PhishGuard - Detection of URL phishing using machine learning

Vyom Raval
22BCE301
Batch:- E2
Institute of Technology, Nirma University
Ahmedabad, India
22bce301@nirmauni.ac.in

Sumay Ramani
22BCE295
Batch:- E2
Institute of Technology, Nirma University
Ahmedabad, India
22bce295@nirmauni.ac.in

***Abstract*—Unlike phishing, it is one of the most common cyber attacks that fool users into revealing confidential information. Using machine learning models, the project detected phishing websites based on URL features, which achieved good accuracy using XGBoost. Future work would be developing browser extensions or GUIs to improve usability.**

***Index Terms*—Phishing detection, Machine learning, URL analysis, XGBoost, Cybersecurity,Random Forest, ANN,Decision Tree**

I. Introduction

Phishing remains one of the most widespread cybersecurity threats today, using psychological manipulation techniques to trick users into revealing confidential information. Unlike malware that targets technical vulnerabilities, phishing exploits human tendencies toward trust and quick action, making it particularly difficult to defend against through technical means alone. This project applies multiple machine learning algorithms to distinguish between legitimate websites and phishing sites by analyzing key features extracted from URLs, domain information, and webpage content. The dataset was sourced from Kaggle, while model development and training were performed using Google Colab, a cloud-based platform that provides access to computational resources including CPUs and GPUs. Google Colab serves as an Infrastructure as a Service (IaaS) solution, eliminating the need for local high-performance hardware.

II. Motive and Objective

A. Motive

- To address the growing threat of phishing attacks that compromise user data and security.
- To enhance cybersecurity awareness and provide tools to detect malicious websites effectively.
- To leverage advanced machine learning techniques and Cloud Computing for proactive phishing detection.

B. Objective

- Develop a dataset comprising legitimate and phishing URLs from reliable open-source platforms.
- Extract and analyze key features such as URL structure, domain attributes, and webpage content.
- The project involves training and assessing multiple machine learning algorithms, including Random Forest, Support Vector Machines (SVM), and XGBoost, to achieve precise and reliable classification results. These models were selected for their proven effectiveness in similar classification tasks and their ability to handle the complex feature relationships present in web security data.
- Identify the best-performing model based on accuracy metrics and suggest its application in real-world scenarios.

III. Dataset Understanding

This project includes five data files, which are used to detect phishing websites effectively:

- **Benign_list_big_final**: A comprehensive dataset of benign URLs used as the initial reference for legitimate websites. A glimpse of the dataset is shown in Figure 1.
- **online-valid**: A dataset containing various URLs, including phishing and legitimate entries. A sample of this dataset is displayed in Figure 13.
- **legitimate**: A derived dataset containing legitimate URLs, created after feature extraction from the first two files. See Figure 3 for a preview.
- **phishing**: A derived dataset containing phishing URLs, also created after feature extraction from the first two files. Figure 4 shows an example.
- **urldata**: A final dataset that combines random samples of phishing and legitimate URLs, ensuring balanced representation for model training and evaluation. Figure 5 provides a glimpse of this dataset.

Initially, the first two files (**Benign_list_big_final** and **online-valid**) were studied in detail. Feature extraction techniques were applied to distinguish legitimate and phishing websites. This process resulted in the creation of two new datasets: **legitimate** and **phishing**. Finally, random samples from these two datasets were combined to form the **urldata** file, which serves as the primary input for the machine learning models.

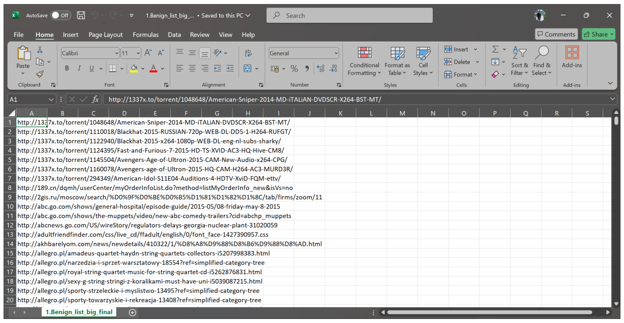

Fig. 1. Sample of the **Benign_list_big_final** dataset.

