Malicious URL Classification

Gopalkrishna Ratilal Waja

November 25, 2023

Abstract

This project tackles the classification of URLs into Safe, Phishing, or Malware categories, responding to the escalating cyber threat landscape. Conventional methods like blacklist lookups prove insufficient, leading to an exploration of statistical and machine learning techniques. Challenges include limited usable data and the inadequacy of relying solely on URL text. Drawing from prior research, the methodology involves manual feature extraction, including lexical, host-based, and content features. Various models such as SVM, Logistic Regression, Decision Trees, Random Forest, KNN, and XgBoost are trained and evaluated. The project also explores the use of GloVe word embeddings for neural network-based classification, comparing results with other methods. Experiment reveals manual feature extraction give really good performance for tree based models and also indicate superiority of pretrained + fine tuned embeddings over regular embeddings specific to the dataset.

1 Introduction

1.1 Motivation and Problem Statement

In an era dominated by digital interactions, the pervasive threat of cyberattacks looms large, with malicious URLs serving as a potent vector for various forms of cyber threats, including phishing, malware distribution, and fraud. This project endeavors to address the pressing issue of classifying URLs into three distinct categories: Safe, Phishing, or Malware. The motivation behind this initiative stems from the alarming rise in cyber-attacks conducted post pandemic through dissemination of malicious URLs. Last year alone, as reported by Cisco, these attacks accounted for 36% of all data breaches in the United States. Notably, conventional approaches, such as blacklist lookups employed by major Safe Browsing tools, are limited in their effectiveness, particularly in countering zero-day attacks. Therefore statistical and machine learnin.g techniques can be used to mitigate the limitations of the traditional approaches.

The primary objective of this project is to perform classification of URLs into Safe, Phishing, or Malware classes. The challenge lies in the limited availability of usable data, with a substantial portion of malicious URLs no longer live. Additionally, relying solely on URL text for predictions proves insufficient, necessitating a comprehensive approach to feature extraction.

1.2 Background Research

Before I dived into possible methodologies for dealing with this problem I first went over some of the past research which was done on this problem. There were many papers where URL classification was performed but in this section I have tried to summarize the two prominent papers which I feel would be useful for my project.

In the paper titled "Towards Fighting Cybercrime: Malicious URL Attack Type Detection using Multiclass Classification" an approach to multi-class URL classification is described. Leveraging boosting ensemble learners, the authors extracted 18 lexical and mathematical features from a substantial dataset, achieving an impressive overall accuracy of 0.95. The study makes suggestions for further improvement by incorporating additional discriminative features such as domain-based and javascript-based features, along with the implementation of imbalance reduction techniques like resampling.

In another paper titled "Phishing Website Detection using Machine Learning Algorithms." detection of phishing URLs is performed using a combination of 16 lexical, domain, and javascript-based features. The study, which includes the application of Decision Tree, Random Forest, and Support Vector Machine algorithms, identifies the Random Forest algorithm as the most effective, boasting an accuracy of 97.14%. The authors suggest future enhancements by extending this model to multiclass classification and integrating a blacklist with deep neural networks for improved predictions.

1.3 Methodology

Based on my research I would like to experiment with using manual feature extraction for extracting lexical features. Additionally, as suggested in future work of the above research paper, I would extract host-based, and content features from the URL strings, providing insights into length, encoding, domain properties, and webpage structure. After that, train various models, including SVM, Logistic Regression, Decision Trees, Random Forest, KNN and XgBoost. This is similar to what is done in the research paper with inclusion of the work suggested in future work.

Also, since word-embeddings have become so popular recently I would like to analyze the effect of using GloVe word embeddings for the URL for classification using neural networks. Using Neural Networks allows automated feature extraction at high-level of abstraction. Additionally I will also analyze the effect of using pretrained and fine-tuned embeddding and compare it with both the results obtained using manual feature extraction and results obtained using non-pretrained GloVe Embeddings.

Key components of the approach include lexical, host-based, and content features, along with the exploration of deep learning models and word embeddings. While the project aspires to advance URL classification, it faces the challenge of limited availability of usable data.

2 Experimental Setup

2.1 Dataset Description

For URL Classifier we have used a dataset of 41078 URLs where the URL instances for the three classes have been collected from different sources as mentioned in the table below. It must be noted that all the URLs used are live because certain features can be only extracted from live URLs, and classifying live (not blocked/taken down) is the main aim of our classifier.

