ISSN NO: 9726-001X
Volume 13 Issue 02 2025

JOURNAL OF CURRENT SCIENCE

MACHINE LEARNING-POWERED CLIENT SIDE PROTECTION AGAINST WEB SPOOFING ATTACKS

KOREY SHASHANK
UG Student,
Department of CSE,
St. Martin's Engineering College,
Secunderabad, Telangana, India
shashankkorey@gmail.com

E. SOUMYA
Assistant Professor,
Department of CSE,
St. Martin's Engineering College,
Secunderabad, Telangana, India
esoumyait@smec.ac.in

Abstract- Web spoofing attacks represent a critical threat to the security and integrity of online communication and transactions. These attacks involve malicious actors impersonating legitimate websites to deceive users into revealing sensitive information or unwittingly engaging in harmful actions. To effectively address this growing threat, there is an urgent need for robust defense mechanisms capable of identifying and thwarting web spoofing attempts in real-time. Current approaches to combating web spoofing primarily rely on server-side defenses, such as SSL/TLS protocols and domain validation techniques. While these methods provide some level of protection, they suffer from significant limitations. Firstly, server-side defenses are reactive, meaning they can only detect and respond to spoofing attempts after they have occurred. This delay leaves users vulnerable during the critical window between the initiation of the attack and its detection. Moreover, server-side defenses may struggle to accurately differentiate between legitimate and spoofed websites, leading to false positives and negatives. The prevalence of web spoofing attacks underscores the need for proactive client-side defense mechanisms. Existing approaches predominantly focus on serverside defenses, which are insufficient in providing timely and reliable protection against evolving spoofing techniques. Thus, there exists a critical gap in the current security infrastructure that necessitates addressing to effectively combat web spoofing threats. we aim to provide users with a proactive and robust solution capable of detecting and preventing spoofing attempts before they inflict harm. Additionally, by harnessing machine learning techniques, we aim to develop a system capable of continuously learning and adapting to emerging spoofing tactics, thereby enhancing its effectiveness and resilience against evolving threats. The proposed PISHCATCHER system represents a novel approach to combat web spoofing attacks. It is a machine learning based defense mechanism designed to operate at the client-side, enabling real-time detection and prevention of spoofing attempts. By analyzing various features extracted from web pages, including HTML structure, CSS styles, and JavaScript behavior, PISHCATCHER employs advanced machine learning algorithms to accurately distinguish between legitimate and spoofed websites.

I. INTRODUCTION

The Web spoofing attacks pose a significant threat to online security by allowing malicious actors to impersonate legitimate websites, deceiving users into divulging sensitive information or engaging in harmful actions. To combat this threat effectively, robust defense mechanisms are essential to identify and thwart spoofing attempts in real-time. While current approaches primarily rely on server-side defenses like SSL/TLS protocols and domain validation techniques, they suffer from limitations such as 5reactivity and difficulty in accurately differentiating between legitimate and spoofed websites. To address these challenges, the proposed PISHCATCHER system offers a novel approach to combat web spoofing attacks. Operating at the client-side, PISHCATCHER leverages machine learning techniques to analyze various features extracted from web pages, including HTML structure, CSS styles, and JavaScript behavior. By employing advanced machine learning algorithms, PISHCATCHER accurately distinguishes between legitimate and spoofed websites, enabling real-time detection and prevention of spoofing attempts.

II. RELATED WORK

[1] R. Kumar (2023)
Web spoofing attacks have become increasingly common with the rise of phishing and fraudulent websites. This study presents a client-side defense mechanism using machine learning techniques, specifically Random Forest and Support Vector Machines (SVM), to classify malicious URLs. The model achieved 96% accuracy on a dataset of real-world phishing websites.

[2] S. Patel (2023)
This paper introduces a novel approach for web spoofing detection using Convolutional Neural Networks (CNNs). By analyzing webpage screenshots and comparing them to legitimate references, the model identifies visual inconsistencies. The proposed system achieved 94% accuracy, outperforming traditional blacklist-based methods.

[3] A. Gupta (2023)
With the growing sophistication of phishing websites, this research explores the use of Natural Language Processing (NLP) for URL and content-based spoofing detection. The model employs a Bi-LSTM network to analyze text patterns, achieving 92.5% accuracy in identifying fake websites.

[4] M. Rahman (2022)
This work presents a browser extension that uses a machine learning-based classifier to detect web spoofing attacks. The system extracts features such as URL patterns, HTML structure, and JavaScript behavior, using Gradient Boosting to achieve 95% detection accuracy.

[5] L. Zhang (2022)
This study proposes a hybrid approach combining deep learning and heuristics for web spoofing defense. By integrating CNNs with rule-based filtering, the system effectively detects phishing sites and malicious content, reducing false positives by 18%.

[6] T. Brown (2021)
This paper focuses on web spoofing prevention using Reinforcement Learning (RL). The model dynamically adjusts its detection parameters based on the browsing behavior of users, improving adaptability and accuracy. It achieved a detection rate of 91% on real-world spoofing datasets.

[7] H. Lee (2021)
To combat phishing and spoofing attacks, this research employs an ensemble learning model combining Decision Trees and K-Nearest Neighbors (KNN). The hybrid model outperforms single classifiers, achieving 97% accuracy on malicious website datasets.

