Predicting Phishing URLs

Jordan Waldroop
Leeds School of Business
University of Colorado - Boulder
Boulder, Colorado
jordan.waldroop@colorado.edu

Jack Beck
Leeds School of Business
University of Colorado - Boulder
Boulder, Colorado
jack.beck@colorado.edu

***Abstract*—The goal of this project is to take a dataset created using a current URL feature extraction algorithm and use it to achieve a machine learning goal of creating a model that can predict whether or not a given URL will lead to a phishing website based on the characters within the URL itself.**

***Keywords*—phishing, URL, models, random forest, neural network, features**

I. Introduction

This project aims to expand upon the work of a previous research study from the scientific journal "Data in Brief", with the goal of creating classifier models to predict phishing URLs. Phishing is a type of cybercrime where an attacker contacts targets through email, text message, or telephone and attempts to steal the target's personal or financial information by posing as a legitimate company or institution.

Oftentimes attackers engaging in phishing attacks will create websites that closely resemble their legitimate counterparts. The attacker will then send the target a URL link to their mock website via email or text, with a message mimicking that of the legitimate company, encouraging the target to click on the malicious URL. From here the target may be prompted to give up sensitive information, with them believing that they are simply providing the information to the legitimate service provider.

Generally speaking, there are often subtle giveaways that the URL may in fact be a phishing scam. For example, if an attacker were attempting to phish login credentials for a website: myuniversity.edu/login, they might disguise their phishing URL as myuniversity.edu-login.com. These two URLs both look very similar, and at a glance the phishing URL can easily trick unsuspecting users into giving up their personal information.

As a result of the subtle discrepancies between legitimate and phishing URLs, our team decided to create and train machine learning models to predict whether a given URL is a phishing scam or a legitimate website based on metrics specific to the URL. The data article that this project is based on contains a dataset which is composed of thousands of individual URLs, with each one broken down into 111 different features, each pertaining to a different metric or parameter specific to every URL. Details about the features and their extraction, as well as the machine learning models and their evaluation, will be covered throughout this report.

II. Business Understanding

Phishing was the most prevalent form of cybercrime and 74% of US companies were the victim of a successful phishing attack in 2020 according to the FBI [1], [2]. Though not all were successful, it is estimated that 75% of companies worldwide experienced a phishing attack. With the average breach costing an organization $3.92 million, the importance of preventative measures is only growing [1], [2].

According to the Anti-Phishing Working Group's 4th Quarter 2020 Trend Report, phishing attacks doubled over the course of 2020, peaking in October [3]. Business email compromise wire transfers averaged $75,000 in Q4, though the highest average for 2020 was reported in Q2, at approximately $80,000 [3]–[5]. Financial institutions and SaaS/Webmail services were the two most frequent industry sectors to be targeted throughout the entirety of 2020 [3]–[6].

There have been many companies and organizations that have developed software to monitor for phishing attacks, with many of those models being AI-based to allow for continuous learning [6], [7]. Previous studies cited by our primary source used swarm intelligence to further improve on "manually tuned deep neural network[s]", which only had an maximum accuracy of 3.8% with an F1 score of 24% [7].

III. Data Understanding

As previously mentioned, the dataset used in this project comes from the work of a previous research study "Datasets for phishing websites detection", which was published in the scientific journal "Data in Brief" by a team of Slovanian researchers. The dataset was created for machine learning classifier use in predicting phishing websites. The data article contains two different variations of the data, the "full" version and the "small" version. The small version was created to be relatively balanced between the target classes, with 30,647 observations labelled as phishing websites and 27,998 observations labelled as legitimate websites. The full version of the dataset contains 30,647 observations labelled as phishing websites and 58,000 observations labelled as legitimate websites, for a total of 88,647 observations. The full dataset was designed with the intention of replicating a more realistic scenario, where there are more legitimate websites than phishing websites. For the purposes of our machine learning project, our team chose to only work with the full dataset, as our machine learning models would ideally be

used in a realistic/practical setting. The confirmed phishing URLs in the dataset were obtained from Phishtank, a free community website where users can report phishing websites they encounter, which are then recorded so that the phishing sites can easily be verified and tracked. The legitimate URLs in the dataset were obtained from Alexa, an Amazon website which records data on global internet traffic.

