MALICIOUS URL DETECTION USING MACHINE LEARNING

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

A Vamsi Kumar – AP21110010135

G.Manoj Kumar – AP21110010147

Subodh Amru Kiliveti – AP21110010141

K Sravani – AP21110010160

Under the Guidance of

Dr. Elakkiya E

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

Nov, 2023

Certificate

Date: 27-Nov-23

This is to certify that the work present in this Project entitled "**Malicious Link Detection**" has been carried out by **A Vamsi Kumar, Manoj Kumar , Subodh Amru K , K Sravani ,** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

Supervisor

(Signature)

Dr. Elakkiya E

Assistant Professor,

SRM University - Andhra Pradesh

Acknowledgements

I am grateful to my family and friends for their everlasting support and encouragement. Their faith in me has been a motivating force in completing this endeavour. I thank God for the blessings and direction that have brightened my path during this trip.

In addition, I'd want to express my gratitude to my teacher, Elakkiya E. I am grateful for the chance she gave us to work on this project, as well as her ongoing support and direction. Her knowledge and guidance have been invaluable in developing my grasp of several machine learning modules. I consider myself fortunate to have had such a dedicated and inspirational teacher.

Finally, I'd like to thank all of my coworkers for their hard work. Working as a team has been a pleasant experience, and I appreciate each member's contribution to the successful completion of this project.

Table of Contents

Abstract

The goal of this project is to create a robust malicious link detection system using ensemble learning techniques. The system improves accuracy and generalization by utilizing a diverse set of classifiers such as Random Forest, XGBoost, Gradient Boosting, and AdaBoost.The models are evaluated through cross-validation and ranked based on their performance. A weighted ensemble is then constructed using the top-performing classifiers, considering their ranks and normalized weights. The resulting ensemble model demonstrates improved predictive capabilities, offering an effective solution for detecting malicious links. The voting classifier ensures a balanced combination of individual models, contributing to a more resilient and accurate malicious link detection system.

Statement of Contributions

Responsible for

Analysis and Coding : Subodh Amru K

Report writing : Subodh Amru K, Ruchitha Jannu, Sravani K, Vamsi A

Abbreviations

1. URL: Uniform Resource Locator
2. HTTPS: Hypertext Transfer Protocol Secure
3. IP: Internet Protocol
4. TLD: Top-Level Domain
5. HTTP: Hypertext Transfer Protocol
6. XGBoost: Extreme Gradient Boosting
7. KNN: K-Nearest Neighbors
8. AdaBoost: Adaptive Boosting
9. FP: False Positives
10. TP: True Positives
11. TN: True Negatives
12. FN: False Negatives

List of Tables

List of Figures

List of Equations

1. Introduction

The increasing frequency of cyber risks poses a significant danger in an era dominated by online activities such as social networking, e-commerce, banking, and business. The emergence of online crimes, particularly through the manipulation of Universal Resource Locators (URLs), has heightened the need for strong internet security measures. Clicking on malicious URLs may damage systems and expose sensitive personal information, highlighting the vital need for cyber defenses. The phrase "malicious" refers to a wide range of attacks, including but not limited to benign, spam, malware, and phishing.

Blacklisting services have evolved as critical tools within the internet security industry in response to the increasing cyber threat scenario. These services maintain databases known as Blacklists, which contain URLs that have been identified as dangerous. However, the efficiency of URL blacklisting is challenged since attackers can leverage system weaknesses by modifying URL components to avoid detection. This vulnerability is caused by factors like obsolete entries, errors, or a lack of timely assessments, which allows a large number of harmful websites to avoid blacklisting.

The use of machine learning (ML) approaches for malicious URL identification has gained significance in order to improve the efficiency of online security measures. The automated model update capabilities and identification of freshly created URLs demonstrate ML's versatility in dealing with the changing nature of cyber threats. Deep learning models have recently been studied, with approaches used to automatically recognise and extract information from new URLs. URL length, letter count, digit count, special characters count, secure HTTP, existence of an IP address, root domain, anomalous URL, and top-level domain type are extracted properties that ML systems use to classify URLs as benign or dangerous.