CATEGORY	SOURCE	NUMBER	TOTAL
Safe - (0)	DMOZ	15,000	
Phishing - (1)	PhishTank	22,000	
Malware - (2)	URLhaus	4,078	41078

Table 1: Dataset Description for URL Classifier

2.2 Feature Extraction and Word Embedding

The dataset given above only has the URL text with their label. As described earlier it is difficult to make prediction just based of the URL text and therefore I will be extracting the following list of features: -

1. **havingIP:** Checks for the presence of an IP address in the URL. Prevents potential information theft or user confusion.
2. **haveAtSign:** Checks for the "@" symbol in the URL. Guards against potential address obfuscation by attackers.

3. **countDot:** Counts the number of dots in the URL. Doing this identifies patterns associated with malicious URLs
4. **prefixSuffix:** Checks for '-' in the domain part. Detects phishers' attempts to legitimize URLs. Eg Atacker might spoof "overleaf.com" as "over-leaf.com".
5. **redirection:** Checks for "//" in the URL path.Indicates potential redirection to another website that what is mentioned earlier.
6. **httpDomain:** Checks for the presence of "HTTPS" in the domain. Detects attempts to deceive users with "HTTPS" in the URL.
7. **tinyURL:** Checks if the URL is shortened using services like bit.ly. Uncovers potentially hidden destinations in phishing attacks.
8. **getLength:** Calculates the URL length. Malicious URLs often have longer lengths.
9. **getSlash:** Counts the number of slashes in the URL.
10. **numDigits:** Counts the number of digits in the URL.
11. **numFragments:** Counts the number of "" in the URL.
12. **numSubDomains:** Counts the number of subdomains. Malicious URLs may have multiple subdomains.
13. **domainExtension:** Returns the domain extension.
14. **url_shannon_entropy:** Calculates URL Shannon Entropy. Safe Urls have lower entropy when compared to malicious urls.
15. **domain_shannon_entropy:** Calculates domain Shannon Entropy.
16. **path_shannon_entropy:** Calculates path Shannon Entropy.
17. **query_shannon_entropy:** Calculates query Shannon Entropy.
18. **query_path_shannon_entropy:** Calculates combined path and query Shannon Entropy.
19. **suspiciousExtension:** Checks for suspicious extensions. Identifies potentially harmful extensions like '.exe', '.pif'.
20. **spacePresent:** Checks for "%20" in the URL. Safe URLs generally lack "%20".
21. **digitToLetterRatio:** Calculates digit to letter ratio.
22. **specialCharacters:** Counts special characters.
23. **suspiciousWords:** Checks for suspicious words. Phishers often use security-sensitive words for deception.
24. **domainAge:** Calculates domain age. Older domains are considered less likely to be dangerous.
25. **country:** Returns the country of the domain.
26. **imgCount:** Counts the number of images in the URL content.
27. **iframe:** Checks for "¡iframe" or "frameBorder="0"" in the URL. Detects the use of invisible iframes by phishers.
28. **mouseOver:** Checks the effect of mouse over on the status bar. Identifies potential malicious script triggers on mouse over.

In addition to these manually extracted features I have also obtained the GloVe Embedding corresponding to the URLs in the dataset. Here I have two sets of embedding, one obtained directly by fitting over our dataset and another is obtained by taking pretrained GloVe embeddings trained on Wikipedia and news data and then fine tuned on my dataset. All the embeddings have dimention equal to 300.

