Malicious URL Detection and Classification

Antonio Alonso*, Jonah Bishop†, Sammy Fellah*, Yashit Agarwal*
*Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, Virginia 24060
Email: antonioa19@vt.edu, sammyfellah@vt.edu, yashitagarwal@vt.edu
†Department of Computer Science, Virginia Tech, Blacksburg, Virginia 24060
Email: jonahbishop@vt.edu

I. Abstract

Malicious URLs have been a common and serious problem in cybersecurity for decades. The ongoing dangers that these fraudulent websites pose go beyond ordinary viruses; firms and individuals are being robbed everyday from scamming tactics, phishing attacks, and other luring attempts made by hackers to steal private information. This report will cover the specifics about the type of dataset utilized, the data pre-processing and transformation, the various machine learning methods we explored, and lastly, the different types of Neural Networks built from scratch.

TABLE I
Dataset URL Breakdown

URL Type	Count
Benign	428,103
Malware	96,457
Defacement	94,111
Phishing	32,520

The dataset contains over 600k URLs and its appropriate Label. This project created many complex deep learning models which out performed baseline classification methods for both Binomial and Multi-Class classification. The highest Binomial Classification model achieved an F1 score of 0.94 and highest Multi Class achived an F1 Score of 0.7757.

II. Data transformation / Pre-processing

A. Feature Extraction

For the first preprocessing approach, we opted to manually generate all of the features based on random things identified as potential indicators for a URL being safe or unsafe. The following were deemed useful features: URL length, backslash count, http type, subdomain, periods in suffix, domain, trimmed suffix, contains percent, and file extension. Once we had established the features, we needed to transform a lot of them from categorical features to numeric ones. Initially, we opted for One Hot Encoding to maximize the number of features and potentially improve performance. Unfortunately, for many of the features such as domain, subdomain, there were too many unique values so One Hot Encoding was able to be utilized. Instead, we mapped the three to five most common values to a corresponding number and made it zero if it did not fit in a category. Useful to this feature extraction were several libraries made for parsing out information from URLs. Each numerical feature was normalized by the largest value to have a range of [0 ... 1] for improved model performance.

B. URL Vector Hashing Representation

The second preprocessing approach taken was using sklearn's Hashing Vectorizer for feature extraction. This approach converted the URLs into a matrix of token occurrences. It turns the strings into sparse matrices with token frequency normalization. Advantages to using this method includes low memory usage due to not storing the dictionary of words into memory. The number of features chosen to represent a URL was 500.

1) Principal Component Analysis (PCA) Analysis: The computation time is linear due to the number of generated features therefore PCA was used to find the optimal number of features per URL.

C. Character Embedding Representation

TABLE II
Character Embedding

char	'a'	'b'	'c'	...	'y'	'z'	'!'	'\'	'.'	'?'	'ukn'
map	1	2	3	...	53	54	55	56	57	58	59

The final preprocessing approach taken was to use a character level embedding of each URL. This embedding is then inputted into a one-dimensional convolutional neural network for inference. Firstly, each of the URL's were all converted to lowercase in preparation for the embedding. A Tokenizer was used with a dictionary of all lowercase ascii characters as well a string punctuation to give a character to number representation. Since each URL is of different lengths, each URL was then padded to the longest URL size so that each input vector is of the same shape.

III. Classification Techniques

Given the fact that our team had four members, we did several things to increase the scope to make this viable for a group our size. One of the first things we did was break it into binary classification where we are simply trying to classify a URL as either safe or unsafe versus multi-class classification of each of the specific unsafe URL types. Another thing to increase the scope was using the baseline classifiers from sklearn and using TensorFlow with Keras to build complex NN.

A. *Binomial Classification*

TABLE III
DATASET URL BINOMIAL ENCODING

URL Type	Encoding
Benign	0
Malware	1
Defacement	1
Phishing	1

One type of classification used in this project was the detection of either a safe vs. unsafe URL. The Benign URLs were encoded into 0's and the Phishing, Malware, and Defacement URLs were encoded into 1's. This was done to determine classification on a Binomial scale to then compare to Multi-Class Classification.

B. *Multi-Class Classification*

TABLE IV
DATASET URL MULTI ENCODING

URL Type	Encoding
Benign	0
Malware	1
Defacement	2
Phishing	3

Another type of classification used in this project was the detection of each of the classes in the dataset. These different classes were then encoded to [0 .. n].