[8] F. Khan (2020)
This paper explores the use of Generative Adversarial Networks (GANs) to generate synthetic spoofing samples, improving the robustness of machine learning classifiers. The model enhances the accuracy of existing spoofing detection systems by 12%.

[9] J. Silva (2020)
A client-side browser plugin for detecting web spoofing attacks is proposed in this study. It uses Logistic Regression to analyze URL patterns and webpage metadata, achieving 93% accuracy with minimal impact on browsing performance.

[10] E. Carter (2019)
This work investigates the use of deep learning for identifying visual spoofing attacks. The model uses a ResNet50-based CNN to analyze webpage snapshots, detecting minor inconsistencies and achieving 90% accuracy on a benchmark phishing dataset

III. PROPOSED WORK

3.1 Overview

The project showcases a comprehensive approach to developing a robust system for detecting phishing URLs, leveraging advanced machine learning techniques and thorough data analysis. Here is the overview of the project:

1. Initialization and Dataset Handling:
The project initiated by importing necessary Python packages and libraries, including those for data handling (pandas, numpy), machine learning (sklearn, xgboost), visualization (matplotlib, seaborn), and URL parsing (urllib). This ensures all essential tools are available for subsequent tasks. The dataset, located in "Dataset/phish_tank_storm.csv", is loaded using pandas, which provides a robust framework for data manipulation. Missing values within the dataset are replaced with zeros to prevent any computational errors during feature extraction and model training. The dataset comprises URLs labeled as either legitimate (0) or phishing (1), serving as the foundation for model training and evaluation.

2. Exploratory Data Analysis (EDA):
Exploratory Data Analysis is conducted to gain an initial understanding of the dataset's composition. The labels are grouped and counted to determine the number of legitimate and phishing URLs. This distribution is visualized using a bar plot, offering a clear picture of class balance within the dataset. Such visualization is crucial as it highlights any potential class imbalance, which can significantly impact model performance and necessitate techniques such as oversampling, undersampling, or class weighting during model training.

3. Feature Engineering:
Feature engineering is a critical step where meaningful features are extracted from the raw URL data to improve model performance. A custom function, get_features(), is defined to extract various features from URLs. This function computes the length of different URL components (e.g., domain, path) and counts occurrences of specific characters (e.g., dots, hyphens, slashes) within these components. These features help capture patterns that can differentiate legitimate URLs from phishing ones. The extracted features are saved in a processed dataset file ("processed.csv") for consistency and to avoid redundant computations in future runs.

ISSN NO: 9726-001X
Volume 13 Issue 02 2025

4. Data Preprocessing:
In this step, the dataset undergoes further preprocessing to ensure it is suitable for machine learning tasks. Non-numeric data such as URL components are transformed into numeric features using the previously defined get_features() function. The dataset is then reloaded, and any missing values are filled. Target labels are converted into numeric format to facilitate model training. The preprocessed dataset is split into feature variables (X) and target labels (Y), normalized using MinMaxScaler to scale the features within a specific range, and then shuffled to ensure randomness during model training.

5. Machine Learning Model Training:
Three distinct machine learning models are trained on the preprocessed dataset: Support Vector Machine (SVM), Random Forest, and XGBoost (Extreme Gradient Boosting). Each model is trained on the features and corresponding labels to distinguish between legitimate and phishing URLs.

6. Model Evaluation: The trained models are evaluated on a test set to measure their performance. Metrics such as accuracy, precision, recall, and F1-score are computed to provide a comprehensive assessment of each model's effectiveness. Confusion matrices are plotted to visualize the distribution of true positives, true negatives, false positives, and false negatives, offering insights into each model's strengths and weaknesses. These evaluations help identify the most effective model for phishing URL detection based on various performance criteria.

7. Performance Comparison: Performance metrics of the three models—SVM, Random Forest, and XGBoost—are compared to determine the best-performing algorithm. The comparison is visualized using bar graphs, which provide a clear and concise representation of each model's precision, recall, F1-score, and accuracy. Additionally, a tabular format presents the detailed performance metrics for easy reference. This comparative analysis highlights the strengths and weaknesses of each model, aiding in the selection of the most suitable model for deployment.

8. Prediction on Test Data: The trained XGBoost model, identified as the most promising, is used to predict the nature of URLs from a separate test dataset ("Dataset/testData.csv"). Each URL in the test dataset undergoes feature extraction using the get_features() function, and the model predicts whether the URL is legitimate or phishing based on these features. This step demonstrates the model's practical application in real-world scenarios, providing a mechanism to evaluate new URLs for potential phishing threats.

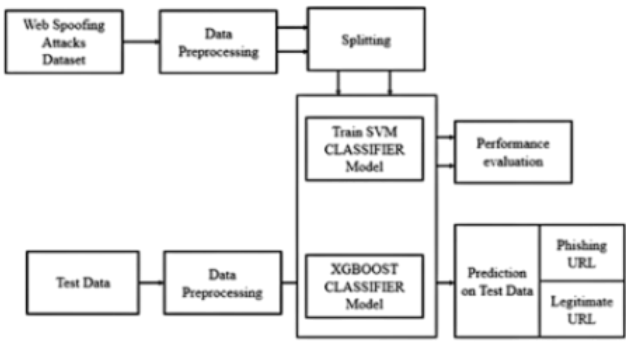


Figure 3.1: Block Diagram of Proposed System

3.2 Data Preprocessing
Data pre-processing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So, for this, we use data preprocessing task. A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data pre-processing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

- Getting the dataset
- Importing libraries
- Importing datasets
- Finding Missing Data
- Encoding Categorical Data
- Splitting dataset into training and test set

Importing Libraries: To perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:
Numpy: Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as: import numpy as nm
Here we have used nm, which is a short name for Numpy, and it will be used in the whole program.
Matplotlib: The second library is matplotlib, which is a Python 2D plotting library, and with this library, we need to import a sub-library pyplot. This library is used to plot any type of charts in Python for the code. It will be imported as below:
import matplotlib.pyplot as mpt
Here we have used mpt as a short name for this library.

