Phishing URL Detection

Paul Lestz | Brown University ‘25 | 12/15/24

https://github.com/plestz/phishing_url_detection

I. Introduction

Phishing URL scams are becoming increasingly prevalent in the Information Age. Since 2019, unique phishing web attacks (via URLs) have exponentially eclipsed that of unique phishing email attacks (Fig. 1). Those who fall victim to phishing scams are at risk of irreversible loss of money, information, and time to malicious agents. Thus, the objective of an effective countermeasure is clear: create a machine learning model that can effectively distinguish between legitimate and phishing URLs, and flag the latter.

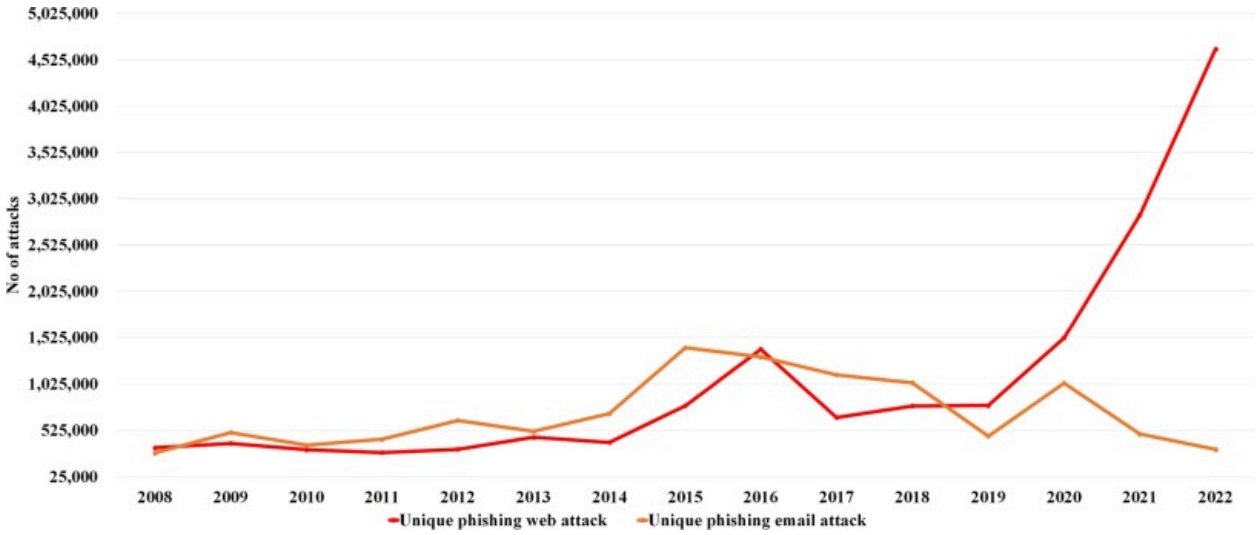

Figure 1: Exponential increase in phishing URL attacks in recent years, relative to email attacks (Prasad and Chandra).

The dataset for this problem is *PhiUSIIL Phishing URL (Website)* [2], which comprises 134,850 legitimate and 100,945 phishing URLs. With no missing values and 53 columns relevant to prediction, the dataset contains an unprecedented percentage of the positive (typically ‘rare’) class.

While the dataset utilized is the same as Prasad and Chandra, the use of an extremely limited feature set (see II. EDA) makes this a unique problem.

II. EDA

This is a binary classification problem: either a URL is legitimate (Class 0) or phishing (Class 1). The ratio of labels is ~57:43 – a fairly balanced dataset.