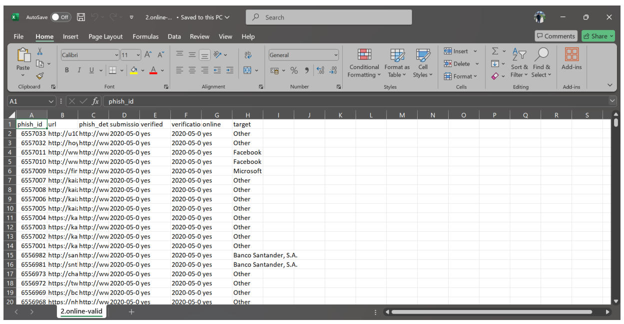

Fig. 2. Sample of the **online-valid** dataset.

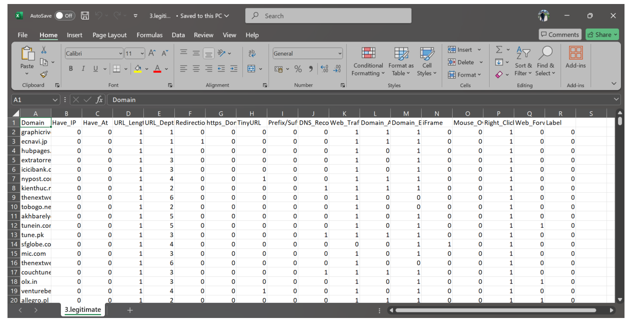

Fig. 3. Sample of the **legitimate** dataset.

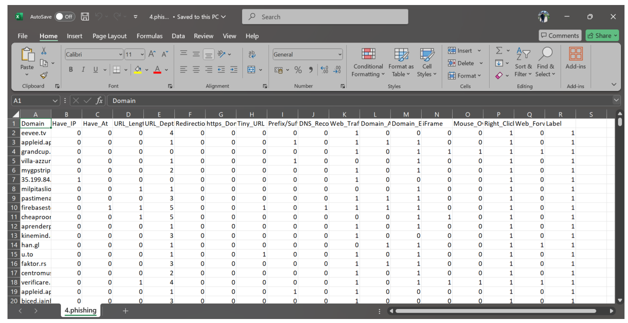

Fig. 4. Sample of the **phishing** dataset.

IV. Methodology

The methodology of this study outlines the development and evaluation of a machine learning model for detecting whether a URL is legitimate or phished, I'll rewrite that passage about the features and computing resources: The classification relies on a carefully selected set of 13 features extracted from URL properties and behavioral patterns. Due to the substantial size of the dataset and the computational demands of the algorithms, Google Colab's GPU capabilities were utilized instead of standard CPU processing. This implementation of cloud-based GPU resources exemplifies the practical benefits of Infrastructure as a Service (IaaS), enabling complex analysis without requiring specialized local hardware.

A. Data Collection and Preprocessing

- **Dataset Source:** The dataset was obtained from [mention source, e.g., public datasets, proprietary data, or web scraping].
- **Features:** The dataset included 13 features such as URL length, presence of special characters, domain age, SSL certificate validity, and WHOIS information.
- **Data Cleaning:**
  - Duplicate entries were removed.
  - Missing values were addressed through [imputation strategies or feature removal if sparsity was significant].
- **Encoding Categorical Data:** Features like domain registration details were encoded using [label encoding, one-hot encoding, etc.].

B. Feature Engineering

- **Feature Extraction:** URL-based features were extracted, such as:
  - Structural attributes (length, presence of IP addresses).
  - Lexical attributes (subdomain count, suspicious keywords).
  - Behavioral indicators (domain traffic rank, click-through patterns).
- **Correlation Analysis:** Correlation matrices and statistical methods identified features contributing most to phishing detection.

C. Model Selection

Several machine learning algorithms were evaluated to identify the optimal model for URL classification:

- Logistic Regression
- Support Vector Machines (SVM)
- Decision Tree Classifier
- XGBoost Classifier
- MLPClassifier
- Random Forest
- ANN(Keras)
- Deep Learning Models (e.g., Multi-Layer Perceptrons)

Hyperparameter tuning was performed using Grid Search or Bayesian Optimization.