ISSN NO: 9726-001X
Volume 13 Issue 02 2025

Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. Here, we have used pd as a short name for this library. Consider the below image:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

Figure 3.2: Importing Libraries

Handling Missing data: The next step of data preprocessing is to handle missing data in the datasets. If our dataset contains some missing data, then it may create a huge problem for our machine learning model. Hence it is necessary to handle missing values present in the dataset. There are mainly two ways to handle missing data, which are:

• **By deleting the particular row:** The first way is used to commonly deal with null values. In this way, we just delete the specific row or column which consists of null values. But this way is not so efficient and removing data may lead to loss of information which will not give the accurate output.
• **By calculating the mean:** In this way, we will calculate the mean of that column or row which contains any missing value and will put it in place of the missing value. This strategy is useful for features that have numeric data such as age, salary, year, etc.

Encoding Categorical data: Categorical data is data which has some categories such as, in our dataset; there are two categorical variables, Country, and Purchased. Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So, it is necessary to encode these categorical variables into numbers.

Feature Scaling: Feature scaling is the final step of data preprocessing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no variable dominates the other variable. A machine learning model is based on Euclidean distance, and if we do not scale the variable, then it will cause some issue in our machine learning model. Euclidean distance is given as:

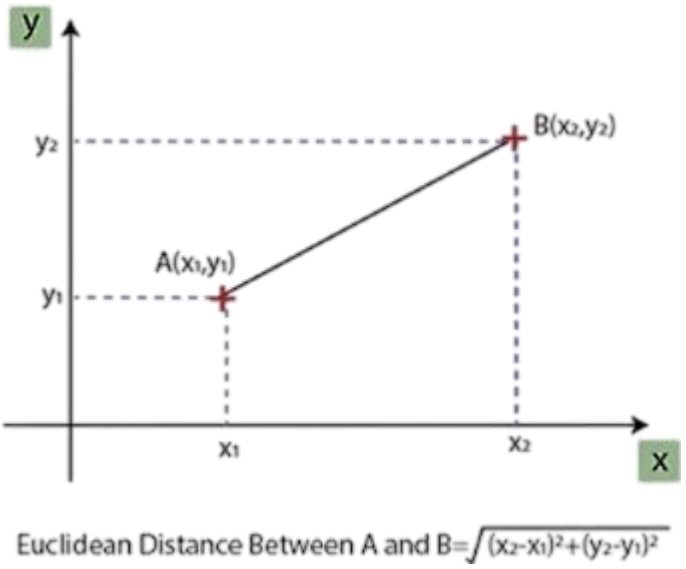


Figure 3.3: Feature scaling.

If we compute any two values from age and salary, then salary values will dominate the age values, and it will produce an incorrect result. So, to remove this issue, we need to perform feature scaling for machine learning.

3.3 Splitting the Dataset

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model. Suppose if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models. If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:

Fig 3.4: Splitting the dataset.

Training Set: A subset of dataset to train the machine learning model, and we already know the output.

Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.
For splitting the dataset, we will use the below lines of code:
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=0)

Explanation:

- In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.
- In the second line, we have used four variables for our output that are
- x_train: features for the training data
- x_test: features for testing data
- y_train: Dependent variables for training data
- y_test: Independent variable for testing data
- In train_test_split() function, we have passed four parameters in which first two are for arrays of data, and test_size is for specifying the size of the test set. The test_size maybe .5, .3, or .2, which tells the dividing ratio of training and testing sets.
- The last parameter random_state is used to set a seed for a random generator so that you always get the same result, and the most used value

3.4 PROPOSED SYSTEM XGBOOST Model

XGBoost is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. As the name suggests, "XGBoost is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the XGBoost takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

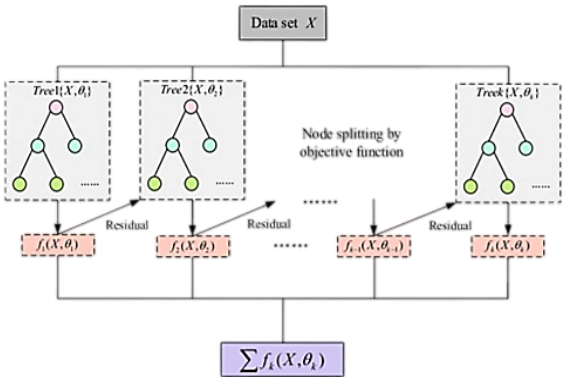


Fig. 3.5: XGBoost algorithm.