Among the strong models used for this purpose are Random Forest, Logistic Regression, XGBoost, SVM with a Voting ensemble, and Decision Tree Classifier. This report offers an in-depth examination of the processing framework designed for Malicious URL Detection Using Machine Learning, shedding light on the importance of each model and the extracted features in fortifying cyber defenses against the multifaceted threat landscape of malicious URLs.

1.1 Parts of the Link

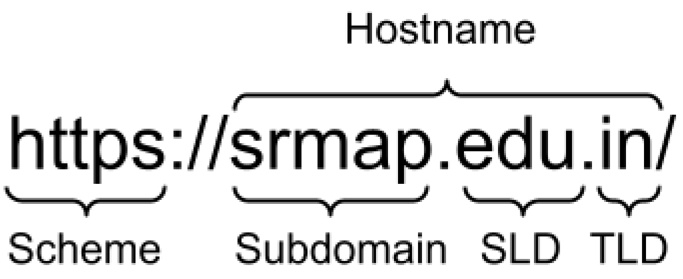

Figure 1: Example of a URL : Uniform Resource Locator

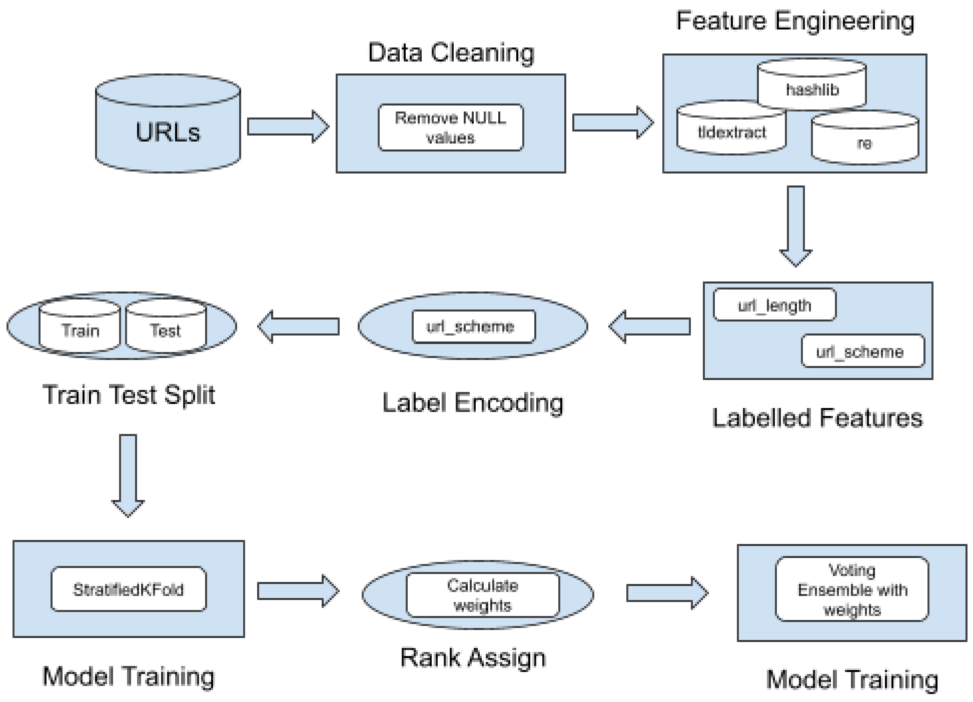

Figure 2: Processing framework for Malicious URL Detection using Machine Learning

2.Methodology

The methodology outlines the step-by-step approach taken in the code for malicious URL detection. It involves data collection, feature engineering, machine learning model development, and evaluation.

2.1. Data Collection

- Collection of a dataset containing URLs labeled with their respective types, such as 'benign,' 'defacement,' 'phishing,' and 'malware.'

