Detecting Malicious URLs using Predictive Modeling

Ana Truong
a8truong@ucsd.edu

Isa Vidanes
ividanes@ucsd.edu

Code: https://github.com/a8truong/dsc148_project

1 Dataset

1.1 Identify dataset

The dataset we used was the kaggle dataset “Tabular dataset ready for malicious url detection,” which has 6728848 rows and 60 columns. While there was both a train and test dataset available, we are only using the training dataset and working with a train-test split due to its sheer size. The dataset includes 6728848 URLs that are either benign or malicious, with the columns consisting of various features such as URL components, domain information, and SSL certificate details. All of these are features that can potentially aid in determining whether or not a URL is malicious. To begin our EDA, we will first take a closer look at the provided columns.

When we look at the data types of the columns, we can see that the majority of them are numerical- the only ones that are not are the url, source, and TLD, which are all components of the url itself. We can see that the unique values in the url column is equal to the number of rows in our dataframe, so we can omit this column when training our model since it essentially acts as an ID for the row. Of the numerical columns, 10 of them are binary columns relating to whether the url/path has a certain feature and if the TLD is suspicious. The rest are counts of various features of the URL, path, and domain, as well as the lengths of the URL, path, query, domain, TLD, and subdomain.

1.2 EDA

Taking a look at our observations, we can see that 5283175 are benign while 1445673 are malicious, meaning that we have a rough 4:1 benign to malicious ratio in our dataset. Since our classes are so imbalanced, we will have to account for this when it comes to selecting our model. Looking at the mean and standard deviations of the numerical features barring the boolean columns, separated by class, we can identify features of interest. First we will compare the means- some features of interest are: url length and counts of https, http, percentage, hyphen, www, atrate, hash, semicolon, underscore, question mark, equal, amp, letter, digit, sensitive financial words, sensitive words, path length, path counts of directory, embeds, zero, pertwent, upper, nonascii, query length, query components, pdomain length, pdomain counts of hyphen, atrate, non-alphanumeric, digit, as well as subdomain length and count of dots. In short, we will not be considering url entropy or hamming, url count of dots, number of unique characters ratio, path count lower, TLD length, and pdomain minimum distance. For our features of interest, we will compare the means and standard deviations to remove potentially problematic features. Features that may cause issues are: url length, url count digit, query length. These features have a high standard deviation compared to the mean, and will not be used when training our model.

Now we will explore our binary features: what percentage of True/False for each feature are labeled as benign or malicious? This will be important for determining which features we should train our model on. We can see that there are a handful of features where True