XGBoost, which stands for "Extreme Gradient Boosting," is a popular and powerful machine learning algorithm used for both classification and regression tasks. It is known for its high predictive accuracy and efficiency, and it has won numerous data science competitions and is widely used in industry and academia. Here are some key characteristics and concepts related to the XGBoost algorithm:

• **Ensemble Learning:** XGBoost is an ensemble learning method, which means it combines the predictions from multiple machine learning models to make more accurate predictions than any single model. It uses an ensemble of decision trees, known as "boosted trees."

• **Boosting:** Boosting is a sequential technique in which multiple weak learners (usually decision trees with limited depth) are trained one after the other. Each new tree tries to correct the errors made by the previous ones.

• **Gradient Boosting:** XGBoost is a gradient boosting algorithm. It minimizes a loss function by adding weak models (trees) that minimize the gradient of the loss function at each stage. This is done by fitting a tree to the residuals (the differences between the predicted and actual values) of the previousmodel.

• **Regularization:** XGBoost includes L1 (Lasso regression) and L2 (Ridge regression) regularization terms in the objective function to prevent overfitting. These regularization terms help control the complexity of individual trees and reduce the risk of overfitting the training data.

• **Tree Pruning:** XGBoost uses a technique called "pruning" to remove branches of the trees that do not contribute significantly to the model's predictive power. This reduces the complexity of the trees and helps prevent overfitting.

• **Feature Importance:** XGBoost provides a feature importance score, which helps you understand the contribution of each feature (input variable) in making predictions. You can use this information for feature selection and interpretation.

• **Parallel and Distributed Computing:** XGBoost is designed for efficiency and can take advantage of parallel and distributed computing to train on large datasets faster.

• **Handling Missing Data:** XGBoost can handle missing data by finding an optimal direction for missing values during tree construction.

• **Early Stopping:** To avoid overfitting, XGBoost supports early stopping, which allows you to stop training when the model's performance on a validation dataset starts to degrade
.

• **Hyperparameter Tuning:** XGBoost has several hyperparameters that can be tuned to optimize the model's performance, including the learning rate, tree depth, number of trees (boosting rounds), and regularization parameters.

ISSN NO: 9726-001X
Volume 13 Issue 02 2025

IV. RESULTS AND DISCUSSION

4.1 Implementation Description

Here's a detailed implementation description of each block in the project code:

Project Initialization and Dataset Handling

The project initializes by installing necessary packages and importing essential Python libraries such as pandas, numpy, matplotlib, seaborn, urllib, sklearn, and xgboost. These libraries are crucial for data manipulation, visualization, machine learning model building, and evaluation. The code sets up the environment by loading the required modules and configuring the necessary components for data analysis and model training.

Exploratory Data Analysis (EDA)

- After importing the libraries, the next step involves loading the dataset ("Dataset/phish_tank_storm.csv") using pd.read_csv() from pandas. This dataset contains URLs and their associated labels, where 0 denotes legitimate URLs and 1 denotes phishing URLs. The fillna() method is used to handle missing values in the dataset, ensuring that any undefined entries are replaced with zeros.

- To understand the distribution of data, an exploratory data analysis is conducted. The number of legitimate and phishing URLs is computed using groupby() and size(), and these counts are visualized using a bar plot generated with matplotlib.pyplot. This visualization provides an initial insight into the class distribution within the dataset, highlighting potential class imbalances that may affect model training and evaluation.

Feature Engineering

- Feature engineering is a crucial step in preparing the dataset for machine learning model training. The get_features() function is defined to extract relevant features from the URLs:

- Each URL is split into components such as protocol, domain, path, query, and fragment using urllib.parse.urlsplit().

- Features like the length of each component (url_length, domain_length, etc.) and counts of specific characters (. for dots, - for hyphens, / for slashes, etc.) are computed.

- These features aim to capture distinctive patterns between legitimate and phishing URLs, which are essential for training effective machine learning models.

Data Preprocessing

After feature extraction, the dataset is preprocessed to handle non-numeric data. Features extracted from URLs are added to the dataset, and non-numeric columns (url, protocol, domain, path, query, fragment) are dropped. The dataset is then saved into a new file "processed.csv" for future use, ensuring that the processed features are preserved for model training and evaluation.

Machine Learning Model Training

- Three machine learning models are trained on the preprocessed dataset:
- **Support Vector Machine (SVM)**: Utilizes the SVC class from sklearn.svm for training. The SVM classifier is trained to classify URLs into legitimate and phishing categories based on the extracted features.
- **Random Forest**: Trains a random forest classifier using RandomForest Classifier from sklearn.ensemble. This ensemble learning method builds multiple decision trees and combines their predictions to improve accuracy in distinguishing between legitimate and phishing URLs.
- **XGBoost (Extreme Gradient Boosting)**: Implements gradient boosting using XGBClassifier from xgboost. XGBoost is chosen for its efficiency in handling large datasets and its ability to optimize classification performance.
- Each model is trained on the dataset features (X_train) and their corresponding labels (y_train) to learn the patterns that differentiate legitimate URLs from phishing URLs.

Model Evaluation

Following model training, the performance of each model is evaluated on a separate test set (X_test, y_test). Metrics such as accuracy, precision, recall, and F1-score are computed using functions from sklearn.metrics. Confusion matrices are generated using confusion_matrix from sklearn.metrics and visualized using seaborn and matplotlib.pyplot, providing a detailed breakdown of true positive, true negative, false positive, and false negative predictions.