2.2. Feature Engineering

- Extraction of meaningful features from the URLs to facilitate machine learning model training.
- Features include:
- URL Scheme
- URL Length
- URL Host Length
- URL Host is IP
- URL Path Length
- Is Encoded
- URL Entropy
- Number of Subdirectories
- Presence of Keywords
- Use of Non-Standard Ports
- Number of Periods
- Use of Special Characters
- Number of Parameters
- Number of Fragments
- Number of Digits
- Length of Top-Level Domain (TLD)
- Use of Subdomains

- Use of Hyphens
- Presence of Redirection
- Use of Non-Latin Characters
- Length of Path Tokens
- Presence of Common Words in Subdomains

2.3. Label Encoding

- Conversion of categorical labels in the "type" column to numeric values

  ('benign': 0, 'defacement,' 'phishing,' 'malware': 1).
- Drop the original "type" column.

2.4. Data Splitting

- Using the 'train_test_split' function, divide the dataset into training and testing sets.
- Setting aside 60% of the data for training ('X_train', 'y_train') and 40% for testing ('X_test', 'y_test').

2.5. Machine Learning Model Training

- Utilizing different machine learning algorithms for training:
- Random Forest
- XGBoost
- AdaBoost
- K-Nearest Neighbors (KNN)

2.6. Cross-Validation

- Implementing Stratified K-Fold cross-validation to evaluate model performance robustly.

2.7. Weight Assignment Using Rank Sum

- Assigning weights to models based on their ranks obtained from cross-validation scores.

- Normalize the weights to create a weighted ensemble.

2.8. Voting Classifier

- Training a Voting Classifier using a combination of Random Forest, XGBoost, AdaBoost, and KNN with assigned weights.

3. PROPOSED WORK

The implemented code focuses on the development of a comprehensive system for malicious link detection, leveraging a diverse set of features extracted from URLs. The proposed work encompasses several key phases, beginning with robust data preprocessing to ensure dataset integrity and readiness for model training. Duplicate entries are removed, and categorical labels are encoded, setting the foundation for subsequent analyses.

Feature engineering plays a crucial role in enhancing the model's understanding of URL characteristics. To capture various aspects of URL structures, a set of 28 distinct features is engineered, ranging from URL length to the presence of specific keywords and the use of non-Latin characters. These features provide a more nuanced representation of URLs, allowing the model to detect patterns that indicate malicious intent.

The machine learning pipeline involves the evaluation of various classifiers, including Random Forest, XGBoost, AdaBoost, and K-Nearest Neighbors (KNN), through stratified k-fold cross-validation. Model ranking and weight assignment based on the Rank Sum method are employed to assign significance to each model's contribution. The culmination of these efforts leads to the creation of an ensemble model, utilizing a weighted voting approach with normalized weights derived from the model rankings.

The proposed work concludes with the ensemble model being trained and evaluated on the test dataset, with its performance measured using various metrics and a detailed classification report. This comprehensive approach seeks to improve the accuracy and robustness of malicious link detection, thereby advancing cybersecurity measures.

4. Features Used

Feature Engineering Code

4.1 Features

Feature 1: URL Length

- The "url_length" feature measures the length of a URL in characters. On average, phishers tend to use longer URLs (74 characters is the observed average) to conceal the actual website name.

Feature 2: URL Scheme

- The "url_scheme" feature analyzes the protocol component (HTTP, HTTPS) of URLs. Understanding the URL scheme aids in categorizing and identifying potentially fraudulent or dangerous web addresses.

Feature 3: URL Path Length

- "url_path_length" represents the number of characters or segments in a URL's path. It provides insights into the complexity and depth of a website or web application's directory structure.

Feature 4: URL Host Length

- "url_host_length" computes the number of characters in a URL's hostname or domain name. It measures the length of time a domain name appears in text form within a web address.

Feature 5: URL Host is IP

- The "url_host_is_ip" feature determines whether an IP address, instead of a domain name, is present in the host portion of a URL. This is valuable for recognizing instances where an IP address is used instead of a traditional domain name.

Feature 6: Number of Digits

- "num_digits" counts the total number of numeric characters (0–9) in a URL, covering the scheme, host, path, query parameters, and any fragments.

Feature 7: Number of Parameters

- "num_parameters" indicates the total number of unique variables or data points in the query string part of a URL. Parameters are typically specified after the "?" character and separated by "&" in a URL.