The dataset was made using a feature extraction algorithm that was originally created by a Brazilian GitHub user, and was subsequently modified by the authors of the dataset. The algorithm breaks input URLs into a multitude of different features based on the URL properties, URL resolving metrics, and external services that are relevant to the given URL. The features extracted by the algorithm each belong to one of six feature subsets, which are defined in the original data article as tables one through six. Each table contains features that are specific to a certain part of the input URL or resolving metrics and external services relevant to the input URL. A visual representation of these different URL subsets can be found in the appendix.

Table 1 contains 20 unique features that are based on properties of the entire URL, with the vast majority of its features being different quantities of specific characters in the URL, I.E. quantity of slashes (/) in the URL. Tables 2, 3, 4, and 5 all contain similar features to table 1, but with all the attributes being specific to the different sections of the URL. Table 2 features are specific to the "domain" section of the URL, Table 3 features are specific to the "directory" section of the URL, Table 4 features are specific to the "file" section of the URL, and Table 5 features are specific to the "parameters" section of the URL. Finally, Table 6 contains features that differ from the rest of the tables. Rather than containing attributes specific to a particular section of the URL, Table 6's features pertain to different resolving metrics and external services that are relevant to the input URL, i.e., the domain's lookup time response, or the domain's activation time in days. A list of all the tables and their specific features is provided in the appendix.

IV. Data Preparation

Thanks to the relatively robust feature extraction algorithm, the data preparation required for machine learning use was fairly light. Upon initially reviewing the dataset and the extraction algorithm, an error within the data was identified. Certain feature values, e.g., the quantity of a certain character, had values of -1 as a result of an error in the extraction process. A negative value in these features should be impossible, as there can't be a negative count of a character. The occurrences of this error were rare, so our team decided it was best to simply replace these negative values with a zero. From here, a unit test was performed to ensure that the imputations had been performed correctly across the different features. Beyond this simple imputation, the only other data preparation performed was indexing the data to create the different table subsets, as outlined in the data article.

V. Models

When deciding which modeling techniques to use for this project, our team decided it would be best to have two different modeling approaches, one which utilized a traditional machine learning model, and one that employed the use of deep learning models. Ultimately, the team settled on a Random Forest Classifier for the traditional model, and a Sequential Neural Network Classifier for the deep learning model.

A. Random Forest Classifier

A Random Forest Classifier model was chosen to be used as the machine learning model in this project as our preliminary data analysis indicated that there was some multicollinearity between some of the features in the dataset. Random forest models are well suited to handle data with multicollinearity issues as the model naturally avoids these issues due to its underlying mechanics. To construct and evaluate the model, the scikit-learn machine learning library for Python was used. Initially, a basic exploratory model was created, which utilized all features present in the dataset and default model parameters. The data was split into train and test sets using sklearn's `train_test_split()` function, which split the data into a training set consisting of 75% of the data and a test set consisting of the remaining 25%. From here, the model was evaluated and the number of estimators and max depth of each estimator were changed from their default parameters to optimize the model's performance.

Following the preliminary model, the data was subset based on the tables defined by the scientific journal and each of the subsets was split into train and test for use in their own model. The goal for these models was to see if there was any specific subsection of the URL metrics which had greater predictive power than the others. Each of these models employed the same parameters used in the initial comprehensive model. The relevant performance metrics for all models can be found in the evaluation section below.

After evaluating each of the table subset models, a final iteration of the model was created for use in future deployment of the project. Based on the performance metrics of the initial comprehensive model and the individual table subset models, it was determined that the final model should be a further refinement of the comprehensive model. To do this, the number of input features for the model was reduced from 111 features to 50 features using sklearn's recursive feature elimination function (RFE) from the feature selection module. The RFE function initially trains the model on the entire feature set and ranks each feature based on importance. The lowest performing features are then pruned from the feature set. The function performs the process recursively until the desired feature count has been reached.