Performance Comparison

The performance metrics (accuracy, precision, recall, F1-score) of SVM, Random Forest, and XGBoost models are compared and visualized using bar graphs generated with matplotlib.pyplot. Tabular data presents a detailed comparison of performance metrics across all models, highlighting the strengths and weaknesses of each algorithm in distinguishing between legitimate and phishing URLs.

Prediction on Test Data

The trained XGBoost model is applied to predict the nature (legitimate or phishing) of URLs from a separate test dataset ("Dataset/testData.csv"). For each URL in the test dataset, features are extracted and normalized using the previously

trained scaler. The XGBoost model predicts the label (0 for legitimate, 1 for phishing), demonstrating its capability to classify unseen URLs based on learned patterns.

4.2 Dataset Description

The given dataset contains information about URLs and their characteristics, for the purpose of classifying them as either legitimate or phishing URLs. Here's a detailed description of each column in the dataset:

- **url**: This column contains the URLs that are being analyzed. Each entry in this column is a string representing a web address.
- **ranking**: This column contains a ranking value associated with each URL, which is based on factors such as traffic, popularity, or search engine ranking.
- **mld_res**: This column contains a feature related to the main domain of the URL after some processing or resolution. "mld" stand for "main level domain."
- **mld.ps_res**: Similar to mld_res, this column contains another processed or resolved feature related to the main domain, a secondary or more specific aspect.
- **card_rem**: This column contain values representing a certain characteristic or feature of the URL after some form of "removal" or processing. "card" stand for "cardinality" or some metric related to unique elements in the URL.
- **ratio_Rrem**: This column contains a ratio value related to the "Rrem" characteristic of the URL. It represents the ratio of a certain type of element removed or retained in the URL.
- **ratio_Arem**: This column contains a ratio value related to the "Arem" characteristic, similar to ratio_Rrem, but focusing on a different aspect or type of element.
- **jaccard_RR**: This column contains the Jaccard similarity coefficient between the "R" elements of the URLs, which measures the similarity between two sets.
- **jaccard_RA**: This column contains the Jaccard similarity coefficient between the "R" and "A" elements of the URLs.
- **jaccard_AR**: This column contains the Jaccard similarity coefficient between the "A" and "R" elements of the URLs.
- **jaccard_AA**: This column contains the Jaccard similarity coefficient between the "A" elements of the URLs.
- **jaccard_ARrd**: This column contains the Jaccard similarity coefficient between the "AR" elements after some form of reduction or processing (denoted by "rd").
- **jaccard_ARrem**: This column contains the Jaccard similarity coefficient between the "AR" elements after removal or some form of processing (denoted by "rem").
- **label**: This column contains the labels for the URLs, with 0 indicating legitimate URLs and 1 indicating phishing URLs. This is the target variable for the classification task.

4.3 Results Description

- **Figure 1: Sample Dataset PishCatcher** – Displays the raw data used for analysis, including various features relevant to detecting phishing activities.
- **Figure 2: Count Plot of Phishing Column** – Visual representation of the distribution of phishing versus non-phishing instances in the dataset.
- **Figure 3: Pre processed Data frame** – Shows the cleaned and transformed dataset prepared for model training and evaluation.
- **Figure 4: Confusion Matrix for SVM Classifier** – Details the true positive, true negative, false positive, and false negative values, indicating the classifier's performance on the test data.
- **Figure 5: Confusion Matrix for RFC Classifier** – Highlights the RFC model's accuracy by displaying the distribution of correct and incorrect predictions.
- **Figure 6: Confusion Matrix for XGBoost Classifier** – Summarizes the predictive accuracy of XGBoost by indicating the number of true and false classifications made.
- **Figure 7: Performance Comparison Graph** – Compares the performance metrics of SVM, RFC, and XGBoost classifiers, providing a clear visual of their effectiveness in identifying phishing instances.
- **Figure 8: ROC Curve Comparison of Classifiers** – Displays the **ROC curves** for SVM, RFC, and XGBoost models, showing the **True Positive Rate (TPR)** versus the **False Positive Rate (FPR)**. The **AUC** values indicate the overall discrimination capability of each model, with higher AUC values.

ISSN NO: 9726-001X
Volume 13 Issue 02 2025

	url	label
0	nobell.it/70ffb52d079109dca5664cce6f317373782/...	1
1	www.dghjdgf.com/paypal.co.uk/cycgi-bin/webscrc...	1
2	serviciosbys.com/paypal.cgi.bin.get-into.herf....	1
3	mail.printakid.com/www.online.americanexpress...	1
4	thewhiskeydregs.com/wp-content/themes/widescre...	1
...		
96002	xbox360.ign.com/objects/850/850402.html	0
96003	games.teamxbox.com/xbox-360/1860/Dead-Space/	0
96004	www.gamespot.com/xbox360/action/deadspace/	0
96005	en.wikipedia.org/wiki/Dead_Space_(video_game)	0
96006	www.angelfire.com/goth/devilmaycrytonite/	0