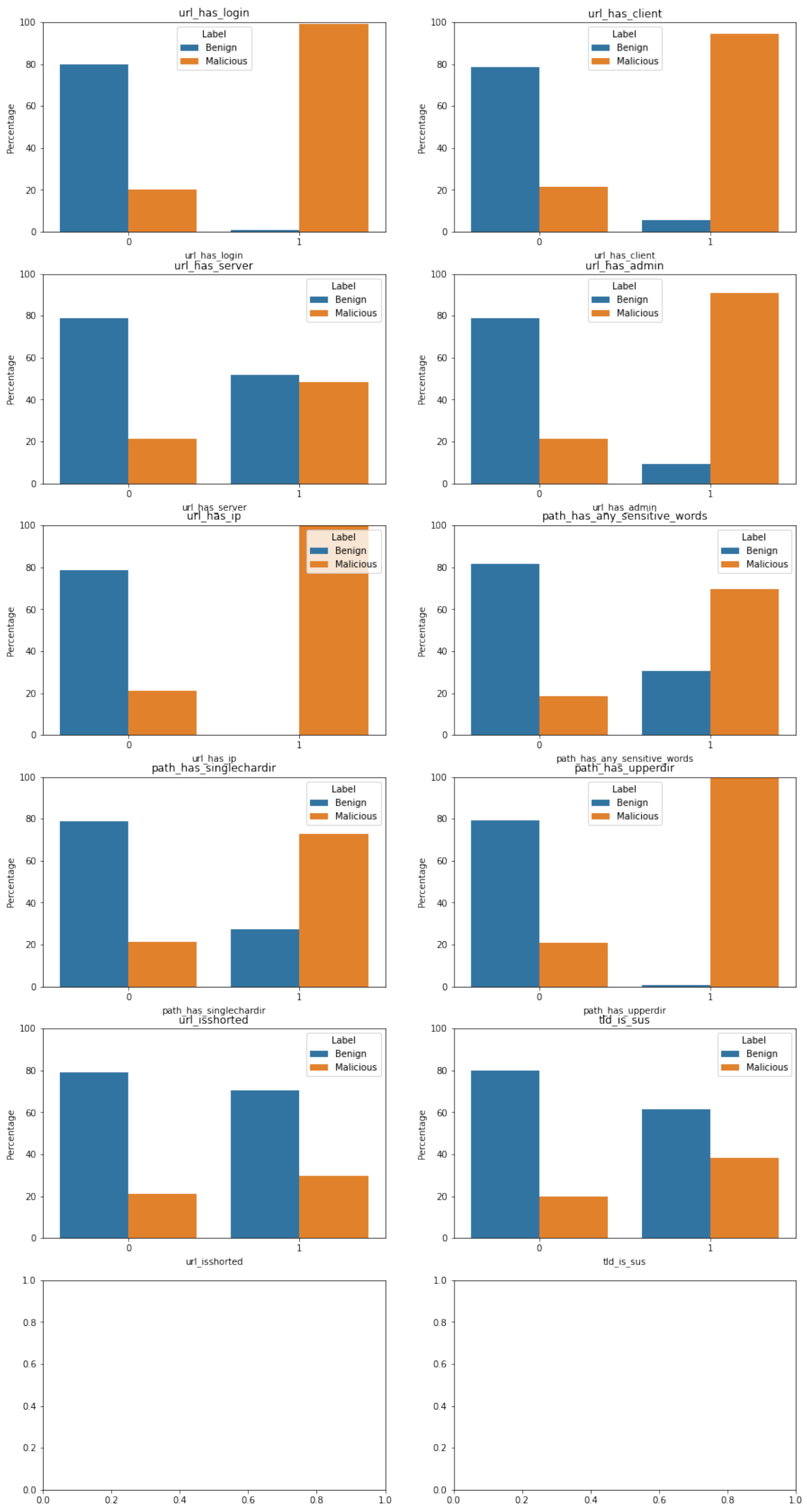
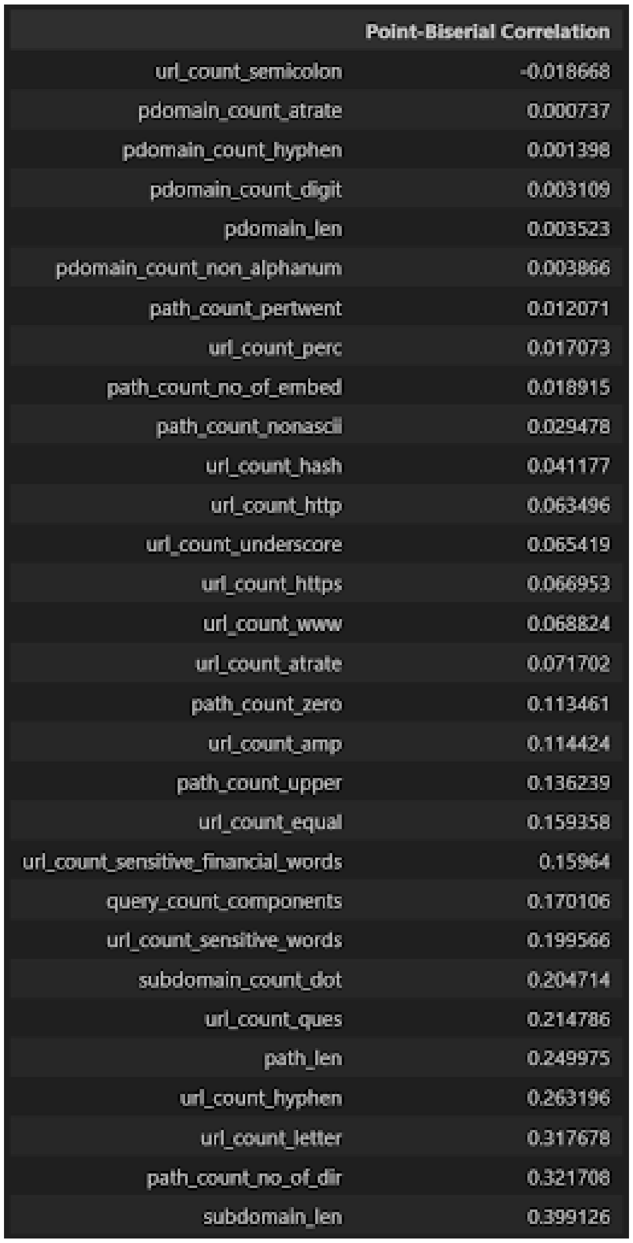