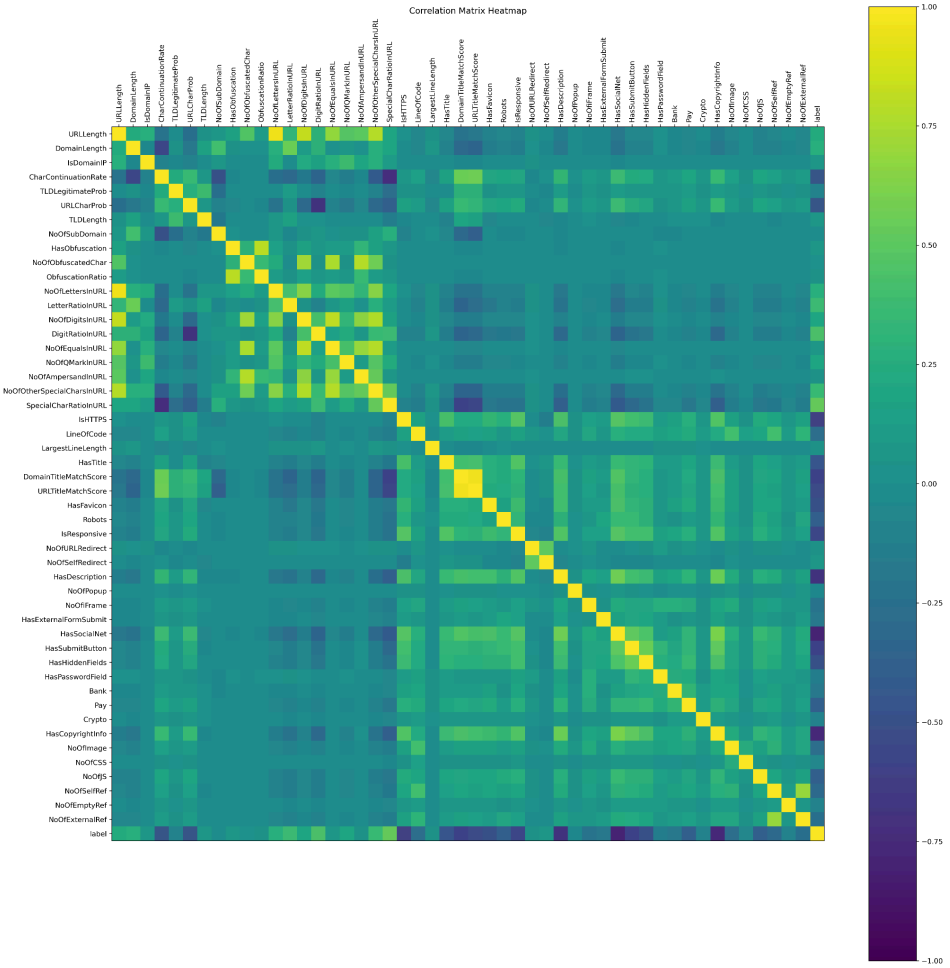

Figure 2: The Pearson correlation pairwise heatmap of all features. More 'predictive' features are those with a higher magnitude correlation with 'label'.