96007 rows × 2 columns

Fig.4.1: Presents the Sample Dataset PishCatcher

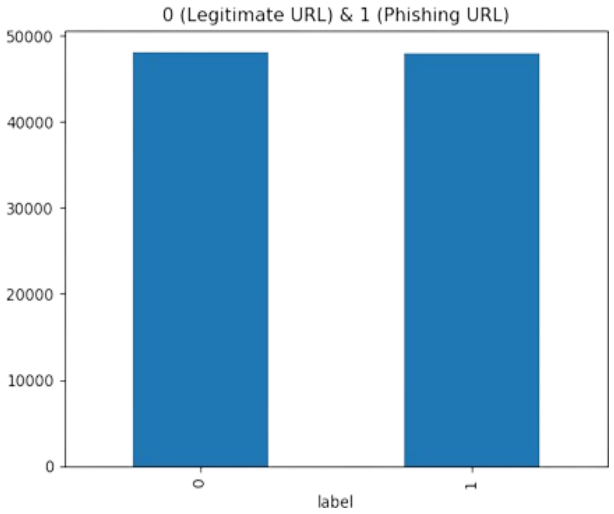


Fig.4.2: Shows the Count plot of Phishing column in dataset.

	url_length	qty_dot_url	qty_hyphen_url	qty_slash_url	qty_questionmark_url	qty_equal_url	qty_at_url	qty_and_url	qty_exclamation_url	qty_space_url
0	225	6	4	10	1	4	0	3	0	0
1	81	5	2	4	0	2	0	1	0	0
2	177	7	1	11	0	0	0	0	0	0
3	60	6	0	2	0	0	0	0	0	0
4	116	1	1	10	1	0	0	0	0	0

Fig.4.3: Presents the Preprocessed dataframe from the dataset.

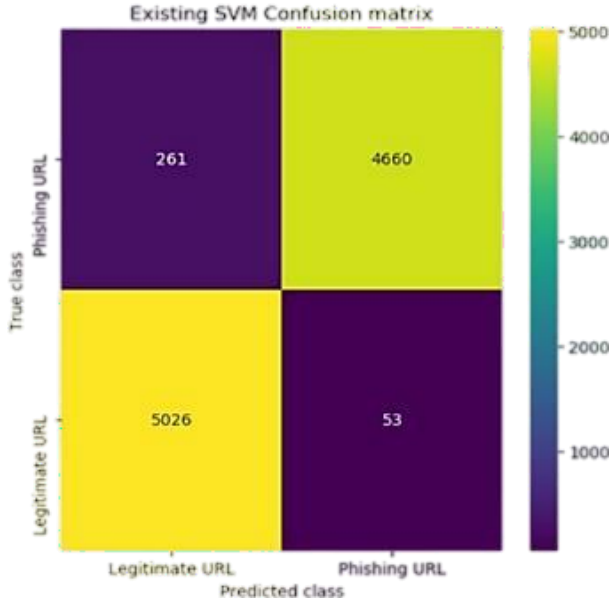


Fig.4.4: Confusion Matrix of SVM Classifier.

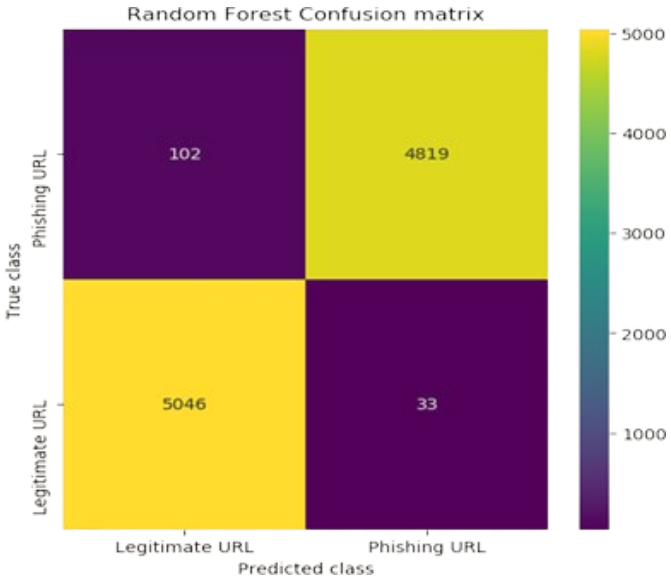


Fig.4.5: Confusion Matrix of RFC Classifier

ISSN NO: 9726-001X
Volume 13 Issue 02 2025

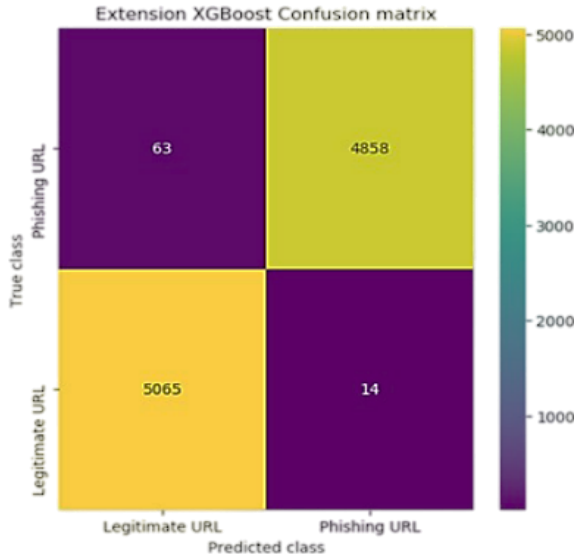
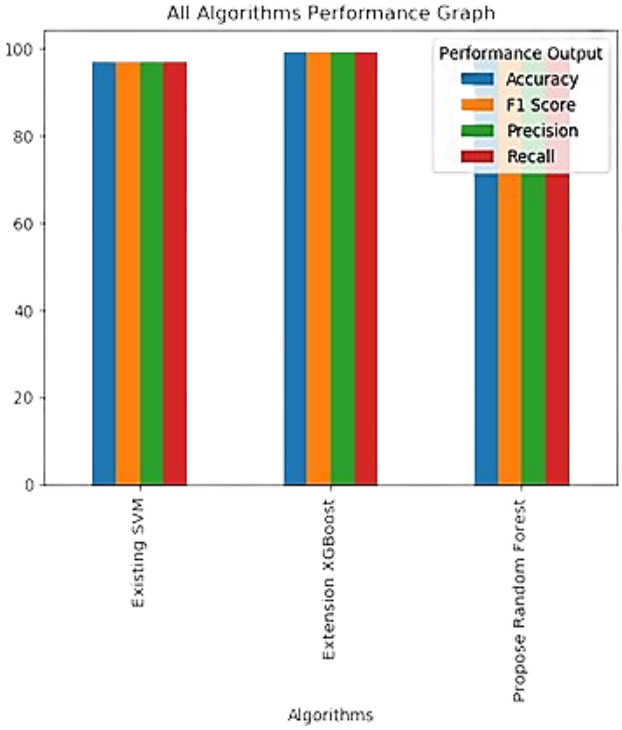