Figure 1: Percentage of features by label

has an overwhelming percentage of malicious urls: login, client, admin, ip, and upper directory. Sensitive words and single character directory may also be considered. This indicates that these features are more likely to be useful in predicting whether or not a url is malicious, while the others are not very significant and would be more likely to lead to our model overfitting to the data. Now we will take a look at our other numerical data, start-



	Point-Biserial Correlation
url_count_semicolon	-0.018668
pdomain_count_atrate	0.000737
pdomain_count_hyphen	0.001398
pdomain_count_digit	0.003109
pdomain_len	0.003523
pdomain_count_non_alphanum	0.003866
path_count_pertwent	0.012071
url_count_perc	0.017073
path_count_no_of_embed	0.018915
path_count_nonascii	0.029478
url_count_hash	0.041177
url_count_http	0.063496
url_count_underscore	0.065419
url_count_https	0.066953
url_count_www	0.068824
url_count_atrate	0.071702
path_count_zero	0.113461
url_count_amp	0.114424
path_count_upper	0.136239
url_count_equal	0.159358
url_count_sensitive_financial_words	0.15964
query_count_components	0.170106
url_count_sensitive_words	0.199566
subdomain_count_dot	0.204714
url_count_ques	0.214786
path_len	0.249975
url_count_hyphen	0.263196
url_count_letter	0.317678
path_count_no_of_dir	0.321708
subdomain_len	0.399126

Figure 2: Correlation coefficients

ing with our features that consist of counts. We will calculate the point-biserial correlation coefficient between the label and feature. We are using this instead of the more common Pearson's correlation coefficient because the point-biserial correlation coefficient is specifically designed for when one variable is dichotomous, which is the case for our label. We will be removing features that have little to no correlation to our label. After calculating the point-biserial correlation coefficients for our features of interest, we can see that those features are: pdomain_count_atrate, pdomain_count_hyphen, pdomain_count_digit, pdomain_len, pdomain_count_non_alphanum.

We also looked at the source column, plotting the percentage of the urls from each source that were malicious. As we can see below, the sources seem to strongly indicate whether or not a url will be malicious.

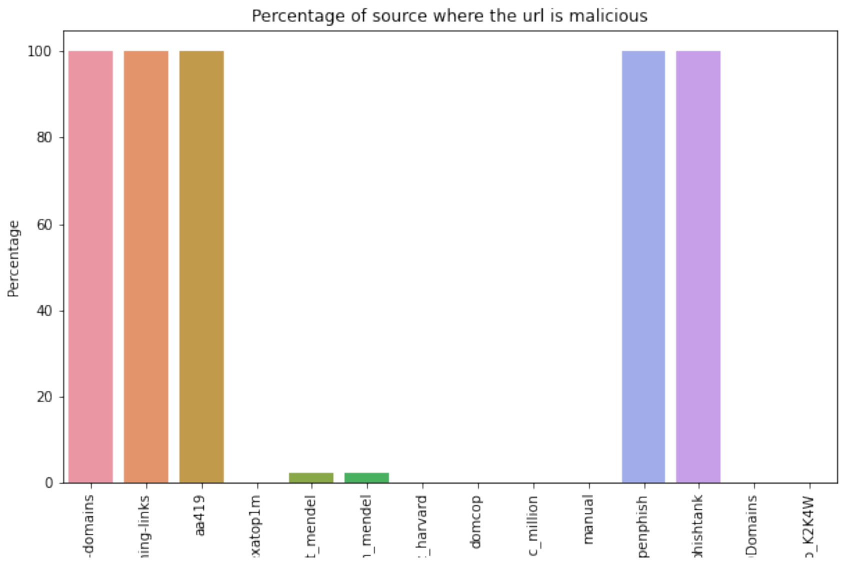


Figure 3: Percentages of malicious labels by source

2 Predictive Task

Our predictive task will be the classification problem of identifying whether or not a URL is malicious, as our dataset is extremely targeted towards that task. When considering how we will evaluate the different models for this task, we must first think about the consequences of false positives vs false negatives. False negatives would be costly as it would result in inviting malware into the device or stolen personal, while false positives, though not ideal, would not really matter and can easily be bypassed if necessary. Taking this into consideration, we will be evaluating our models on both recall and F1 score, since recall focuses more on reducing false negatives, while F1 score is a balance between false positives and false negatives. We would ideally reduce both, but want to focus more on reducing false negatives.

The baseline models we would want to compare with are other classification models, such as decision trees. A decision tree classifier would be an appropriate baseline due to it being the base of our random forest model. Since a random forest combines multiple decision trees to make its decision, a decision tree would work well with our data in a similar way a random forest would. We can tweak the hyperparameters of the decision tree (such as maximum decision splits) to avoid overfitting to the data.