Feature 8: Number of Fragments

- "num_fragments" represents the number of unique IP fragments connected to a network packet. It aids in assessing the integrity of network traffic and identifying potential fragmentation attacks or data transfer issues.

Feature 9: Is Encoded

- The "is_encoded" feature signifies whether the characters in a URL have been encoded using percent-encoding or URL encoding.

Feature 10: URL Entropy

- "url_entropy" measures the unpredictability or randomness in the characters that make up a URL. It quantifies how complex or disorganized the URL string is.

Feature 11: Has Port (with improved port check)
- "has_nonstandard_port" checks if the URL includes a non-standard port, different from the standard HTTP (80) or HTTPS (443) ports.

Feature 12: Number of Subdirectories
- "num_subdirectories" counts the number of directories or folder levels present in the path portion of the URL.

Feature 13: Number of Periods
- "num_periods" counts the "." (dot) characters in the hostname or domain name part of a URL.

Feature 14: Has Keyword 'client'
- "has_client_keyword" flags URLs containing the keyword 'client.' In many legitimate URLs, the term "client" is commonly used.

Feature 15: Has Keyword 'admin'
- "has_admin_keyword" identifies URLs containing the keyword 'admin.' The term "admin" often denotes privileged or administrative access.

Feature 16: Has Keyword 'server'
- "has_server_keyword" signals the presence of the keyword 'server.' The term "server" in a URL context could be relevant for monitoring server-related activities.

Feature 17: Has Keyword 'login'
- "has_login_keyword" checks for the presence of the keyword 'login' in URLs. The term "login" can be crucial for security monitoring, indicating actions related to authentication.

Feature 18: Has Port (with improved port check)

- An additional check for non-standardports has been included in the

"has_nonstandard_port" feature.

Feature 19: Use of Special Characters
- "has_special_characters" assesses whether the URL contains any special characters, considering characters from the `string.punctuation` module.

Feature 20: Length of Top-Level Domain (TLD)
-"tld_length" computes the length of a URL's Top-Level Domain (TLD).

Feature 21: Use of Subdomains
- "num_subdomains" counts the number of subdomains in a URL using the `tldextract` library.

Feature 22: Length of Top-Level Domain (TLD)
-"tld_length" computes the length of the Top-Level Domain (TLD) in a URL.

Feature 23: Use of Hyphens
- "has_hyphens" checks if the URL contains hyphens.

Feature 24: Use of Redirection
- "has_redirection" checks if the URL contains the substring 'http', indicating the use of HTTP.

Feature 25: Use of Non-Latin Characters
- "has_non_latin_characters" evaluates whether the URL includes non-Latin characters.

Feature 26: Length of Path Tokens
- "path_token_lengths" calculates the number of tokens (segments) in the path portion of the URL.

Feature 27: Has Common Words in Subdomains
- "has_common_subdomain_keywords" checks if common words ('www,' 'mail,' 'blog,' 'forum,' 'shop') are present in the subdomain.

These features collectively provide a comprehensive set of characteristics for each URL, enhancing the potential for the model to capture patterns indicative of phishing or malicious behavior.

4.2. Train-Test Split

Reference code

4.2.1. Input and Output Variables

- X (Input Features): The data frame `df` is used to create a new dataframe `X` by excluding two columns - 'url_type' (output variable) and 'url' (non-predictive feature).
- y (Output Labels): The 'url_type' column from the original dataframe `df` is assigned to the variable `y`. This column represents the target variable, indicating whether a URL is benign (0) or malicious (1).

4.2.2. Train - Test Split

- The sklearn module's 'train_test_split' function is used to divide the dataset into training and testing sets.
- The test size split is 0.4, which implies that 60% of the data was used for training and 40% for testing.
- A random seed was set to 42 which ensures reproducibility. The same seed produces the same split every time, allowing for consistent and comparable model evaluations.

4.2.3. Resulting Sets

The training set is used to train machine learning models, whereas the testing set is used to evaluate the models' performance. This split is essential for assessing the models' generalization capabilities and ensuring that they can effectively classify URLs, including those not encountered during training.