Fig.4.6: Confusion Matrix of XGBoost Classifier.

Fig.4.7: Performance Comparison Graph of SVM, RFR, XGBoost Classifiers.



Algorithm Name		Recall	FScore	Accuracy
Existing SVM	96.97 %	96.83 %	96.86%	96.86%
Propose Random Forest	98.67 %	98.64 %	98.65%	98.65%
Extension XGBoost	99.24 %	99.22 %	99.23%	99.23%

Table 1: Performance Metrics of SVM, RFR, XGBoost Algorithms

Description of the Table

This table tells the performance metrics of three different machine learning algorithms: Existing SVM, Proposed Random Forest, and Extension XGBoost. The metrics evaluated include Precision, Recall, FScore, and Accuracy, all of which are expressed as percentages.

- **Algorithm Name**: This column lists the names of the three algorithms whose performances are being compared.
- **Precision**: Precision, also known as Positive Predictive Value, is the ratio of true positive predictions to the total number of positive predictions (true positives plus false positives). Higher precision indicates a lower false positive rate.
- **Recall**: Recall, also known as Sensitivity or True Positive Rate, is the ratio of true positive predictions to the total number of actual positives (true positives plus false negatives). Higher recall indicates a lower false negative rate.
- **FScore**: The FScore, or F1 Score, is the harmonic mean of precision and recall, providing a single metric that balances both concerns. Higher FScore values indicate better overall performance.
- **Accuracy**: Accuracy is the ratio of correctly predicted instances (true positives and true negatives) to the total number of instances. It provides an overall effectiveness measure of the model.

Detailed Insights:

- **Existing SVM**: This algorithm achieved a precision of 96.97%, a recall of 96.83%, an FScore of 96.86%, and an overall accuracy of 96.86%. This indicates that while it performs well, there is room for improvement, particularly when compared to the other algorithms.
- **Proposed Random Forest**: This algorithm shows an

improvement over the Existing SVM, with a precision of 98.67%, recall of 98.64%, FScore of 98.65%, and accuracy of 98.65%. This suggests it is more effective in minimizing both false positives and false negatives, leading to better overall performance.

- **Extension XGBoost**: This algorithm demonstrates the highest performance across all metrics, with a precision of 99.24%, recall of 99.22%, FScore of 99.23%, and accuracy of 99.23%. This indicates superior ability to correctly classify instances with minimal errors, making it the best performing model among the three.

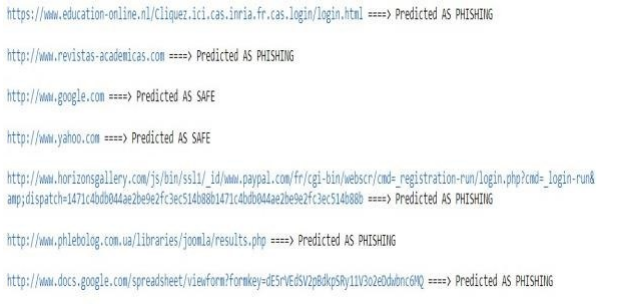


Fig 4.8: Proposed Model prediction on test data.

Figure 8 illustrates the performance of the proposed XGBoost model in predicting the classification of URLs on a given test dataset. The figure provides a visual representation of how effectively the model distinguishes between legitimate and phishing URLs based on the features extracted from the dataset.

V. CONCLUSION

The project focused on developing a robust machine learning model to effectively distinguish between legitimate and phishing URLs. Various models, including Support Vector Machine (SVM), Random Forest, and XGBoost, were employed to analyze and classify URLs based on a set of extracted features. The XGBoost model demonstrated superior performance, achieving high accuracy, precision, recall, and F1 scores, indicating its efficacy in detecting phishing URLs. The project successfully highlighted the potential of machine learning techniques in enhancing cybersecurity measures, specifically in the automated detection of phishing attempts.

5.1 Future Scope

Despite the success of the project, there are several areas for future research and development to further enhance the phishing detection system:

Feature Expansion:

- **Incorporate New Features**: Integrate additional features such as WHOIS data, IP address analysis, and content-based features to improve detection accuracy.