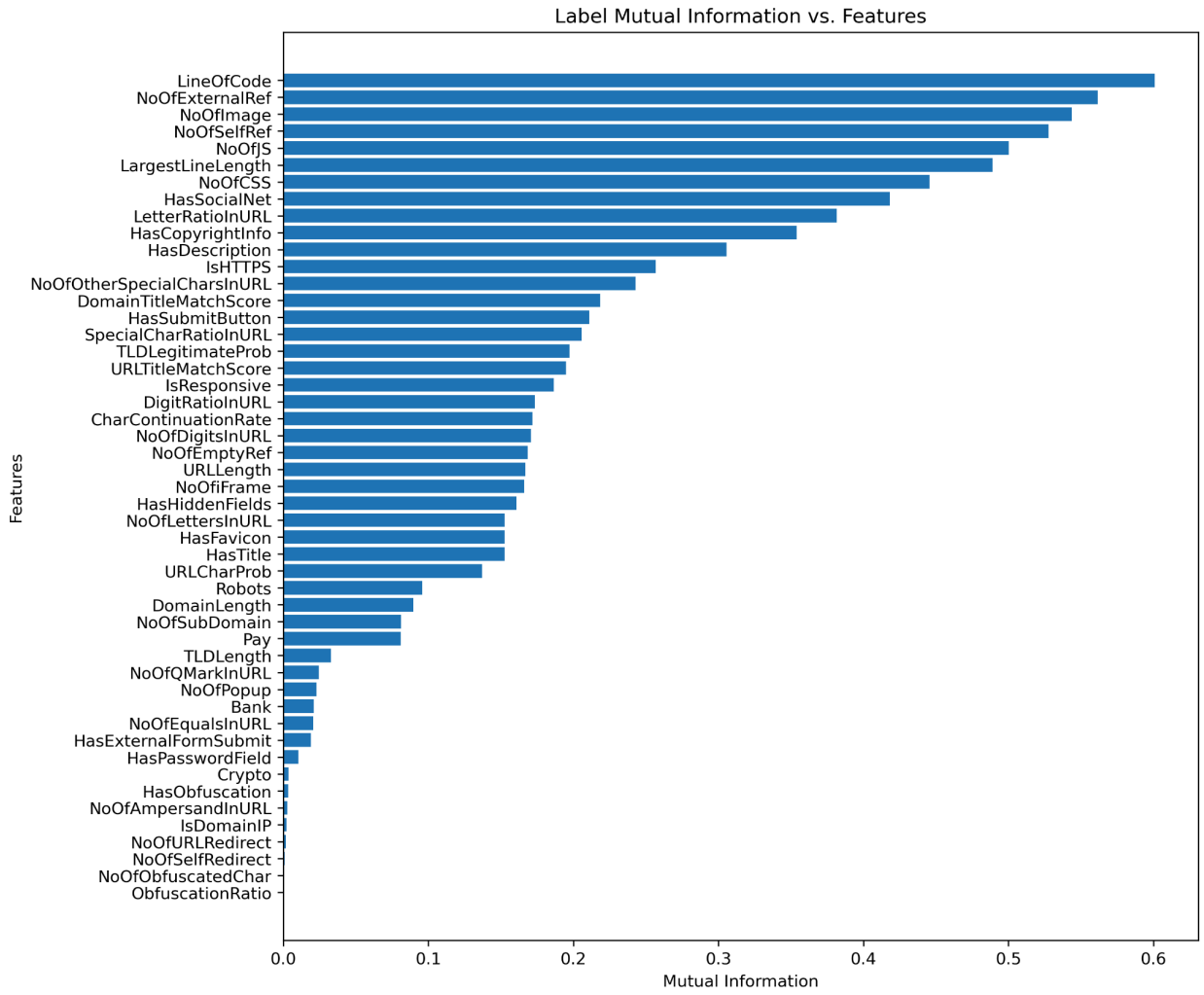

Figure 3: The mutual information horizontal bar plot of all features' relationship magnitude with 'label'. Thus, more 'predictive' features have longer bars.

From the 53 potential predictors, I selected eight from among the top correlations with the label, both linear (via Pearson correlation) and nonlinear (via mutual information) (Figs. 2, 3). Specifically, the features are LineOfCode, NoOfExternalRef, NoOfImage, NoOfSelfRef, NoOfJS, LargestLineLength, LetterRatioInURL, and HasSocialNet.

Each feature provides unique insight into distinguishing between whether a URL is likely to be of a legitimate or phishing variety. For example, legitimate URLs tend to have a narrower (and lower on average) range of LetterRatioInURL values compared to phishing URLs (Fig. 4).