B. Sequential Neural Network Classifier

For the neural network, the Keras library, which operates on TensorFlow's structure and backend, was used to build a Sequential model. The reason the sequential neural network was chosen was because the layers have only one input tensor

and one output tensor, and with so many features, the neural net will be able to detect patterns and connections between the features to more accurately predict the target variable. Additionally, we thought it would be beneficial to see if we could improve on the metrics reported a prior study that was performed using a neural network with swarm intelligence to detect phishing URLs [7].

The first step required before building the neural network was to set the X and y variables and complete a training/testing split. Using sklearn's `train_test_split()` function, the data was split into a training set consisting of 75% of the data and a test set consisting of the remaining 25%. The training/testing split is incredibly integral to this model's integrity, which will be explained in more detail below. While the other model in this project uses the RFE function, after multiple iterations using the individual tables and the RFE selected features, it was found that the neural network performs best when given all 111 features.

In the Sequential Keras model, there are 26 layers in total. These layers are: the input and output layers, 8 dense layers, 8 batch normalization layers, 7 dropout layers, and a flattening layer directly prior to the output layer application. Each dense layer and the output layer have activation functions applied; the dense layers are activated with a Rectified Linear Unit (ReLU) activation function and the output layer has a Sigmoid activation function applied to keep the probabilistic predictions between 0 and 1. The neurons in each of the dense layers are as follows, respectively: 64, 64, 50, 32, 32, 16, 16, and 111. Since the dense layers connect the nodes between each dense layer, batch normalization is called after each dense layer to normalize the inputs going into the next dense layer.

Before those inputs are passed into the next dense layer, but after batch normalization, a dropout layer is applied to remove a certain amount of neurons so that the model is forced to relearn those dropped aspects and so that neighboring neurons are not too dependent on others; this allows for mitigation of overfitting and better generalization to new data. The first 5 dropout layers drop 20% of neurons and the last two dropout layers drop 40% of neurons. The flatten layer reshapes the incoming tensor to equal the number of elements in the following tensor.

The model was compiled using the Adam algorithm optimizer, in large part because of the algorithm's ability to handle noise and the computational efficiency. The loss function selected for measurement is binary cross-entropy, calculating the cross-entropy loss between true labels and predicted data. This is a probabilistic loss measure. The metrics measured are: binary accuracy and AUC. A callback was applied to monitor the validation data's maximum binary accuracy with a patience of 25 epochs, in addition to restoring the best weights measured. Having an early stopping callback is integral in this model because of the parameters of the model fit.

The model fit uses the variables train_X and train_y, with a validation split of 0.30 with shuffle set to 'True'. The purpose for the validation split and shuffle is for the model to take the training data (75% of the dataset) and further split it (70/30) for model fitting. In addition to now having the data split into three sets (52.5% - training, 22.5% - testing, 25% - validation), the training and testing set were shuffled, or certain rows alternated between training and testing, after each epoch. It is important to note that the original validation split for X and y are not used at all during the model fit. The data split ends up being 52.5% training, 22.5% validation, and the remaining 25% being held out for model evaluation and predictions. This has been done in the hopes of mitigating overfitting.

The batch size (how many rows are fed into the model at a time) is set to 1500. The model is set to run for 250 epochs, which is where the early stopping callback function plays an important role. The large batch size will allow for faster computation. The Adam optimizer algorithm's learning rate was not changed from 0.001 to retain algorithm integrity. With the early stopping callback applied, no matter the number of features, the model never came close to completing all epochs.

Again using the TensorFlow Keras library, the `evaluate()` function was called post-model fitting, where both the testing and validation data were evaluated on what the model had learned during fitting. This is the first time the model encountered the original 25% validation data. The `evaluate()` function returned the binary cross entropy (as 'loss') and accuracy.

In order to ensure model integrity and mitigation of overfitting, five-fold cross-validation scoring and predictions were performed after model compilation. The model was originally created as a Sequential object and for better deployment and compatibility, the model was then built inside a user-defined function. For cross-validation to work correctly, the model (as a user-defined function), was fed into the `KerasClassifier()` function, which is a TensorFlow developed wrapper for sklearn compatibility. The ability to feed the model into this wrapper allows for gathering of more metrics and descriptive statistics of the model, particularly since neural networks are generally black-box and the connections and operations are performed independently by the model. In total, the model has the ability to train 20,835 parameters. See "Fig. 1" for a visual depiction of the model structure.