While a decision tree model could work well with our dataset, we believe that implementing a random forest classifier would outperform this model since random forest works great on high dimensional data with non-linear relationships while avoiding overfitting. A random forest classifier would be more robust than a decision tree classifier since a random forest compares multiple decision tree results while the decision tree only has one result, and therefore is more likely to overfit to the data. Random forest classification does not have these limitations and is robust to noise in the data and can handle nonlinear decision boundaries. Alongside this, our classes are imbalanced with only roughly 20% of our dataset being malicious URLs. Other classification models, such as the SVM, does not perform as well for imbalanced datasets, while as for a random forest, we can set our class weight to be balanced in order to deal with this issue.

3 Model

We first ran 2 different baseline models: a decision tree and an SVM. The decision tree performed well and was able to fit the training data in an appropriate amount of time (39 seconds, to be exact.) The SVM classifier, on the other hand, was computationally too large and took an unreasonable amount of time to run. It ultimately never finished running, so we decided to scratch the SVM model and go with the decision tree as our baseline. To perform our predictive task, we decided to use a random forest classifier as our primary model. This algorithm works well with high-dimensional data (given that our dataset is large) and is effective at avoiding overfitting. The dataset came with calculated features ready to analyze and train from, allowing us to select certain features without having to create or design many. The features we did create, however, related to the URL source, which we discovered to have potential significant implications for classifying the data through our EDA. In order to use the URL source column (which contains strings), we one-hot encoded these features. After doing so, we filtered out any additional features we decided to use based on our EDA and did not make any alterations to the existing features that remained. After carrying out our feature selections, we decided to optimize beforehand by running a k-folds cross validation to find out which hyperparameters generate a model producing the most ideal results. Going into our random forest classifier, the main deciding factor was the fact that RF is a collection of decision trees, and can scale well to massive datasets by training the trees in parallel to each other. These trees are, essentially, trained on subsets of the data, and then the resulting predictions are combined together. This method is not only effective at avoiding overfitting of the data, but it also is robust to any existing noise or outliers.

4 Literature

The task of malicious url detection is not one that is novel – with the widespread use of the internet in modern times, and new websites constantly showing up, it is extremely important to be able to distinguish between a legitimate website and one that would steal users' information or riddle their devices with malware. Many studies have been done on malicious url detection, with variety in the techniques and features used.

4.1 Classification of Malicious URLs Using Machine Learning

https://www.mdpi.com/1424-8220/23/18/7760 (Shayan Abad 2023)

This paper discusses the ramifications of inadequate cybersecurity and how detection algorithms are necessary for protecting sensitive user information. By conducting their own study, the researchers were able to create a classification algorithm for identifying malicious URLs through the use of SVMs, random forests, decision trees, k-nearest neighbors as well as Bayesian optimization. Their work proved the use of random forests to be very useful as these algorithms resulted in high precision, recall, and F1 scores. A combination of these algorithms and data pre-processing techniques facilitates an effective model performance in identifying unsafe URLs. Our work establishes similar baseline / primary models and employs ways to optimize our models, such as k-fold cross validation.

4.2 Detecting Malicious URLs Using Machine Learning Techniques: Review and Research Directions

https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9950508 (Malak Aljabri 2022)

This journal article provides an in-depth exploration of various techniques that are commonly used to discern safe URLs from malicious ones.The researchers not only cover what models have proven to be effective, but also some methods that come with limitations and thus need improvements or expanding on. One approach is to distinguish a correlation between a new URL and the signature of an existing URL labeled as malicious (dubbed as the heuristic method). This has resulted in failed attempts to protect users and websites on multiple occasions, yet it is the simplest approach and can identify malicious / benign URLs at the very least. Another approach is the use of machine learning and deep learning models. The researchers compared a numerous amount of models producing an accuracy below or higher than 90%. They determined that XGBoost, random forests, k-means, and decision trees as the best performing algorithms with an accuracy of 99% and above. However, they did note that XGBoost is rarely used in malicious URL detection due to it being sensitive to noise and hard to scale. Other algorithmic methods, such as batch normalization, neural networks, and deep belief networks resulted in a lower performance with accuracy scores below 90%. Overall, the ML methods that proved to be the most effective in this research paper further corroborates our model choices and gives us a useful basis to refer to.

