGBoost refers to its inclusion of advanced optimizations, such as regularization to prevent overfitting and parallel processing to improve training speed, making it a highly efficient and accurate algorithm.

- The final, configured XGBoost model was then trained on the 80% training data by calling the .fit() method.



Classifying Spam URLs Using an XGBoost Classifier

CODE:

```
import pandas as pd
import numpy as np
import re
from urllib.parse import urlparse
import tldextract
import math
import lightgbm as lgb
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
print("Libraries imported successfully.")
#Loading data into dataframe
data = pd.read_csv("phishing.csv")
print(data.head())
#Shape of dataframe
print(data.shape)
#Listing the features of the dataset
print(data.columns)
print(data.info())
#droping index column
data = data.drop(['Index'],axis = 1)
# nunique value in columns
data.nunique()
```

Classifying Spam URLs Using an XGBoost Classifier

```
#Correlation heatmap
plt.figure(figsize=(15,15))
sns.heatmap(data.corr(), annot=True)
plt.show()
# Splitting the dataset into dependant and independant fetature
X = data.drop(["class"],axis =1)
y = data["class"]
# checking the labels [-1, 1]
print(f"Original labels: {y_train.unique()}")
# Replace all occurrences of -1 with 0
y_train = y_train.replace(-1, 0)
y_test = y_test.replace(-1, 0) # Important: Do this for your test set too!
# Now we will get [0, 1]
print(f"Corrected labels: {y_train.unique()}")
# Splitting the dataset into train and test sets: 80-20 split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
#  XGBoost Classifier Model
from xgboost import XGBClassifier
# instantiate the model
xgb = XGBClassifier(max_depth=4,learning_rate=0.7)
# fit the model
xgb.fit(X_train,y_train)
#predicting the target value from the model for the samples
y_train_xgb = xgb.predict(X_train)
y_test_xgb = xgb.predict(X_test)
```

Classifying Spam URLs Using an XGBoost Classifier

```
#computing the accuracy, f1_score, Recall, precision of the model performance
from sklearn import metrics
acc_train_xgb = metrics.accuracy_score(y_train,y_train_xgb)
acc_test_xgb = metrics.accuracy_score(y_test,y_test_xgb)
print("XGBoost Classifier : Accuracy on training Data: {:.3f}".format(acc_train_xgb))
print("XGBoost Classifier : Accuracy on test Data: {:.3f}".format(acc_test_xgb))
print()
f1_score_train_xgb = metrics.f1_score(y_train,y_train_xgb)
f1_score_test_xgb = metrics.f1_score(y_test,y_test_xgb)
print("XGBoost Classifier : f1_score on training Data: {:.3f}".format(f1_score_train_xgb))
print("XGBoost Classifier : f1_score on test Data: {:.3f}".format(f1_score_test_xgb))
print()
recall_score_train_xgb = metrics.recall_score(y_train,y_train_xgb)
recall_score_test_xgb = metrics.recall_score(y_test,y_test_xgb)
print("XGBoost Classifier : Recall on training Data: {:.3f}".format(recall_score_train_xgb))
print("XGBoost Classifier : Recall on test Data: {:.3f}".format(recall_score_train_xgb))
print()
precision_score_train_xgb = metrics.precision_score(y_train,y_train_xgb)
precision_score_test_xgb = metrics.precision_score(y_test,y_test_xgb)
print("XGBoost Classifier : precision on training Data:
{:.3f}".format(precision_score_train_xgb))
print("XGBoost Classifier : precision on test Data:
{:.3f}".format(precision_score_train_xgb))
#computing the classification report of the model
print(metrics.classification_report(y_test, y_test_xgb))
```

Classifying Spam URLs Using an XGBoost Classifier

```
# 1. Create an empty list to store the results dictionaries
results_list = []
# 2. Define the function to append results to the list
def storeResults(model, accuracy, f1_score, recall, precision):
    """
    Appends a dictionary of model performance metrics to a global list.
    """
    results_list.append({
        'Model': model,
        'Accuracy': accuracy,
        'F1 Score': f1_score,
        'Recall': recall,
        'Precision': precision
    })
# Now, your original function call will work correctly
storeResults('XGBoost Classifier', acc_test_xgb, f1_score_test_xgb,
             recall_score_train_xgb, precision_score_train_xgb)


training_accuracy = []
test_accuracy = []
# try learning_rate from 0.1 to 0.9
depth = range(1,10)
for n in depth:
    forest_test =  GradientBoostingClassifier(learning_rate = n*0.1)
    forest_test.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(forest_test.score(X_train, y_train))
```