The model fitting of the various tables and features is located in the `phishing_nn.ipynb` file, with the final model located in `NN_final_model.ipynb`. In addition, the deployable version of the model, which can be run from the command line, can be found in the file titled `deployable_nn.py`. All of these are available in the GitHub repository.

VI. Evaluation

A. Random Forest Classifier

Evaluation of the various random forest models began with the preliminary model which incorporates the entire dataset. This model performed the second best out of all model iterations, having a mean accuracy on the test data of 0.964, an F-1 Score of 0.949, and an average precision of 0.988.

Following this, performance metrics for each of the table subset models were calculated.

To reduce clutter in this report, most of these performance metrics will be withheld, but can be found in the random forest models Jupyter notebook, along with each model's respective ROC curve and precision-recall curve. Overall, all of the table subset models performed worse across all metrics than the comprehensive model. The table subset models ranked from best to worst is as follows: Table 1, Table 3, Table 6, Table 4, Table 2, and Table 5. Table 1, which is composed mainly of the quantities of certain characters across the whole URL, performed the best, suggesting that the count of characters across the entire URL is more important than the count in one particular subsection of the URL. Another interesting insight gathered from the subsetted models comes from the performance of Table 6, which performed very well, with a mean accuracy of 0.88. Table 6 contains features that are specific to metrics regarding resolving the URL and external services associated with it. The high performance of Table 6 suggests that certain data outside of the URLs characters could have significant importance in the random forest model. Specific important features will be expanded upon in the final model's evaluation.

The final model is a revised version of the initial comprehensive model which utilized recursive feature elimination to remove unnecessary features from the dataset. The final produced marginally improved performance metrics over the initial model. The model scored a mean accuracy of 0.967, an F-1 Score of 0.953, and an average precision of 0.990. Finally, a 5-fold cross validation was performed to verify the accuracy of the model. The cross validation produced an average accuracy of 0.967.

B. Sequential Neural Network Classifier

The neural network model first had to be evaluated for predictive efficiency, which involved altering various layer parameters, changing the training and testing split sizes, adding/removing layers, and altering the batch size and number of epochs during compilation. Once the appropriate model parameters had been established, the features to be used had to be evaluated.

As mentioned above, the model was fit separately against each table's subset of features, along with the features extracted from the RFE. While the metrics would be too lengthy to list here, the table-subsetted features and the RFE selected features did not perform as well as when the model was applied to the entirety of the dataset. Those metrics are available in the `\phishing_nn.ipynb"` file in the GitHub repository. If one wanted a model that did not use all 111 features, the next best iteration of the model would be to use the top 50 features selected from the RFE functionality, with an accuracy of 92.45% and binary cross entropy of 20.26%. The model iterations of the individual tables, ranked from best to worst, are as follows: Table 1, Table 3, Table 4, Table 6, Table 2, Table 5. The features included in Table 1 are general URL metrics that count certain characteristics included in the URL. When the validation dataset for Table 1 was evaluated, the metrics returned at 90.14% accuracy with binary cross entropy loss measured at 24.47%. The validation data evaluation of the model for Table 5, which was the worst performing feature set the neural network evaluated, returned 72.55% accuracy and 57.66% binary cross entropy loss.

In the final iteration of the Keras Sequential neural network model, there was marginal improvement in the metrics compared to the model version with the top 50 most important features from the RFE package as discussed hereinabove. When the Sequential model, as an object, was evaluated using the 25% held-out validation split and the Keras `evaluate()`, the binary cross entropy loss was measured at 0.2080, with an accuracy of 0.9254, an AUC of 0.9726. With the Keras `predict()` function, probabilistic predictions are returned for X. The model was given the 25% validation data to predict on. Subsequently, the same prediction function was called and given a threshold of greater than 0.5 to qualify as phishing. After these predictions were made, the model returned an accuracy score of 0.9254, an F1 score of 0.8931, a precision score of 0.88, a recall score of 0.904, and an AUC of 0.973. The prediction scores aligning with the `evaluate()` metrics is a good indication that all the functions are reporting accurately. These metrics are incredibly positive that the model is not overfitting the data, though there is still room for improvement.