C. *Classification Methods*

- Logistic Regression ($L2 - regularization$)
- Random Forests ($maxdepth = 10, criterion = gini$)
- Gaussian NB
- K-Nearest Neighbors ($n = 5$)

IV. MODEL ARCHITECTURE

A. *Sequential Neural Network*

The training data was trained on a deep five layer sequential Neural Network with a dense layer of size one or four based on either binomial:

$$BinomialEntropy : -(y \log(p) + (1 - y) \log(1 - p)) \quad (1)$$

$$CategoricalCrossEntropy : -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \quad (2)$$

The model either used a categorical or binary cross entropy loss function based on classification type and used the Adam optimizer function.

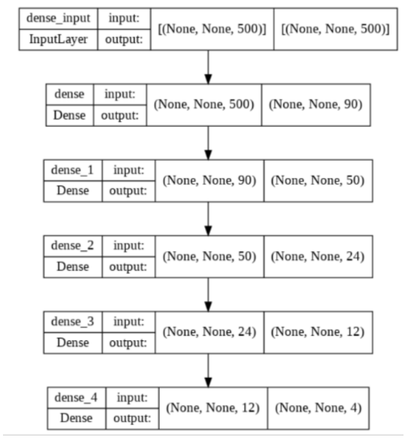
dense_input | InputLayer | input: | output: | [(None, None, 500)] | [(None, None, 500)]
dense | Dense | input: | output: | (None, None, 500) | (None, None, 90)
dense_1 | Dense | input: | output: | (None, None, 90) | (None, None, 50)
dense_2 | Dense | input: | output: | (None, None, 50) | (None, None, 24)
dense_3 | Dense | input: | output: | (None, None, 24) | (None, None, 12)
dense_4 | Dense | input: | output: | (None, None, 12) | (None, None, 4)

Fig. 1. Multi-Class Sequential Model developed using Tensorflow with Hashed String URL input.

1) Feature Based Neural Network: A sequential model was developed in python using the front-end Keras API with a TensorFlow back end using Google Co-lab. This model used the Feature based pre-processing technique where each URL was transformed into a vector of shape (8,) which visualized each of the features that were selected for analysis.

2) URL Hashed Based Neural Network: This model uses the same model architecture as the Feature based network however uses a different input shape of (500,). As explain in the prepossessing section each section was converted into a vector of five hundred features to represent each URL. These URLs were then passed in to the network for model training and testing.

B. *Convolutional Neural Network (CNN)*

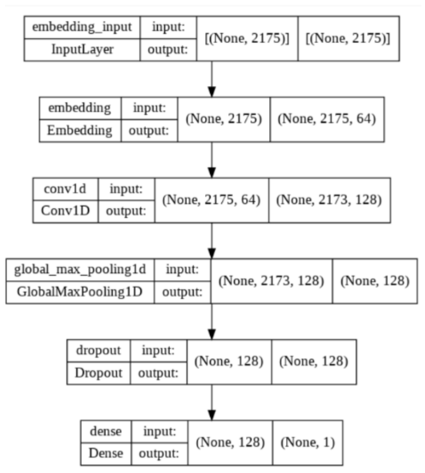
embedding_input | InputLayer | input: | output: | [(None, 2175)] | [(None, 2175)]
embedding | Embedding | input: | output: | (None, 2175) | (None, 2175, 64)
conv1d | Conv1D | input: | output: | (None, 2175, 64) | (None, 2173, 128)
global_max_pooling1d | GlobalMaxPooling1D | input: | output: | (None, 2173, 128) | (None, 128)
dropout | Dropout | input: | output: | (None, 128) | (None, 128)
dense | Dense | input: | output: | (None, 128) | (None, 1)

Fig. 2. CNN Model developed using Tensorflow.

A 1D Convolutional Neural Network was developed to handle the Character Embedding Representation of the URLs. These URL vectors of shape (URL max length,) is applied to an embedding layer which then becomes a 2D vector of shape (2175, 64,). The longest URL size is 2175 characters long with an embedding space of 64. The embedding are then 1D convoluted with 128 filters and then Max Pooled. Then through a dropout layer to increase computation time and reduce over fitting. Then using a fully connected Dense layer to then classify the URL. The last layer uses the sigmoid funtion to classify the label.