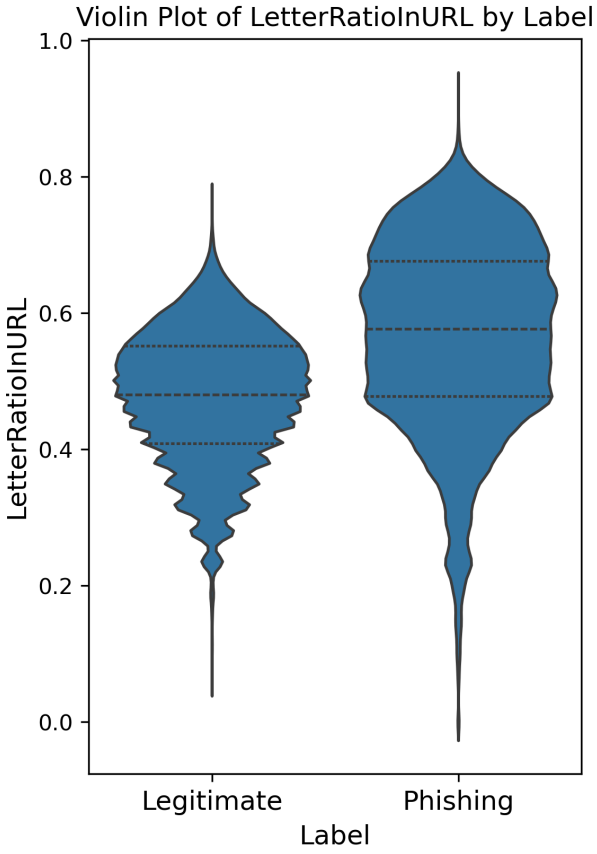

Figure 4: A violin plot containing a distribution of LetterRatioInURL values for instances with each of the Legitimate and Phishing true labels.

With eight of the most predictive features, we have sufficiently reduced our feature space to allow our models to capture most of the relationship between them and the ground truth.

III. Methods

False positives and false negatives both present issues for a phishing URL detector. If the model misclassifies a URL as a false positive, then it has flagged a legitimate URL as phishing; while this might require an individual to override the phishing flag (which presents a mild time burden), there is little risk of loss. If a model misclassifies a URL as a false negative, then it has flagged a phishing URL as legitimate; the ramifications of doing so are severe: an individual is at high risk of losing information, time, or money. Given the relative ramifications, our model should prioritize avoiding false negatives (higher recall) over false positives

(higher precision), while still accounting for both. Thus, the F2 score (emphasizing recall) is a clear performance evaluation metric for our models.

To ensure scale consistency among features, the preprocessing step for every model includes a MinMaxScaler (for LetterRatioInURL), followed by a StandardScaler for all features. This is important for the proper optimization of the non-tree based models (i.e. Logistic Regression's gradient descent, and Support Vector Classifier's distance-based approach).

Because the dataset is slightly imbalanced, stratification of all datasets (by label) is valuable for ensuring proper representation. Given I.I.D. instances, KFold cross-validation will be used to effectively measure and compare model performance across hyperparameter combinations to find the best estimator. From the original dataset, 20% will go to a stratified test set, while the remaining 80% will be used for cross-validation. For each of the four models in this project (Logistic Regression [LogR], Random Forest Classifier [RFC], Support Vector Classifier [SVC], and XGBoost Classifier [XGBC]), five seeds-worth of cross-validation procedures were completed, to obtain five best estimators, and thus five test scores, for each model. These test scores are used to examine the mean and standard deviation of model performance across different splits and non-deterministic optimization conditions (for tree-based methods) in the Results section. StratifiedKFold will be used to split the cross-validation set, with GridSearchCV being used for cross-validation itself (implemented manually for XGBoost Classifier, to take advantage of early stopping and a dynamic evaluation set).

For (linear) Logistic Regression, consistent parameters were solver = 'saga', max_iter = 1000, n_jobs = -1, and tol = 0.005; its tuned hyperparameters were 'C' (the regularization parameter), for values of [1e-3, 1e-2, 1e-1, 1, 1e1, 1e2, ***1e3***], and 'penalty' (the regularization type), for values of ['l1', **'*l2*'**]. The bolded values produced the model with the highest cross-validation score; the same pattern applies to the below.

For the (nonlinear) Random Forest Classifier, consistent parameters were n_estimators = 100 and n_jobs = -1; its tuned hyperparameters were 'max_depth' (the maximum tree depth of an estimator), for values of [3, 5, *7*], and 'max_features' (the maximum number of features considered per split), for values of [***2***, 3, 5, 8].

For the (nonlinear) Support Vector Classifier (RBF kernel), consistent parameters were max_iter = 10000 and tol = 1; its tuned hyperparameters were 'C' (the regularization parameter), for values of [0.1, 1, ***10***], and 'gamma' (the kernel coefficient), for values of [***0.1***, 1, 10].