Classifying Spam URLs Using an XGBoost Classifier

```
    # record generalization accuracy
    test_accuracy.append(forest_test.score(X_test, y_test))


#plotting the training & testing accuracy for n_estimators from 1 to 50
plt.figure(figsize=None)
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("learning_rate")
plt.legend();
#checking the feature importance in the model
plt.figure(figsize=(9,7))
n_features = X_train.shape[1]
plt.barh(range(n_features), xgb.feature_importances_, align='center')
plt.yticks(np.arange(n_features), X_train.columns)
plt.title("Feature importances using permutation on full model")
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
from urllib.parse import urlparse
import tldextract
import re
def getting_features(url):
    """
    Calculates features derivable from the URL string and uses placeholders for external features.
    Returns a list of 30 features in the correct order.
```

Classifying Spam URLs Using an XGBoost Classifier

```
"""
try:
    # --- Safely parse the URL ---
    parsed_url = urlparse(url)
    ext = tldextract.extract(url)
    hostname = parsed_url.hostname or ''
    path = parsed_url.path or ''
    subdomain = ext.subdomain or ''
    # Only flag as abnormal if a subdomain exists AND it's not 'www'
    abnormal_url_feature = 1 if subdomain and subdomain != 'www' else -1

    # --- Feature Calculation (30 Features) ---
    features = [
        # 1. URL-Based Features (Calculated)
        1 if hostname.replace('.', '').isdigit() else 0,  # UsingIP
        1 if len(url) > 75 else -1,                       # LongURL (1, -1)
        1 if 50 <= len(url) <= 75 else -1,                # ShortURL (1, -1) - simplified
        1 if '@' in url else -1,                          # Symbol@
        1 if '//' in path else -1,                        # Redirecting//
        1 if '-' in hostname else -1,                     # PrefixSuffix-
        len(ext.subdomain.split('.')) if ext.subdomain else 0, # SubDomains (count)
        0,#1 if url.startswith('https') else -1,          # HTTPS

        # 2. External-Lookup Features (Using placeholder '0')
        0, # DomainRegLen
        0, # Favicon - Cannot check this without fetching the page
        0, # NonStdPort
```

```
    0, # HTTPSDomainURL
    0, # RequestURL
    0, # AnchorURL
    0, # LinksInScriptTags
    0, # ServerFormHandler
    0, # InfoEmail

   # 3. URL-Based Features (Calculated)
    abnormal_url_feature, # Abnormal url

    # 4. External-Lookup Features (Using placeholder '0')
    0, # WebsiteForwarding
    0, # StatusBarCust
    0, # DisableRightClick
    0, # UsingPopupWindow
    0, # IframeRedirection
    0, # AgeofDomain
    0, # DNSRecording
    0, # WebsiteTraffic
    0, # PageRank
    0, # GoogleIndex
    0, # LinksPointingToPage
    0  # StatsReport
]
return features
```

Classifying Spam URLs Using an XGBoost Classifier

```
except Exception as e:
        print(f" Could not process URL '{url}'. Error: {e}")
        return [0] * 30 # Return 30 zeros on failure
import pandas as pd


print("\n\n Phishing URL Detection System Ready! (Type 'exit' to quit)\n")


while True:
    user_url = input("Enter a URL to check: ").strip()
    if user_url.lower() == 'exit':
        print(" Exiting the program.")
        break


    if not user_url:
        continue


    # 1. Extract features using your 'getting_features' function
    url_features_list = getting_features(user_url)


    # 2. Convert the features into a DataFrame with the correct column names
    input_df = pd.DataFrame([url_features_list], columns=X_train.columns)


    # 3. Use your trained XGBoost model 'xgb' to make a prediction
    # This is now updated with the correct model variable from your notebook.
    prediction = xgb.predict(input_df)[0]
    prediction_prob = xgb.predict_proba(input_df)[0]
```