V. Evaluation Metrics

- Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$
- Precision = $\frac{TP}{TP+FP}$
- Recall = $\frac{TP}{TP+FN}$
- F1 = $\frac{2*Precision*Recall}{Precision+Recall} = \frac{2*TP}{2*TP+FP+FN}$

VI. Performance and Analysis

A. Binomial Performance

TABLE V
Performance for all Binomial Models

Model Name	Accuracy	Precision	Recall	F1 Score
Random Forest	0.952	0.975	0.8829	0.9265
Gaussian NB	0.8507	0.772	0.800	0.786
KNN	0.9268	0.9709	0.8107	0.8835
Logistic Regression	0.8824	0.8831	0.757	0.8153
Feature Model	0.94	0.945	0.915	0.925
Hashing Model	0.94	0.94	0.92	0.925
CNN Model	0.9494	0.945	0.94	0.94

Model with best performance: CNN Model with an F1 Score: 0.94. This model had the best accuracy overall compared to both the baseline models as well as the other neural networks.

B. Multi-Class Performance

TABLE VI
Performance for all MultiClass Models

Model Name	Accuracy	Precision	Recall	F1 Score
Feature Model	0.8984	0.5575	0.9025	0.6475
Hashing Model	0.9256	0.7175	0.965	0.7775
CNN Model	0.9265	0.705	0.915	0.7575

Model with best performance: Hashing Model Approach with an F1 score of 0.77. This model was slightly better over the CNN model but was pretty relative in performance. Classification between Phishing', 'Defacement', and 'Malware' were hard to differentiate due to possibly because of a small amount of sample's relative to the benign sample size.
Hashing model Performance Breakdown:

VII. Conclusion and Future Work

Detecting malicious URLs is a critical component in the cyber-security industry, and clearly Machine Learning approaches can dramatically improve the detection and classification process with certain types of techniques. In our experiment, we were able to successfully categorize malicious

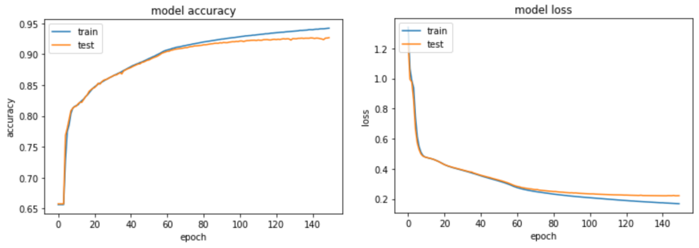

Fig. 3. Model Accuracy and Loss for Hashing Multi Class Classification.

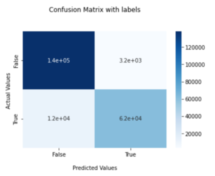

Fig. 4. Confusion Matrix for Hashing Multi-Class Model.

from benign URLs by using several types of models. PCA decreased the performance and increased the efficiency of the hashing models. However, the computation time was not worth the drop in our model performance. This report highlights how for both binary and multi-class classification, our custom built neural networks significantly outperformed the baseline and ensemble classifiers. For our binomial performance, our best model was the CNN-based model, which yielded an F1 score of 0.94; our worst model was the Gaussian NB, which salvaged an F1 score of 0.786. For multi-class performance, our best model was the hashing model, which had an F1 score of 0.7775. However, our worst model was the feature-based model, giving us an F1 score of 0.6475. Though we yielded many great results, if we were given access to GPUs with large calibers of RAM in addition to more malicious URL samples, we would be able to retrieve higher accuracy's and F1 scores, hence improve the experiment as a whole.

VIII. Team Member Contributions

TABLE VII
Performance for all MultiClass Models

Team Member	Contributions:
Yashit Agarwal:	Baseline Classification Training, Cross Validation, PCA, Eval.
Antonio Alonso:	Character Embedding, CNN and Neural Net models, Eval.
Jonah Bishop:	Feature Engineering, Baseline classifier Training, Eval.
Sammy Fellah:	Hashing Vectorizer, Sequential Neural Network, Eval.

References

[1] M. Siddhartha, "Malicious urls dataset," Kaggle, 23-Jul-2021. [Online]. Available: https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset. [Accessed: 25-Apr-2022].
[2] "An introduction to machine learning with scikit-learn," scikit. [Online]. Available: https://scikit-learn.org/stable/tutorial/basic/tutorial.html. [Accessed: 25-Apr-2022].
[3] K. Team, "Keras Documentation: The sequential model," Keras. [Online]. Available: https://keras.io/guides/sequential model/. [Accessed: 25-Apr-2022].