For the (nonlinear) XGBoost Classifier, consistent hyperparameters were early_stopping_rounds = 10, n_jobs = -1, colsample_bytree = 0.9, subsample = 0.66, learning_rate = 0.03, and n_estimators = 10000; its tuned hyperparameters were 'max_depth' (the maximum tree depth of an estimator), for values of [3, ***5***, 8], 'reg_alpha' (L1 regularization strength), for values of [0.01, ***0.1***, 1], and 'reg_lambda' (L2 regularization strength), for values of [***0.01***, 0.1, 1].

IV. Results

Of the four models tested, XGBoost is the most predictive (Fig. 5). SVC and RFC follow close behind, with Logistic Regression lagging relatively far behind in performance. Note that all models have F2 scores above 0.975, indicating high overall performance.

The baseline F2 score (defined only when predicting the minority class) is 0.789157 (Fig. 6); the stratified nature of all splits ensures a standard deviation of 0. Thus, clearly the models well outperform the baseline score. Given the XGBoost Classifier mean score of 0.997236 and standard deviation of 0.000171, the difference between the two means are 0.997236 - 0.789157 = 0.208079. Thus, the best XGBoost Classifier is 0.208079 / 0.000171 = ~1216.84 standard deviations above the baseline score.

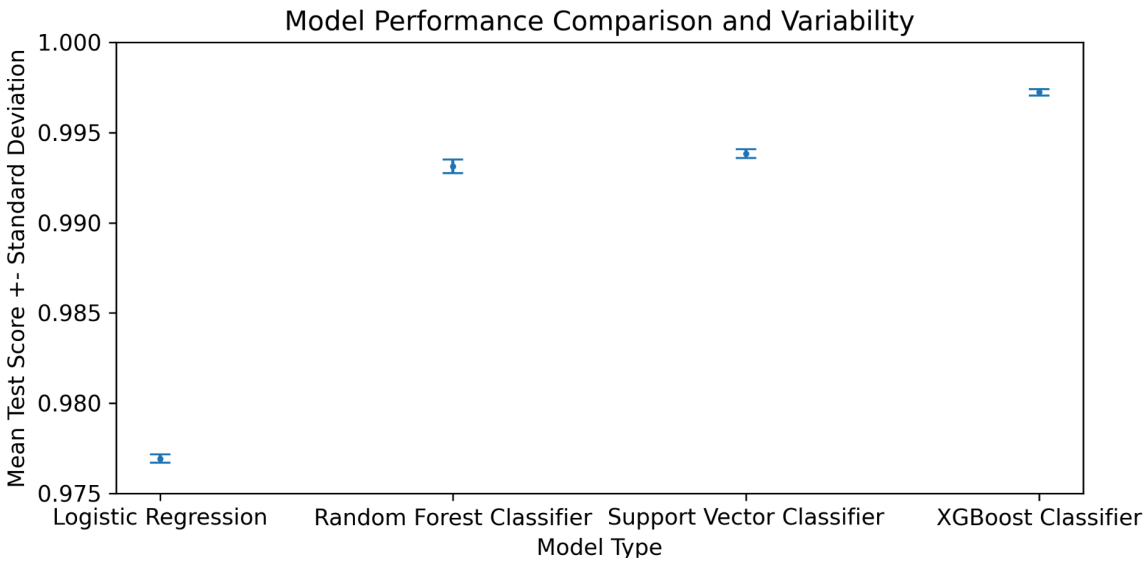

Figure 5: The mean test F2 score and standard deviation error plot for each of the four models.