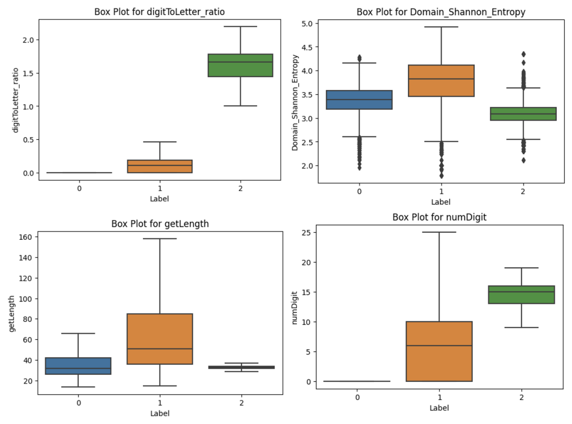


Figure 1: Box Plots for lexical features

Figure 1. reveals some interesting information about some of the lexical features. We can see that 'digit-ToLetter ratio' seems to have significantly different distribution for the 3 classes and hence this may be used as a strong predictor for classification. 'Domain Shannon Entropy' seems to have comparable/similar distribution for the 3 classes and hence this may act as a weak predictor for classification. 'getLength' seems to have comparable/similar distribution for the class 0 and 2 which are different from that of class 1 and hence this may be used to identify phishing urls. Safe url seem to have significantly lower number of digits in the url than phishing and malware urls and hence may be used as a strong predictor for identifying safe urls. Similarly more interesting observations are listed in the notebook.

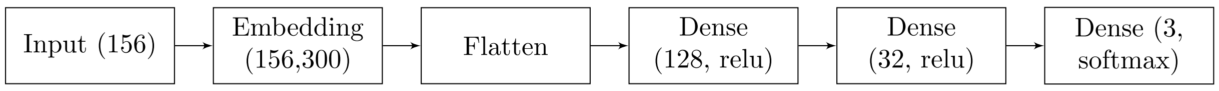
2.3 Experimental Setup

The Experiments will be run on Google Colab using scikit-learn libraries for machine learning models and Tesnsorflow framework along with Keras for deep learning. After the features extraction I will perform a 80-20 split on the data for trainig and testing. For Training machine learning models I will be performing a Grid Search to across the parameters listed below for each of the models.

Model	**Hyperparameters for Grid Search**
Logistic Regression	$C : [0.01, 0.1, 1, 10, 100]$
Decision Tree	$MaxDepth : [2, 5, 10, 20]$
Random Forest	$MaxDepth : [2, 5, 10, 20]$ $Estimators : [50, 100, 200]$
SVM	$C : [0.1, 1, 10]$ $Kernel : ['linear', 'rbf']$
XG Boost	$MaxDepth : [2, 5, 10, 20]$ $Estimators : [50, 100, 200]$
KNN	$NumberofNeighbors : [3, 5, 7, 9, 25]$

Table 2: Hyperparameters for Grid Search

For GloVe embeddings I will be using the neural network have the architecture as shown below. There is no particular reason for choosing this architecture. The main purpose of using neural networks is to study and compare the effect of using pre-trained embeddings. Note the input shape is 156 because the URL with maximum length has 156 tokens. All other URLs with shorter length are padded to match this length.



Layer (type)	Output Shape
input	(None, 156)
embedding_1	(None, 156, 300)
flatten_1	(None, 46,800)
dense_1	(None, 128)
dense_2	(None, 32)
dense_3	(None, 3)

Table 3: Model Architecture Summary

The optimizer used will be adam and the loss function will be 'categorical cross entropy'.

3 Results

The results for best parameters for each of the models as the result of grid search using 10 fold cross validation are as follows:-

Model	Best Parameters
Logistic Regression	C: 100
Decision Tree	max depth: 20
Random Forest	max depth: 20, n estimators: 200
SVM	C: 10, kernel: rbf
XG Boost	max depth: 5, n estimators: 100
KNN	n neighbors: 3

Table 4: Best Parameters for Different Models

While evaluating and comparing the model we are looking for models which have less missed detection / high recall for class 1 and 2 and at the same time low false positive rate. In other words we are looking for models with high F1 Score. The results obtained are tabulated in the table below.

Index	Model	Train Accuracy	Test Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.8823	0.8754	0.8760	0.8754	0.8756
1	Decision Tree	0.9857	0.9371	0.9373	0.9371	0.9371
2	Random Forest	0.9878	0.9550	0.9552	0.9550	0.9550
3	SVM	0.9116	0.9046	0.9081	0.9046	0.9051
4	XG Boost	0.9701	0.9546	0.9548	0.9546	0.9547
5	KNN	0.9464	0.9074	0.9084	0.9074	0.9076
6	Pretrained Embeddings + NN	0.9922	0.9778	0.9779	0.9778	0.9778
7	Normal Embeddings + NN	0.5561	0.5511	0.3832	0.5511	0.4061

Table 5: Model Evaluation Metrics - Precision, Recall and F1 scores are weighted across classes

From the results above and the confusion matrix below we can understand the following things.