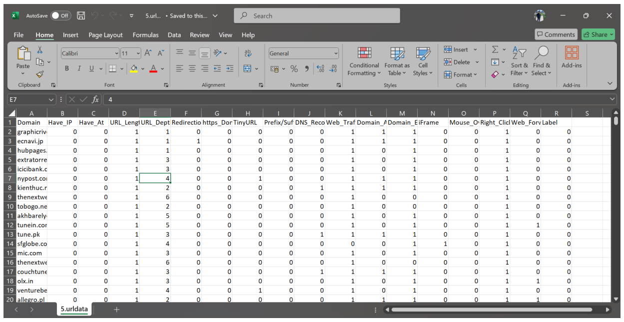

Fig. 5. Sample of the final **urldata** dataset.

D. Data Partitioning and Validation

- The dataset was divided into training (80 percent) and testing (20 percent) portions to ensure proper model evaluation on unseen data.
- Stratified K-Fold Cross-Validation (e.g., 5-fold) ensured class balance during training and validation.

E. Evaluation Metrics

To evaluate model performance, the following metrics were employed:

- **Accuracy:** To assess overall correctness of predictions.
- model performance is evaluated using a comprehensive set of metrics: Precision assesses the accuracy of positive predictions, Recall measures the ability to detect all actual phishing sites, and the F1-Score provides a balanced metric that combines both precision and recall. This multi-metric approach ensures the model effectively identifies malicious URLs while minimizing both false alarms and missed detections
- **Area Under the ROC Curve (AUC-ROC):** To evaluate the trade-off between true positives and false positives.

F. Deployment

After successful development of the model, the model has to be deployed somewhere so that it can be useful for someone else also. So we have to use the Platform as a Service (PaaS) feature of the Cloud Computing where we can deploy our application using the resources of the coloud.

Render will be used as PaaS to deploy the service.

G. Implementation and Real-World Validation

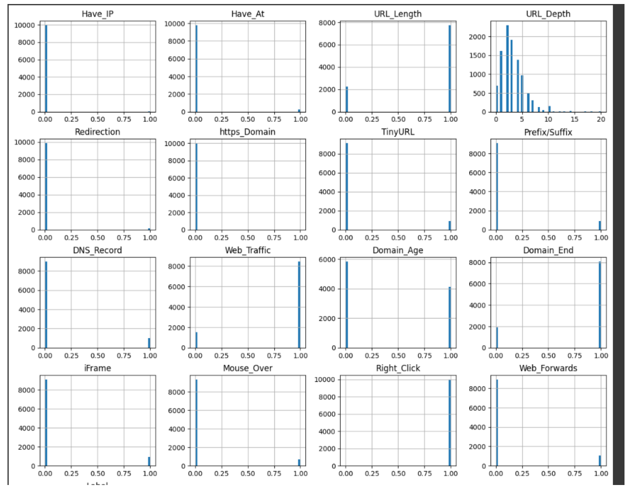

Fig. 6. histogram of the extracted features from the dataset.