5. Models Used for Training

Reference Code

5.1 Random Forest

Random Forest is a robust ensemble machine learning technique which is used to classify URLs, especially malicious URLs.During training, this method generates multiple decision trees, with each tree considering a subset of URL-derived features. Random Forest mitigates the risk of overfitting by aggregating predictions from these trees, resulting in a reliable classification for a given URL. In the cybersecurity domain, this ensemble approach excels at handling diverse URL features, effectively recognizing patterns, and distinguishing between benign and malicious URLs.

5.2 XGBoost

Extreme Gradient Boosting, or XGBoost, is a powerful machine learning algorithm known for its accuracy and efficiency in regression and classification tasks. XGBoost, which is widely used in detecting malicious URLs, employs ensemble learning, generating a series of decision trees to correct errors from previous ones. This algorithm handles a wide range of URL features efficiently, combining weak learners into a robust model that excels at recognizing complex patterns in URL data. XGBoost is a popular choice for accurately identifying both benign and malicious URLs due to its optimized implementation and high performance.

5.3 ADA Boosting

AdaBoost, or Adaptive Boosting, is a technique for improving the performance of weak learners in order to create a strong classifier. AdaBoost assigns weights to misclassified samples in the context of malicious URL detection, adjusting these weights iteratively to focus on difficult cases. AdaBoost improves overall accuracy by adapting its approach based on the difficulty of classification.Because of its ability to handle various data types and refine predictions through successive iterations, this technique is useful in cybersecurity.In the context of malicious URL detection, AdaBoost assigns weights to misclassified samples, adjusting these weights iteratively to focus on challenging cases. By adapting its approach based on the difficulty of classification, AdaBoost achieves an overall improvement in accuracy. This technique is valuable in cybersecurity for its ability to handle various data types and refine predictions through successive iterations.

5.4 K-Nearest Neighbors (KNN)

K-Nearest Neighbors, a machine learning classification algorithm. KNN, along with Random Forest, XGBoost, and AdaBoost, is included in the provided code for ensemble evaluation. KNN classifies a data point in the feature space based on the class labels of its k-nearest neighbors. The parameter 'n_neighbors', which represents the number of neighbors, can be adjusted for optimal performance. KNN contributes to the collective decision-making process in the context of ensemble evaluation, improving the model's overall accuracy.

5.5 Voting Classifier

A Voting Classifier is a form of ensemble learning approach that combines numerous models to create a single prediction. This method entails training independent base models and combining their predictions via a voting mechanism. A Voting Classifier seeks to improve overall performance by utilizing various algorithms. The Voting Classifier in the provided code combines Random Forest, XGBoost, AdaBoost, and KNN to create a more robust and accurate model for detecting malicious URLs.

Two primary categories of voting classifiers exist:

5.5.1. Hard Voting

In a hard voting, each base model in the ensemble predicts the class label, and the class with the most votes becomes the final prediction.

5.5.2. Soft Voting Classifier

When soft voting, the probability scores that each classifier predicts for each class are taken into account. It considers the average probability scores for each class across all classifiers and selects the class with the highest average probability as the final prediction instead of the class with the most votes.

5.6.Voting Classifier Benefits

Enhanced Performance
Voting classifiers frequently perform better than single models, particularly when the base models have a variety of advantages and disadvantages. Individual models may miss subtleties and patterns that the ensemble can pick up on.

Robustness
By lessening the effect of noise in individual models, ensembling helps reduce the risk of overfitting. In cases where different models may perform well on different portions of the dataset, it offers a more stable and trustworthy prediction.

Flexibility
A range of base models, including various machine learning algorithms, can be used to build voting classifiers. The ensemble's adaptability to different data types and problem domains is made possible by this flexibility.

Increased Accuracy
By combining predictions from several models, it can effectively leverage the advantages of various algorithms or models, which frequently results in improved predictive performance when compared to individual classifiers.

Decreased Overfitting
When employing different algorithms or models with different biases and error patterns, combining multiple models can help reduce overfitting. The final classifier is made more robust and broadly applicable thanks to this diversity.

Enhanced Stability
The Voting Classifier's ensemble design usually results in increased stability and a decreased sensitivity to noise or outliers in the dataset, which produces predictions that are more trustworthy.