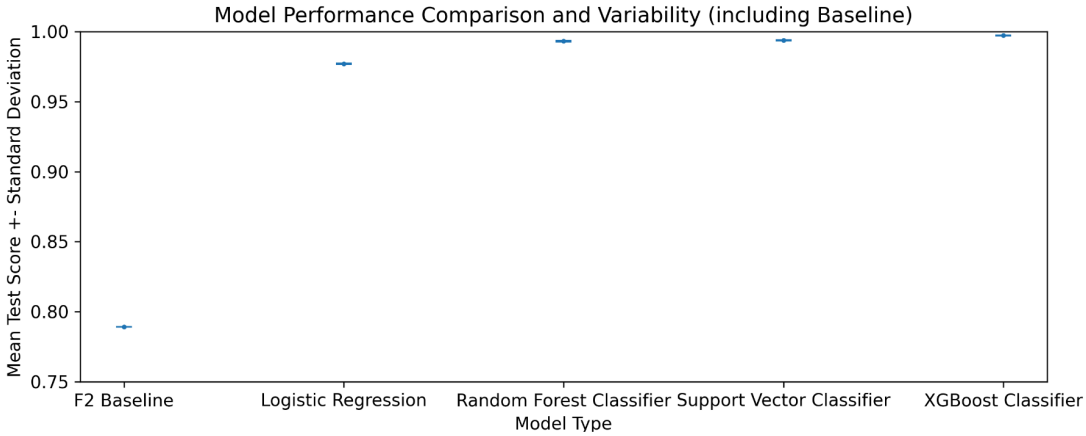

Figure 6: The same as Figure 5, also including the baseline F2 score.

Visually and numerically, XGBoost Classifier is the winner in terms of predictive performance for phishing URL detection. In fact, it is the dominant model: it wins in terms of both mean and standard deviation of F2 score performance. With a mean F2 score outperforming the next-best model (SVC) by 0.003399 and a standard deviation outperforming the next-best model (LogR) by 0.000057, the verdict is clear (Fig. 7).

	F2 Baseline	LogR	RFC	SVC	XGBC
Seed 1	0.789157	0.977156	0.992907	0.993884	0.997345
Seed 2	0.789157	0.977161	0.992775	0.993985	0.997196
Seed 3	0.789157	0.976719	0.993092	0.994020	0.997473
Seed 4	0.789157	0.976736	0.993106	0.993892	0.997097
Seed 5	0.789157	0.976854	0.993779	0.993405	0.997076
Mean	0.789157	0.976931	0.993132	0.993837	0.997236
StDev	0	0.000228	0.000387	0.000248	0.000171

Figure 7: The full layout (per seed) of model (and baseline) F2 test score performance, including means and standard deviations.