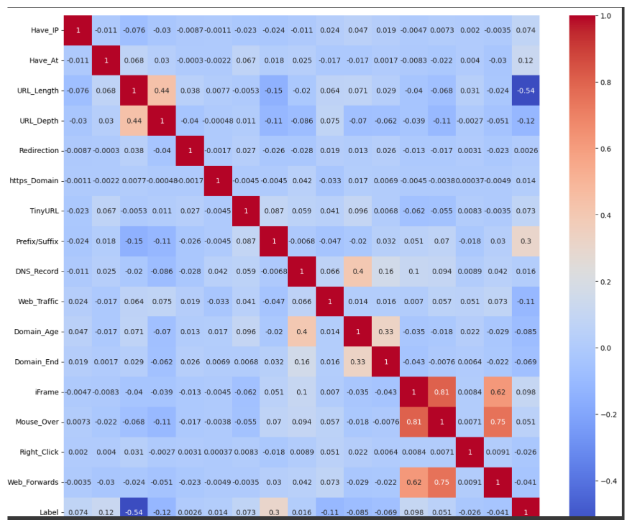

Fig. 7. Corelateion matrix of the features.

```
<ipython-input-11-43592b8f1cd8>:43: FutureWarning: The behavior of DataFram
  results = pd.concat([results, model_results], ignore_index=True)
                 Model  Accuracy  Precision  Recall  F1-Score
0        Decision Tree     0.816      0.978   0.649     0.780
1        Random Forest     0.864      0.930   0.789     0.854
2  Logistic Regression     0.810      0.939   0.667     0.780
3                  SVM     0.804      0.968   0.632     0.765
4  K-Nearest Neighbors     0.849      0.894   0.794     0.841
```

Fig. 8. R2 scores of the models.

```
[ ] from sklearn.metrics import accuracy_score

    # making predictions
    y_pred_test=decisionTree.predict(x_test)
    y_pred_train=decisionTree.predict(x_train)

    accu_train = accuracy_score(y_train,y_pred_train)
    accu_test = accuracy_score(y_test,y_pred_test)
    print("Decision Tree: Accuracy on training Data: {:.3f}".format(accu_train))
    print("Decision Tree: Accuracy on test Data: {:.3f}".format(accu_test))

    Decision Tree: Accuracy on training Data: 0.813
    Decision Tree: Accuracy on test Data: 0.816
```

Fig. 9. Decision tree

```python
from sklearn.metrics import accuracy_score
y_pred_test_rf = random_forest.predict(x_test)
y_pred_train_rf = random_forest.predict(x_train)

accu_train_rf = accuracy_score(y_train, y_pred_train_rf)
accu_test_rf = accuracy_score(y_test, y_pred_test_rf)

print("Random Forest: Accuracy on training Data: {:.3f}".format(accu_train_rf))
print("Random Forest: Accuracy on test Data: {:.3f}".format(accu_test_rf))
```

```
Random Forest: Accuracy on training Data: 0.867
Random Forest: Accuracy on test Data: 0.864
```

Fig. 10. Random Forest

```
MLPClassifier
MLPClassifier(hidden_layer_sizes=(100, 100), max_iter=500, random_state=42)
```

```python
from sklearn.metrics import accuracy_score

y_pred_test_mlp = mlp_classifier.predict(x_test)
y_pred_train_mlp = mlp_classifier.predict(x_train)

accu_train_mlp = accuracy_score(y_train, y_pred_train_mlp)
accu_test_mlp = accuracy_score(y_test, y_pred_test_mlp)

print("MLP Classifier: Accuracy on training Data: {:.3f}".format(accu_train_mlp))
print("MLP Classifier: Accuracy on test Data: {:.3f}".format(accu_test_mlp))
```

```
MLP Classifier: Accuracy on training Data: 0.855
MLP Classifier: Accuracy on test Data: 0.859
```

Fig. 11. MLP classifier

```python
from sklearn.metrics import accuracy_score

y_pred_test_xgb = xgboost_classifier.predict(x_test)
y_pred_train_xgb = xgboost_classifier.predict(x_train)

accu_train_xgb = accuracy_score(y_train, y_pred_train_xgb)
accu_test_xgb = accuracy_score(y_test, y_pred_test_xgb)

print("XGBoost Classifier: Accuracy on training Data: {:.3f}".format(accu_train_xgb))
print("XGBoost Classifier: Accuracy on test Data: {:.3f}".format(accu_test_xgb))
```

```
XGBoost Classifier: Accuracy on training Data: 0.866
XGBoost Classifier: Accuracy on test Data: 0.863
```