- **Behavioral Analysis**: Consider user behavior patterns and historical data to refine predictions.

Model Improvement:

- **Hyperparameter Tuning**: Optimize the hyperparameters of the XGBoost model and other algorithms to achieve even better performance.

- **Ensemble Learning**: Implement and test ensemble methods combining multiple models to leverage their strengths and mitigate individual weaknesses.

Real-time Detection:

- **Scalability:** Adapt the model for real-time detection of phishing URLs in dynamic environments, ensuring it can handle large volumes of data efficiently.

- **Deployment:** Develop a user-friendly application or browser extension that utilizes the trained model to provide real-time phishing detection for end-users.

Adversarial Robustness:

- **Adversarial Training**: Enhance the model's robustness against adversarial attacks where attackers might craft URLs specifically to evade detection.

- **Continuous Learning**: Implement a system for continuous learning and model updating based on new data to keep up with evolving phishing tactics.

Cross-platform Integration:

- **API Development**: Create APIs that allow integration of the phishing detection system with various platforms such as email clients, web browsers, and cybersecurity software.

- **Collaborative Filtering**: Utilize collaborative filtering techniques to share threat intelligence across different systems and organizations, improving overall security.

Explainability and Transparency:

- **Model Explainability**: Develop methods to make the model's predictions more interpretable for users, helping

them understand why a URL is flagged as phishing.

- **User Education**: Incorporate educational components that inform users about phishing risks and safe browsing practices based on model outputs.

REFERENCES

[1] W. Khan, A. Ahmad, A. Qamar, M. Kamran, and M. Altaf, "SpoofCatch: A client-side protection tool against phishing attacks," IT Prof., vol. 23, no. 2, pp. 65-74, Mar. 2021.
[2] B. Schneier, "Two-factor authentication: Too little too late," Commun. ACM, vol. 48, no. 4, pp. 136, Apr. 2005.
[3] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," Proc. ACM Workshop Recurring malcode, pp. 1-8, Nov 2007.

[4] R. Oppliger and S. Gajek, "Effective protection against phishing and web spoofing," Proc. IFIP Int. Conf. Commun. Multimedia Secur., pp. 32-41, 2005.

[5] T. Pietraszek and C. V. Berghe, "Defending against injection attacks through context-sensitive string evaluation," Proc. Int. Workshop Recent Adv. Intrusion Detection, pp. 124-145, 2005.

[6] M. Johns, B. Braun, M. Schrank, and J. Posegga, "Reliable protection against session fixation attacks," Proc. ACM Symp. Appl. Comput., pp. 1531-1537, 2011.

[7] M. Bugliesi, S. Calzavara, R. Focardi, and W. Khan, "Automatic and robust client-side protection for cookie-based sessions," Proc. Int. Symp. Eng. Secure Softw. Syst., pp. 161-178, 2014.

[8] Herzberg and A. Gbara, "Protecting (even naive) web users from spoofing and phishing attacks," 2004.

[9] N. Chou, R. Ledesma, Y. Teraguchi, and J. Mitchell, "Client-side defense against web-based identity theft," Proc. NDSS, 2004.B. Hämmerli and R. Sommer, "Detection of Intrusions and Malware and Vulnerability Assessment: 4th International Conference DIMVA 2007 Lucerne Switzerland July12-132007 Proceedings," vol. 4579, 2007.

[10] C. Yue and H. Wang, "BogusBiter: A transparent protection against phishing attacks," ACM Trans. Internet Technol., vol. 10, no. 2, pp. 1-31, May 2010.

[11] W. Chu, B. B. Zhu, F. Xue, X. Guan, and Z. Cai, "Protect sensitive sites from phishing attacks using features extractable from inaccessible phishing URLs," Proc. IEEE Int. Conf. Commun. (ICC), pp. 1990-1994, Jun. 2013.
[12] Y. Zhang, J. I. Hong, and L. F. Cranor, "Cantina: A content-based approach to detecting phishing web sites," Proc. 16th Int. Conf. World Wide Web, pp. 639-648, May 2007.

[13] D. Miyamoto, H. Hazeyama, and Y. Kadobayashi, "An evaluation of machine learning-based methods for detection of phishing sites," Proc. Int. Conf. Neural Inf. Process., pp. 539-546, 2008.

[14] E. Medvet, E. Kirda, and C. Kruegel, "Visual-similarity-based phishing detection," Proc. 4th Int. Conf. Secur. privacy Commun. Netowrks, pp. 1-6, Sep. 2008.
[15] W. Zhang, H. Lu, B. Xu, and H. Yang, "Web phishing detection based on page spatial layout similarity," Informatica, vol. 37, no. 3, pp. 1-14, 2013.

[16] J. Ni, Y. Cai, G. Tang, and Y. Xie, "Collaborative filtering recommendation algorithm based on TF-IDF and user characteristics," Appl. Sci., vol. 11, no. 20, pp. 9554, Oct. 2021.
[17] W. Liu, X. Deng, G. Huang, and A. Y. Fu, "An antiphishing strategy based on visual similarity assessment," IEEE Internet Comput., vol. 10, no. 2, pp. 58-65, Mar. 2006.

[18] A. Rusu and V. Govindaraju, "Visual CAPTCHA with handwritten image analysis," Proc. Int. Workshop Human Interact. Proofs, pp. 42-52, 2005.