When the model is built as a function and ran inside the `KerasClassifier()` wrapper, the average of the five-fold Stratified K-Fold cross validation accuracy returns at 0.9248, with a standard deviation of 0.004. This low of a standard deviation is significant in that the cross validation accuracy results give validation to the model metrics previously measured and gives confidence that the model is not overfitting. It is important to note that the cross validation scoring and predictions were done on the entirety of the dataset. Using sklearn's `cross_val_predict()` function, five-fold cross-validated predictions were made, returning a 0 or 1. The model returned an accuracy score of 0.9232, an F1 score of 0.8918, a precision score of 0.869, a recall score of 0.916, and an AUC of 0.921.

The accuracy and precision metrics returning marginally different scores, with the exception of the AUC, for the model evaluation and during cross validation is a very positive sign. The AUC seemed high in the original model evaluation when looking at the loss and accuracy metrics, whereas the cross validation AUC seems much more appropriate given the other metrics. One of the major benefits of deploying a neural network is that it can continuously learn; with the baseline for this model being approximately 92% accuracy, there is plenty confidence in the model, yet still room to learn. The most glaring issue with this neural network model is the runtime. When running on the CPU, the deployable file takes between 4 to 8 minutes. With a GPU, the runtime is closer to 15 minutes.

C. Feature Evaluation

Following the evaluation of our models, our analysis turned towards trying to identify which features in the final feature set were of particular importance to the model. To do this, the SHAP (SHapley Additive exPlanations) module was applied to the final random forest model. Unfortunately SHAP does not support deep learning models. This module helps to explain the output of machine learning models using a game theory approach, more can be found about the module at its GitHub page. Using the SHAP values, the top 5 most important features were identified: `qty_slash_url`, `directory_length`, `qty_slash_directory`, `time_domain_activation`, `length_url`. The top 5 most important features identified by SHAP perfectly aligned with the expectations suggested by the table subset models. Two of the top 5 features, `qty_slash_url` and `length_url`, are part of the Table 1 subset. This subset was the highest performer of the subset models and contained features relating to counts of characters in the entire URL. The SHAP output suggests that the quantity of slashes (/) in the URL as well as the total length of the URL are the most important features from this table.

Two of the top 5 features, `directory_length` and `qty_slash_directory`, come from the Table 3 subset, which was the second best performer among the subset models. This subset contained features relating to the counts of character in the directory subsection of the URL. Similar to the Table 1 features, the SHAP output suggests that the quantity of slashes in the directory as well as the directory's overall length are most important in determining whether the URL is a phishing attack. URLs with long lengths and a high quantity of slashes are more likely to be phishing attacks. The remaining feature from the top 5 features is `time_domain_activation`, which is part of the Table 6 subset. The Table 6 subset was the third highest performing of the table subset models and its features are specific to metrics regarding resolving the URL and external services associated with it. Based on the SHAP output, a URL with a high domain activation time is substantially less likely to be a phishing URL. Logically, this makes a lot of sense, as phishing sites are generally only deployed for use in the phishing attack, and once the attacker is satisfied or their attack is uncovered, the site is removed. As a result of this, phishing sites typically have a low domain activation time. A visualization illustrating the top features can be found below in "Fig. 2", and the SHAP values for all models can be found in the random forest Jupyter notebook.

VII. Deployment

The ultimate goal of this project is to create a web application that incorporates both the feature extraction algorithm from the original data article and the project's machine learning and deep learning models. Ideally this would allow public users to input a given URL(s), the extraction algorithm would then extract the relevant features from the input URL(s) and feed them into both the random forest and the neural network, which would then return a prediction to the user.

It is worth noting that while the random forest model is currently performing better than the neural network, this may change once the models are deployed. A benefit of the neural network is that it can continuously learn, therefore, the more data the model takes in, the more knowledgeable the model becomes.