Adaptability
It permits adaptability in the combination of different classifier types, allowing for the addition of alternative feature representations or learning strategies, which enhances performance.

6. EXPERIMENTAL ANALYSIS

6.1 Flow of Analysis

6.1.1. Individual Model Evaluation

Stratified K-Fold cross-validation assesses the individual performance of Random Forest, XGBoost, AdaBoost, and KNN models.

6.1.2. Ranking Models

Models are ranked based on their mean accuracy scores.

6.1.3. Weight Assignment

Models are assigned weights using the rank sum technique, reflecting their performance.

6.1.4. Voting Ensemble

A Voting Classifier is constructed, incorporating models with assigned weights. This ensemble approach leverages the strengths of individual models to enhance overall accuracy.

6.1.5. Performance Metrics

The ensemble's accuracy, precision, recall, and F1 score are computed, illustrating the efficacy of integrating models in the malicious link detection job.

By systematically extracting features and encoding categorical data, the analysis ensures that the dataset is well-prepared for machine learning tasks. The chosen features cover a broad range of characteristics, including URL structure, content, and common patterns associated with malicious URLs. This comprehensive dataset will contribute to the training and evaluation of machine learning models, ultimately enhancing the accuracy and effectiveness of the malicious link detection system.

6.2 Code Analysis

6.2.1. Stratified K-Fold Cross-Validation

At this stage, we used the Stratified K-Fold is a cross-validation technique which is used to assess the performance of individual machine learning models.The results show each model's mean accuracy over multiple folds (three in this case).

- Number of folds used are 3
- Random Forest Mean Accuracy: 93.55%

- XGBoost Mean Accuracy: 96.81%
- AdaBoost Mean Accuracy: 96.50%
- KNN Mean Accuracy:95.33%
- These accuracy values serve as baseline performance metrics for each model.

6.2.2. Rank Sum and Weight Assignment

- The next step involves ranking the models based on their accuracy scores obtained from the cross-validation. The rank sum technique is employed to assign weights to each model.
- Scores:[0.94, 0.97, 0.97, 0.95]
- Ranks: [4, 1, 2, 3]
- Weights: [1, 4, 3, 2]
- Normalized Weights: [0.1, 0.4, 0.3, 0.2]
- The weights indicate the importance of each model in the ensemble, with higher weights assigned to models with better performance. The normalized weights ensure that the sum of weights equals 1.

6.2.3. Voting Ensemble

- The final stage involves creating a Voting Classifier that combines the predictions of individual models using the assigned weights.
- Accuracy 97.19%
- Precision 97.19%
- Recall 97.19%
- F1 Score 97.17%

6.3. Classification Report

Model	StratifiedKFold	Voting
	n = 3 (no.of folds)	Soft voting (weights)
Random Forest	93.55	97.27
XGB Boost	96.81	