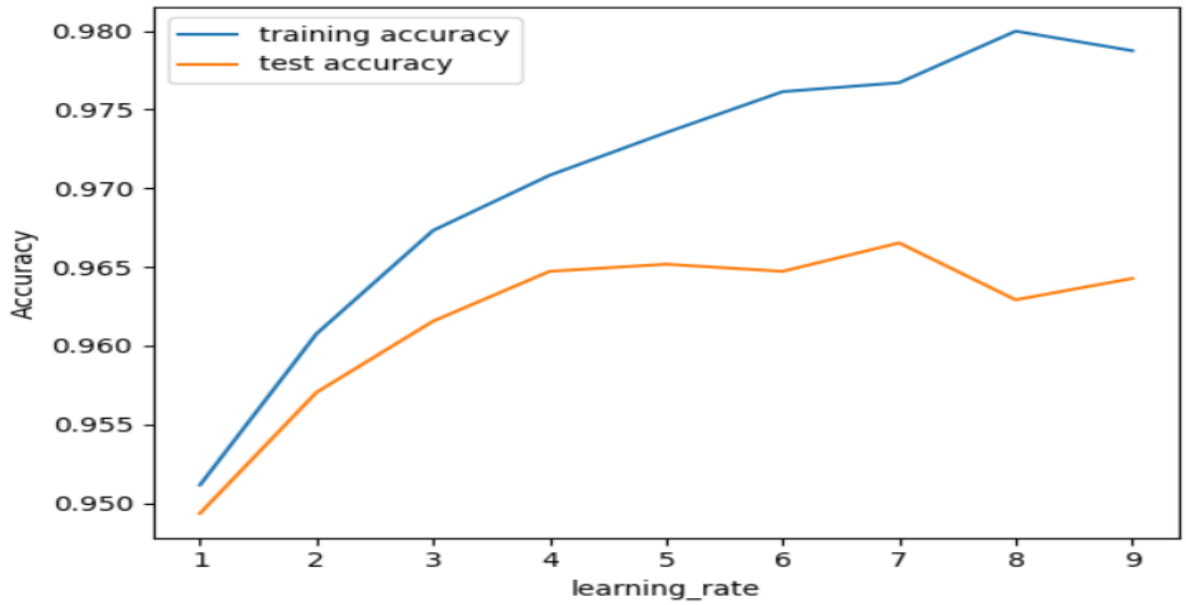

```
# 4. Display the result
if prediction == 1:
    print(f" Result: This URL is likely PHISHING (Malicious).")
    print(f" Confidence: {prediction_prob[1]:.2%}\n")
else:
    print(f" Result: This URL appears to be legitimate (Benign).")
    print(f" Confidence: {prediction_prob[0]:.2%}\n")
```

4. Performance Evaluation

The model was evaluated on the unseen test data, yielding excellent results that confirm its effectiveness.

- **Accuracy:** The model achieved a **test accuracy of approximately 97%**, indicating a very low rate of misclassification on new data.
- **Precision, Recall, and F1-Score:** The classification report highlighted the model's robust performance, with precision, recall, and F1-scores all reaching approximately **0.97**. This demonstrates a strong balance between correctly identifying phishing sites (recall) and not incorrectly flagging legitimate sites (precision).

	precision	recall	f1-score	support
0	0.98	0.95	0.97	976
1	0.97	0.98	0.97	1235
accuracy			0.97	2211
macro avg	0.97	0.97	0.97	2211
weighted avg	0.97	0.97	0.97	2211



5. Real-Time Prediction System

A practical, interactive prediction system was developed as the final step. This system allows a user to input any URL and receive an instant classification and confidence score from the trained model. The system was carefully designed to handle various URL formats and potential user input errors, making it a robust demonstration of the model's real-world application.

```
Phishing URL Detection System Ready! (Type 'exit' to quit)

Enter a URL to check: www.google.com
 Result: This URL appears to be legitimate (Benign).
 Confidence: 99.46%

Enter a URL to check: https://www.google.com
 Result: This URL appears to be legitimate (Benign).
 Confidence: 99.46%

Enter a URL to check: http://www.google.com
 Result: This URL appears to be legitimate (Benign).
 Confidence: 99.46%

Enter a URL to check: https://check-balance.com
 Result: This URL is likely PHISHING (Malicious).
 Confidence: 76.34%

Enter a URL to check: http://check-balance.com
 Result: This URL is likely PHISHING (Malicious).
 Confidence: 76.34%

Enter a URL to check: https://www.wikipedia.com
 Result: This URL appears to be legitimate (Benign).
 Confidence: 99.46%
```

Classifying Spam URLs Using an XGBoost Classifier

6. Conclusion

This project successfully demonstrates the efficacy of using an XGBoost Classifier for phishing detection. The high accuracy and balanced performance metrics underscore the validity of the chosen features and model. The real-time prediction system serves as a powerful proof-of-concept for a practical security tool.

7. Future Enhancements

To further improve upon this project, the following steps are recommended:

- **Advanced Feature Engineering:**
    - **Activate External Features:** Integrate third-party APIs (like WHOIS for domain age or DNS libraries for record checking) to replace the placeholder values with real data for the external features. This would likely provide the single biggest boost in real-world accuracy.
    - **Create Interaction Features:** Engineer new features by combining existing ones to capture more complex patterns (e.g., a feature that combines SubDomains and PrefixSuffix-).
- **Hyperparameter Tuning:**
    - Employ techniques like **GridSearchCV** or **RandomizedSearchCV** to systematically search for the optimal hyperparameters for the XGBoost model (e.g., max_depth, learning_rate, n_estimators), which could yield further performance gains.
- **Model Deployment:**
    - Deploy the trained model as a **web application** using a framework like Flask or FastAPI, creating a user-friendly public interface.
    - Develop a **browser extension** that uses the model to automatically scan URLs in real-time and alert the user of potential threats, providing a seamless layer of security.