To take steps to achieve this ultimate project goal, our team has created two .py files that each contain a "deployable" version of their respective models. These could be added into a pipeline which contains the original extraction algorithm which could then be integrated into the web application. However, this project is still a ways off from achieving this ultimate goal, and due to lack of time left in the semester we will have to save it for a later date.

GitHub Repository

https://github.com/jwaldroop/phishing-url-project

Acknowledgment

We would like to thank the authors of the previous studies who provided great insight and a solid foundation to start our project. Additionally, we would like to thank our professors, Dr. Eric Goodman and Dr. Dave Eargle, who have given us the knowledge and assistance to complete this undertaking.

Finally, we would like to thank our MSBA Security Analytics cohort who have been great support systems throughout the entirety of the program.

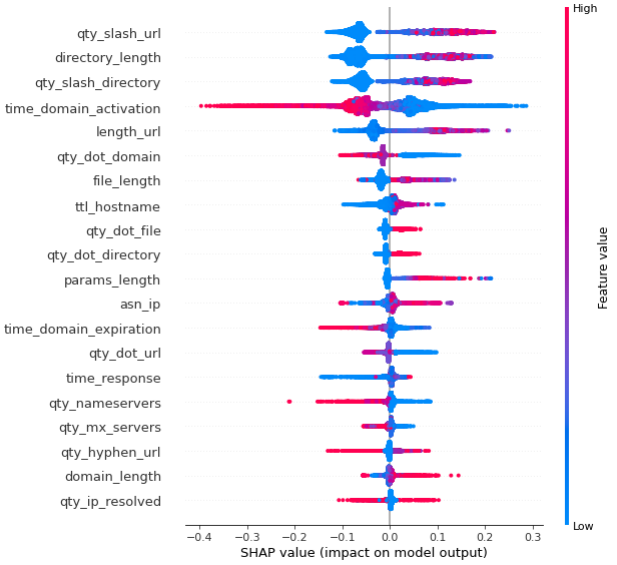
Figures and Tables

Fig. 1. Sequential Neural Network Model Summary

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	7168
batch_normalization (BatchNo	(None, 64)	256
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 64)	4160
batch_normalization_1 (Batch	(None, 64)	256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 50)	3250
batch_normalization_2 (Batch	(None, 50)	200
dropout_2 (Dropout)	(None, 50)	0
dense_3 (Dense)	(None, 32)	1632
batch_normalization_3 (Batch	(None, 32)	128
dropout_3 (Dropout)	(None, 32)	0
dense_4 (Dense)	(None, 32)	1056
batch_normalization_4 (Batch	(None, 32)	128
dropout_4 (Dropout)	(None, 32)	0
dense_5 (Dense)	(None, 16)	528
batch_normalization_5 (Batch	(None, 16)	64
dropout_5 (Dropout)	(None, 16)	0
dense_6 (Dense)	(None, 16)	272
batch_normalization_6 (Batch	(None, 16)	64
dropout_6 (Dropout)	(None, 16)	0
dense_7 (Dense)	(None, 111)	1887
batch_normalization_7 (Batch	(None, 111)	444
flatten (Flatten)	(None, 111)	0
dense_8 (Dense)	(None, 1)	112

Total params: 21,605
Trainable params: 20,835
Non-trainable params: 770

Fig. 2. SHAP values of important features



REFERENCES

[1] M. Rosenthal, "Must-Know Phishing Statistics: Updated 2021," Tessian, 10 February 2021, https://www.tessian.com/blog/phishing-statistics-2020/.

[2] Internet Crime Complaint Center, "Internet Crime Report 2020," Federal Bureau of Investigation, n.d., https://www.ic3.gov/Media/PDF/AnnualReport/2020_IC3Report.pdf.

[3] Anti-Phishing Working Group, "Phishing Activity Trends Report: 4th Quarter 2020," 9 February 2021, https://docs.apwg.org/reports/apwg_trends_report_q4_2020.pdf.

[4] Anti-Phishing Working Group, "Phishing Activity Trends Report: 3rd Quarter 2020," 24 November 2020, https://docs.apwg.org/reports/apwg_trends_report_q3_2020.pdf.