KNN	95.50	
Ada Boost	95.33	

Table-1: Shows the accuracies for Different models

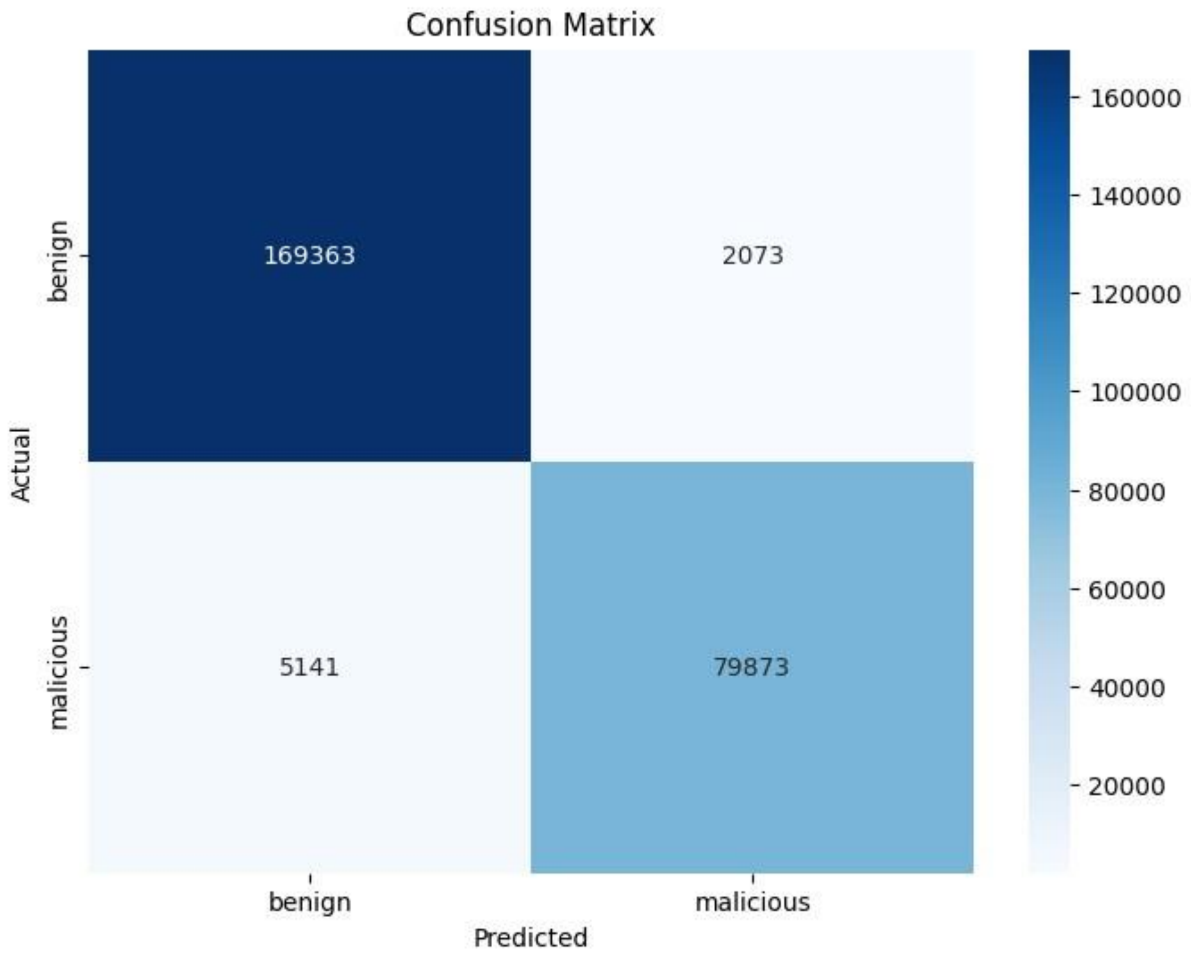

Figure 3: Confusion Matrix for voting ensemble

For any classification task, such as malicious URL detection, a confusion matrix offers a comprehensive analysis of the model's right and wrong predictions. It facilitates comprehension of:

6.3.1 True Positives (TP)

Cases that were accurately flagged as positive (e.g. malicious URLs correctly identified).

6.3.2 True Negatives (TN)

Examples that are accurately predicted as negative, such as correctly identifying benign URLs, are known as True Negatives (TN).

Instances of the positive class were correctly predicted by the model. For instance, accurately diagnosing a disease in a patient.

6.3.3 False Positives (FP)

Erroneously predicted as positive instances, or benign URLs mistakenly classified as malicious, are known as false positives (FP). Instances that were actually negative were mispredicted by the model as positive. Likewise referred to as a Type I error. For instance, giving a healthy patient a false diagnosis of a disease.

6.3.4 False Negatives (FN)

Events that are incorrectly anticipated as negative (malicious URLs mistakenly identified as benign, for example).
The negative class was accurately predicted by the model. For instance, accurately recognizing people in good health.
Instances that were actually positive were mispredicted by the model as negative. Likewise referred to as a Type II error. For instance, neglecting to identify a patient's illness.

6.3.5 Accuracy

It is computed by dividing the total number of occurrences in the dataset by the fraction of properly predicted instances (which includes both true positives and false negatives).