4.3 Malicious URL Detection based on Machine Learning

https://thesai.org/Publications/ViewPaper?Volume=11Issue=1Code=IJACSASerialNo=19 (Cho Do Xuan 2020)

Two machine learning algorithms were used: SVM and random forest. These two models were used with different parameters, producing different accuracy scores and overall results. For their main malicious URL detection algorithm, they utilized three types of features – lexical, host-based, and content-based. Lexical features included the length of the URL, main domain, maximum token domain, path average, and average token. The host-based features included source-related characteristics of the URL, such as the location and the identity of the servers. As for the content-based features, these included the information acquired after downloading an entire webpage. These features combined with the ML model selection of SVM and random forest present an effective detection system. The random forest outperformed the SVM with higher accuracy, precision, and recall scores. With more trees, the scores increased, whereas with more iterations for the SVM resulted in a lower score for certain metrics. The performance of the random forest classifier in this study further assists the feature and model selection we will proceed with for our project.

5 Results

Before selecting and creating our features, we carried out an EDA and discovered some significant characteristics about the data worth noting when choosing which features and which model to work with. There was a severe class imbalance between the benign and malicious URLs present in the datasets. This played a role in our model selection and how we went about choosing our evaluation metrics as well as hyperparameters. Comparing the averages and standard deviations for each column helped us distinguish which columns would presumably be useful or unuseful as well.

There was not much of a need for data cleaning or pre-processing, as the datasets provided were “clean” already and did not contain any missing values. When selecting our features, it is important to note that we omitted the features that had low point-biserial correlation. This type of correlation measures the relationship between the continuous numerical variables with the binary class label. Having a low correlation between these two values indicates that there is little to no linear relationship between the continuous and binary variables. We created one-hot encoded features that differentiated each unique variable in the “source” column into separate columns, which helped establish a relationship between the source of the URL and its class.

The features we have selected are effective in training the classifier to correctly identify whether a URL is malicious or not. With the features we have created and selected, our prediction models result in sufficiently high scores across the evaluation metric board.

We ran a 5-fold cross validation on our random forest classifier using both the training and test datasets. The scores were consistent across the 5 groups in both the training and test data.

The cross-validation scores using the training data:
[0.99629283 0.99623784 0.99614942 0.99622892 0.99627351]
The cross-validation scores using the test data:
[0.99613902, 0.99608849, 0.99621333, 0.99618656, 0.99620143]
Both are consistent and have very small differences between each fold, suggesting that the random forest is robust and can generalize well across various groupings of the data. The consistent high values also suggest the overall high performance of the model and how it can achieve a similar effectiveness when working with new and unseen data.

For our baseline model, we generated a decision tree classifier using the "best" split parameter to choose the most ideal split and the "entropy" function to account for the class imbalance found in our dataset. After running the predictions, this model produced an accuracy score of 99.6%, a recall score of 98.3%, and an F1 score of 99.1%. Our primary model, the random forest classifier, produced similar scores with an accuracy score of 99.6%, a recall score of 98.2%, and an F1 score of 99.1%. We began with 100 trees set for our "n_estimators" parameter and found that this achieved relatively high scores all around. We computed all 3 scores (accuracy, recall, and F1) in order to account for overfitting and the class imbalance found in the dataset.

We were hoping to minimize the false negatives and were able to achieve this goal, which can be seen through the recall scores for both the baseline and primary model. A recall score of 98% for both models indicates that there is a low rate of false negatives present. We were also able to effectively capture positive instances and minimize false positives by achieving an F1 score of 99% for both models.

5.1 Conclusion

Our random forest and decision tree classifiers both performed similarly well. The information provided in the dataset was very informative and illustrates clear patterns for our models to train and predict from. As a result, we can infer that the dataset is relatively straightforward and not as complex as we expected it to be, such that both of our models were able to capture the underlying relationship in the data pretty well. To further investigate any potential areas of misclassification, we also computed the confusion matrix, where we were able to see the existing false positives and negatives produced by our models. The false negative to true negative (as well as the false positive to true positive) ratio was significantly small – evident enough to verify the effectiveness and accuracy of our models.

References

Cho Do Xuan, Tisenko Victor Nikolaevich, Hoa Dinh Nguyen. 2020. "Malicious URL Detection based on Machine Learning." [Link]

Malak Aljabri, Shahd Albelali Maimunah Al-harbi Haya Alhuraib Najd Alotaibi Amal

Alahmadi Fahd Alhaidari Rami Mustafa Mohammad Khaled Salah, Hanan Altamimi. 2022. “Detecting Malicious URLs Using Machine Learning Techniques: Review and Research Directions.” [Link]

Shayan Abad, Mohammad Aslani, Hassan Gholamy. 2023. “Classification of Malicious URLs Using Machine Learning.” [Link]