[5] Anti-Phishing Working Group, "Phishing Activity Trends Report: 2nd Quarter 2020," 27 August 2020, https://docs.apwg.org/reports/apwg_trends_report_q2_2020.pdf.

[6] SonicWall, "Phishing in the Age of SaaS," n.d., http://dwcc.dataworld.com.hk/news/2021/q1/sonicwall/webpage/cloudappsecurity/phishing-in-the-age-of-saas.pdf.

[7] G. Vrbancic, I.J. Fister, V. Podgorelec, "Parameter setting for deep neural networks using swarm intelligence on phishing websites classification," International Journal on Artificial Intelligence Tools 28.06 (2016): 1960008. DOI: https://www.worldscientific.com/doi/abs/10.1142/S0218213019600008X.

APPENDIX A
FEATURES BY TABLE

TABLE I
URL PROPERTIES

No.	*Attribute*	*Format*	*Value Description*
1	qty_dot_url	# of "." signs	Numeric
2	qty_hyphen_url	# of "-" signs	Numeric
3	qty_underline_url	# of "_" signs	Numeric
4	qty_slash_url	# of "/" signs	Numeric
5	qty_questionmark_url	# of "?" signs	Numeric
6	qty_equal_url	# of "=" signs	Numeric
7	qty_at_url	# of "@" signs	Numeric
8	qty_and_url	# of "&" signs	Numeric
9	qty_exclamation_url	# of "!" signs	Numeric
10	qty_space_url	# of " " signs	Numeric
11	qty_tilde_url	# of "~" signs	Numeric
12	qty_comma_url	# of "," signs	Numeric
13	qty_plus_url	# of "+" signs	Numeric
14	qty_asterisk_url	# of "*" signs	Numeric
15	qty_hashtag_url	# of "#" signs	Numeric
16	qty_dollar_url	# of "$" signs	Numeric
17	qty_percent_url	# of "%" signs	Numeric
18	qty_tld_url	Top level domain character length	Numeric
19	length_url	# of characters	Numeric
20	email_in_url	Is email present	Boolean [0, 1]

TABLE II
URL DOMAIN PROPERTIES

No.	*Attribute*	*Format*	*Value Description*
1	qty_dot_domain	# of "." signs	Numeric
2	qty_hyphen_domain	# of "-" signs	Numeric
3	qty_underline_domain	# of "_" signs	Numeric
4	qty_slash_domain	# of "/" signs	Numeric
5	qty_questionmark_domain	# of "?" signs	Numeric
6	qty_equal_domain	# of "=" signs	Numeric
7	qty_at_domain	# of "@" signs	Numeric
8	qty_and_domain	# of "&" signs	Numeric
9	qty_exclamation_domain	# of "!" signs	Numeric
10	qty_space_domain	# of " " signs	Numeric
11	qty_tilde_domain	# of "~" signs	Numeric
12	qty_comma_domain	# of "," signs	Numeric
13	qty_plus_domain	# of "+" signs	Numeric
14	qty_asterisk_domain	# of "*" signs	Numeric
15	qty_hashtag_domain	# of "#" signs	Numeric
16	qty_dollar_domain	# of "$" signs	Numeric
17	qty_percent_domain	# of "%" signs	Numeric
18	qty_vowels_domain	# of vowels	Numeric
19	domain_length	# of domain characters	Numeric
20	domain_in_ip	URL domain in IP address format	Boolean [0, 1]
21	server_client_domain	"server" or "client" in domain	Boolean [0, 1]

TABLE III
URL DIRECTORY PROPERTIES

No.	*Attribute*	*Format*	*Value Description*
1	qty_dot_directory	# of "." signs	Numeric
2	qty_hyphen_directory	# of "-" signs	Numeric
3	qty_underline_directory	# of "_" signs	Numeric
4	qty_slash_directory	# of "/" signs	Numeric
5	qty_questionmark_directory	# of "?" signs	Numeric
6	qty_equal_directory	# of "=" signs	Numeric
7	qty_at_directory	# of "@" signs	Numeric
8	qty_and_directory	# of "&" signs	Numeric
9	qty_exclamation_directory	# of "!" signs	Numeric
10	qty_space_directory	# of " " signs	Numeric
11	qty_tilde_directory	# of "~" signs	Numeric
12	qty_comma_directory	# of "," signs	Numeric
13	qty_plus_directory	# of "+" signs	Numeric
14	qty_asterisk_directory	# of "*" signs	Numeric
15	qty_hashtag_directory	# of "#" signs	Numeric
16	qty_dollar_directory	# of "$" signs	Numeric
17	qty_percent_directory	# of "%" signs	Numeric
18	directory_length	# of directory characters	Numeric