Class	Precision	Recall	F1-Score
class_0	0.84	0.86	0.85
class_1	0.89	0.88	0.88
class_2	0.94	0.91	0.92
Accuracy			0.88
Macro avg	0.89	0.88	0.89
Weighted avg	0.88	0.88	0.88

(a) Logistic Regression

Class	Precision	Recall	F1-Score
class_0	0.91	0.93	0.92
class_1	0.95	0.93	0.94
class_2	0.97	0.98	0.98
Accuracy			0.94
Macro avg	0.94	0.95	0.95
Weighted avg	0.94	0.94	0.94

(b) Decision Tree

Class	Precision	Recall	F1-Score
class_0	0.93	0.95	0.94
class_1	0.96	0.95	0.96
class_2	1.00	0.98	0.99
Accuracy			0.95
Macro avg	0.96	0.96	0.96
Weighted avg	0.96	0.95	0.96

(c) Random Forest

Class	Precision	Recall	F1-Score
class_0	0.84	0.92	0.88
class_1	0.94	0.88	0.91
class_2	0.99	0.97	0.98
Accuracy			0.90
Macro avg	0.92	0.93	0.92
Weighted avg	0.91	0.90	0.91

(d) SVM

Class	Precision	Recall	F1-Score
class_0	0.93	0.95	0.94
class_1	0.96	0.95	0.96
class_2	0.99	0.99	0.99
Accuracy			0.95
Macro avg	0.96	0.96	0.96
Weighted avg	0.95	0.95	0.95

(e) XG Boost

Class	Precision	Recall	F1-Score
class_0	0.86	0.90	0.88
class_1	0.93	0.90	0.91
class_2	0.98	0.98	0.98
Accuracy			0.91
Macro avg	0.92	0.93	0.93
Weighted avg	0.91	0.91	0.91

(f) KNN

Class	Precision	Recall	F1-Score
0	0.98	0.96	0.97
1	0.97	0.99	0.98
2	0.99	0.99	0.99
Accuracy			0.98
Macro avg	0.98	0.98	0.98
Weighted avg	0.98	0.98	0.98

(g) Pretrained Embeddings + NN

Class	Precision	Recall	F1-Score
0	0.00	0.00	0.00
1	0.54	1.00	0.70
2	0.94	0.18	0.31
Accuracy			0.55
Macro avg	0.49	0.39	0.34
Weighted avg	0.38	0.55	0.41

(h) Normal Embeddings + NN

Figure 2: Classification Reports for Different models

Logistic Regression surprisingly performs well with good precision, recall, and F1-score for all classes. Notable is the high precision for class2 (malware), indicating a low false positive rate. The model achieves similar accuracy of 88% on both train and test sets.

Decision Tree, Random Forests and XG Boost shows extremely high performance across all classes with high precision, recall, and F1-score and excel in classifying malware (class2) with almost perfect precision and high recall. The overall accuracy reaches around 95%, making tree based models one of the top-performing models.

Support Vector Machine gives a balanced performance across all classes, achieving an accuracy of 90%. KNN also gives a fair performance with an accuracy of around 91%.

Utilizing pretrained embeddings significantly improves model performance. The model achieves high precision, recall, and F1-score for all classes, with an impressive overall accuracy of 98%. On the contrary Normal embeddings show poor performance. The model struggles to detect safe urls and class2 malware. This shows how pretraining with fine-tuning can significantly improve performance.