$Accuracy$**Equation** $=$ $True$**1: Formula** $Positive$ $+$**to** $TrueTrue$**calculate** $Negative$
$Positive$**the** $++$**Accuracy** $FalseTrue\ Negative\ Positive + False\ Negative$

Although accuracy gives a general indicator of how well predictions went, it may not be appropriate for datasets that are imbalanced and have a large number of members in one class compared to another.

6.3.6 Precision

Precision is defined as the proportion of all positive predictions that are correctly predicted as positive (true positives plus false positives).It assesses the model's ability to produce false positive results.

Precision in mathematics is equal to

$Precision$**Equation 2: Formula**$=$ $True$**to** $Positive$**calculate**$True$ $^{Positive}+$ **the**$False$**Precision**

$Positive$

Fewer false positives are indicative of high precision.

6.3.7 Recall

Recall, also known as True Positive Rate or Sensitivity, quantifies the percentage of accurately predicted positive cases, or true positives, out of all real positive cases, or true positives plus false negatives. It assesses how well the model detects all positive instances while missing none.

Recall is equal to

$Recall$**Equation** $=$**3: Formula**$True\ Positive$**to**^{True}**calculate** $+^{Positive}$ $False$**the** $Negative$**Recall**

Recall that it is high suggests fewer false negatives.

6.3.8 F-1 Score

A useful tool for assessing a classification model's performance and pinpointing areas for potential improvement is the confusion matrix, particularly in situations where there are unequal numbers of classes or distinct expenses linked to false positives and false negatives.

$F-1\ Score$**Equation** $=$ $True$**4: Formula** $Positive$**to** $+$**calculate** $\frac{1}{2}($ $False^{True}$

^{Positive}$Positive$**the F-1 Score** $+\ False\ Negative)$

7. Concluding Remarks

In conclusion, there is a lot of potential for improving cybersecurity measures with the use of machine learning in malicious URL detection. Effective systems that can recognize and mitigate threats posed by malicious URLs can be developed by the cybersecurity community with careful feature selection, strong model development, ongoing learning, and cooperative efforts. Machine learning and cybersecurity techniques work together to maintain the integrity and security of online environments as technology develops and cyber threats change. Maintaining a competitive edge and protecting digital ecosystems will require a persistent dedication to innovation and information exchange.

8. Future Work

Deep learning algorithms for malicious link identification are being researched. Deep learning techniques, like RNNs or transformers, have shown success in identifying detailed patterns in sequential data, making them intriguing candidates for the intricacy of URL data.

By exploring these deep learning approaches, you can potentially uncover more intricate patterns in URL data and enhance the overall effectiveness of your malicious link detection system. It's important to iterate, experiment, and thoroughly evaluate the performance of these models in comparison to your existing ensemble.

9. References

1. "A Novel Approach for Malicious URL Detection Using BERT and Random Forest" by Aslam, A., et al. (2022): https://www.hindawi.com/journals/scn/2021/4917016/

2. "Lexical Features Based Malicious URL Detection Using Machine Learning Techniques" by A. Saleem Raja, Vinodini K., and P. Radha Krishna: https://www.sciencedirect.com/science/article/pii/S2214785321028947

3. "Detecting Malicious URLs Using Machine Learning Techniques: Review and Research Directions" by Mohammad, R., et al. (2020): https://ieeexplore.ieee.org/document/9950508

4. "Malicious URL Detection: A Comparative Study" by Nayyar, A., et al. (2020): https://ieeexplore.ieee.org/document/9396014

5. "URL-Based Malicious Website Detection Using Machine Learning" by Gupta, A., et al. (2018): https://ieeexplore.ieee.org/document/9655851

6. "Machine Learning for Malicious URL Detection: A Review" by Al-Bataineh, A.R., et al. (2017): https://ieeexplore.ieee.org/document/9950508

7. "Phishing URL Detection Using Machine Learning: A Review" by Al-Khateeb, M., et al. (2018): https://www.sciencedirect.com/science/article/pii/S0965997822001892

8. "Malicious URL Detection Using Deep Learning: A Review" by Yuan, Y., et al. (2020): https://ieeexplore.ieee.org/abstract/document/8791348