TABLE IV
URL FILE NAME PROPERTIES

No.	*Attribute*	*Format*	*Value Description*
1	qty_dot_file	# of ”.” signs	Numeric
2	qty_hyphen_file	# of ”-” signs	Numeric
3	qty_underline_file	# of ”_” signs	Numeric
4	qty_slash_file	# of ”/” signs	Numeric
5	qty_questionmark_file	# of ”?” signs	Numeric
6	qty_equal_file	# of ”=” signs	Numeric
7	qty_at_file	# of ”@” signs	Numeric
8	qty_and_file	# of ”&” signs	Numeric
9	qty_exclamation_file	# of ”!” signs	Numeric
10	qty_space_file	# of ” ” signs	Numeric
11	qty_tilde_file	# of ”˜” signs	Numeric
12	qty_comma_file	# of ”,” signs	Numeric
13	qty_plus_file	# of ”+” signs	Numeric
14	qty_asterisk_file	# of ”*” signs	Numeric
15	qty_hashtag_file	# of ”#” signs	Numeric
16	qty_dollar_file	# of ”$” signs	Numeric
17	qty_percent_file	# of ”%” signs	Numeric
18	file_length	# of file name characters	Numeric

TABLE V
URL PARAMETER PROPERTIES

No.	*Attribute*	*Format*	*Value Description*
1	qty_dot_params	# of ”.” signs	Numeric
2	qty_hyphen_params	# of ”-” signs	Numeric
3	qty_underline_params	# of ”_” signs	Numeric
4	qty_slash_params	# of ”/” signs	Numeric
5	qty_questionmark_params	# of ”?” signs	Numeric
6	qty_equal_params	# of ”=” signs	Numeric
7	qty_at_params	# of ”@” signs	Numeric
8	qty_and_params	# of ”&” signs	Numeric
9	qty_exclamation_params	# of ”!” signs	Numeric
10	qty_space_params	# of ” ” signs	Numeric
11	qty_tilde_params	# of ”˜” signs	Numeric
12	qty_comma_params	# of ”,” signs	Numeric
13	qty_plus_params	# of ”+” signs	Numeric
14	qty_asterisk_params	# of ”*” signs	Numeric
15	qty_hashtag_params	# of ”#” signs	Numeric
16	qty_dollar_params	# of ”$” signs	Numeric
17	qty_percent_params	# of ”%” signs	Numeric
18	params_length	# of parameters characters	Numeric
19	tld_present_params	TLD present in parameters	Boolean [0, 1]
20	qty_params	# of parameters	Numeric

TABLE VI
URL RESOLVING EXTERNAL SERVICES METRICS

No.	*Attribute*	*Format*	*Value Description*
1	time_response	Domain lookup time response	Numeric
2	domain_spf	Domain has SPF	Boolean [0, 1]
3	asn_ip	ASN	Numeric
4	time_domain_activation	Domain activation time (in days)	Numeric
5	time_domain_expiration	Domain expiration time (in days)	Numeric
6	qty_ip_resolved	# of resolved IPs	Numeric
7	qty_nameservers	# of resolved NS	Numeric
8	qty_mx_servers	# of MX servers	Numeric
9	ttl_hostname	Time-To-Live (TTL)	Numeric
10	tls_ssl_certificate	Has valid TLS /SSL certificate	Boolean [0, 1]
11	qty_redirects	# of redirects	Numeric
12	url_google_index	Is URL indexed on Google	Boolean [0, 1]
13	domain_google_index	Is domain indexed on Google	Boolean [0, 1]
14	url_shortened	Is URL shortened	Boolean
15	phishing	Is phishing website	Boolean [0, 1]