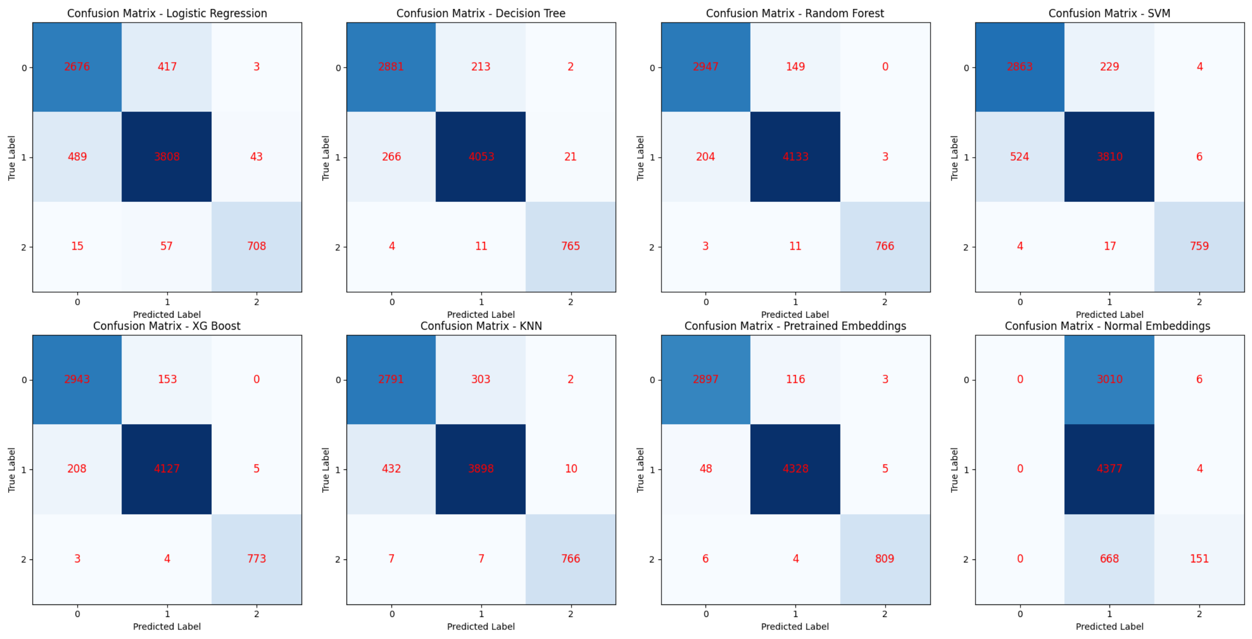


Figure 3: Confusion Matrices for different models

4 Discussion

The results obtained from the model evaluation highlight the significance of both tree-based models and pre-trained embeddings in the context of URL classification. These two approaches represent distinct strategies for feature extraction: manual feature extraction with tree-based models and automated feature extraction using pretrained embeddings and neural nets.

Tree-based models, including Decision Tree, Random Forest, and XG Boost, demonstrate outstanding performance across various evaluation metrics. These models exhibit high precision, recall, and F1-score for all classes. I probably feel that trees are able to perform well in this scenario because of their ability to decipher rule useful from classificaiton from the manually extracted feature.

Similarly, the results highlight the effectiveness of pretrained embeddings in improving model performance. The model leveraging pretrained embeddings, combined with a neural network (Pretrained Embeddings + NN), outperforms other models with an impressive overall accuracy of 98%. This signifies the importance and power of pre-trained embeddings, where the model learns meaningful representations from the data without explicit manual intervention. The pretrained embeddings capture intricate patterns and relationships in the input data, providing a rich source of information for the downstream classification task. On the other hand, the model utilizing normal embeddings without pretraining exhibits poor performance.

Having said that receiving such a high accuracy, both in case of trees and pre-trained embeddings might be deceptive and performance needs to be verified using A/B testing and testing the model on URLs taken from different datasets.

5 Conclusion

In summary, in this project I experimented with several ML algorithms with manual feature extraction and found that tree based models give a very high performance for the classification task. Additionally, pretrained embeddings for automated feature extraction proves to be another powerful strategy for URL classification. While tree-based models excel in capturing explicit patterns in the data, pretrained embeddings enhance the model's ability to generalize and capture intricate relationships. Future work could involve, exploring imbalance reduction techniques like resampling, SMOTE and Near Miss Algorithm to enhance the model's robustness.

6 Reference

[1] T. Manyumwa, P. F. Chapita, H. Wu and S. Ji, Towards Fighting Cy- bercrime: Malicious URL Attack Type Detection using Multiclass Clas- sification; 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 1813-1822, doi: 10.1109/BigData50022.2020.9378029.

[2] Rishikesh Mahajan and Irfan Siddavatam, Phishing Website Detec- tion using Machine Learning Algorithms; International Journal of Com- puter Applications 181(23):45-47, October 2018.

[3] Y. -C. Chen, Y. -W. Ma and J. -L. Chen, Intelligent Malicious URL Detection with Feature Analysis, 2020 IEEE Symposium on Computers and Communications (ISCC), 2020, pp. 1-5, doi: 10.1109/ISCC50000.

[4] Ramakrishnan, R. (2021) URL feature engineering and Classification, Medium. Available at: https://medium.com/nerd-for-tech/url-feature-engineering-and-classification-66c0512fb34d (Accessed: 23 November 2023).