Fig. 12. XGBoost classifier

```
df1.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Domain         5000 non-null   object
 1   Have_IP        5000 non-null   int64
 2   Have_At        5000 non-null   int64
 3   URL_Length     5000 non-null   int64
 4   URL_Depth      5000 non-null   int64
 5   Redirection    5000 non-null   int64
 6   https_Domain   5000 non-null   int64
 7   TinyURL        5000 non-null   int64
 8   Prefix/Suffix  5000 non-null   int64
 9   DNS_Record     5000 non-null   int64
 10  Web_Traffic    5000 non-null   int64
 11  Domain_Age     5000 non-null   int64
 12  Domain_End     5000 non-null   int64
 13  iFrame         5000 non-null   int64
 14  Mouse_Over     5000 non-null   int64
 15  Right_Click    5000 non-null   int64
 16  Web_Forwards   5000 non-null   int64
 17  Label          5000 non-null   int64
dtypes: int64(17), object(1)
memory usage: 703.2+ KB
```

Fig. 13. Features and their data-types

```
csv1_sample = df1.head(3000)
csv2_sample = df2.head(3000)

combined_csv = pd.concat([csv1_sample, csv2_sample])

combined_csv.to_csv('combined_file.csv', index=False)

combined_csv.info()

<class 'pandas.core.frame.DataFrame'>
Index: 6000 entries, 0 to 2999
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Domain         6000 non-null   object
 1   Have_IP        6000 non-null   int64
 2   Have_At        6000 non-null   int64
 3   URL_Length     6000 non-null   int64
 4   URL_Depth      6000 non-null   int64
 5   Redirection    6000 non-null   int64
 6   https_Domain   6000 non-null   int64
 7   TinyURL        3000 non-null   float64
 8   Prefix/Suffix  6000 non-null   int64
 9   DNS_Record     6000 non-null   int64
 10  Web_Traffic    6000 non-null   int64
 11  Domain_Age     6000 non-null   int64
 12  Domain_End     6000 non-null   int64
 13  iFrame         6000 non-null   int64
 14  Mouse_Over     6000 non-null   int64
 15  Right_Click    6000 non-null   int64
 16  Web_Forwards   6000 non-null   int64
 17  Label          6000 non-null   int64
 18  Tiny_URL       3000 non-null   float64
dtypes: float64(2), int64(16), object(1)
memory usage: 937.5+ KB
```

Fig. 14. Merging two csv files to make the training data for the ANN model.

```python
# Input layer
model.add(Dense(units=16, activation='relu', input_dim=X_train.shape[1], kernel_regularizer=l2(0.01)))

# l2 ane dropout
model.add(Dense(units=16, activation='relu', kernel_regularizer=l2(0.01)))
model.add(Dropout(0.5))  # Drop 50% of the units randomly

model.add(Dense(units=8, activation='relu', kernel_regularizer=l2(0.01)))
model.add(Dropout(0.5))

# true or false derive karva mate
model.add(Dense(units=1, activation='sigmoid'))
```

Fig. 15. adding multiple layers to the ANN model.

```python
# true or false derive karva mate
model.add(Dense(units=1, activation='sigmoid'))


model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model with early stopping
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=1, callbacks=[early_stopping])

evaluation = model.evaluate(X_test, y_test)

# Print the evaluation results
print(f"Test Loss: {evaluation[0]}")
```

Fig. 16. adding the final layer and printing the output of the ANN model.

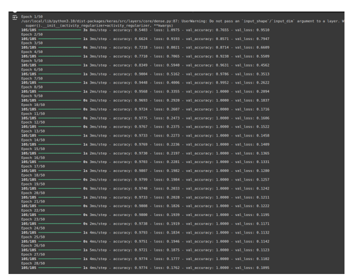

Fig. 17. Epoch output

```
105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9793 - loss: 0.1555 - val_accuracy: 1.0000 - val_loss: 0.0931
Epoch 47/50
105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9801 - loss: 0.1512 - val_accuracy: 1.0000 - val_loss: 0.0921
Epoch 48/50
105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9785 - loss: 0.1548 - val_accuracy: 1.0000 - val_loss: 0.0906
Epoch 49/50
105/105 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9852 - loss: 0.1431 - val_accuracy: 1.0000 - val_loss: 0.0899
Epoch 50/50
105/105 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9800 - loss: 0.1494 - val_accuracy: 1.0000 - val_loss: 0.0893
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0904
Test Loss: 0.09010007232427597
```

Fig. 18. Final output of the last epoch.