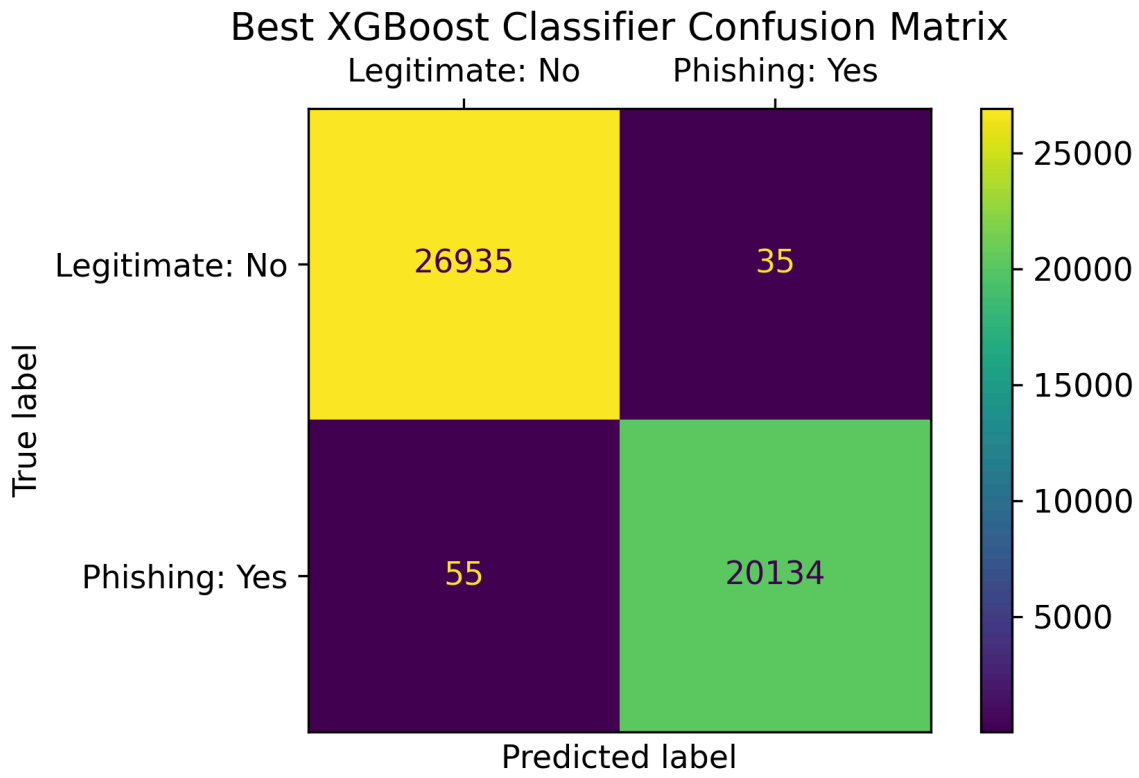

Figure 8: The confusion matrix of the best XGBoost Classifier estimator. Few False Positives and False Negatives are present.

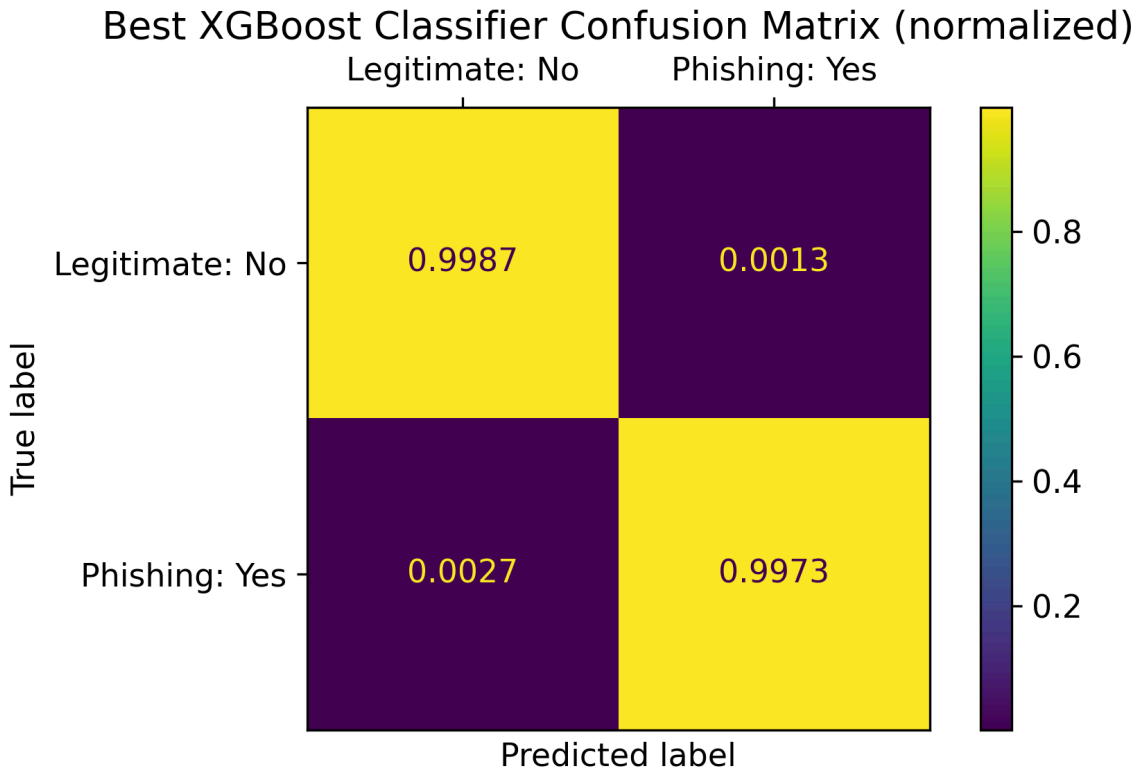

Figure 9: The row-wise normalized version of Figure 8.

When examining the confusion matrix for the best XGBoost Classifier, it is apparent that nearly all instances have been properly classified as True Positives or True Negatives. Only 90 instances in the test set were misclassified (35 as False Positives, 55 as False Negatives) (Fig. 8). Given the sheer quantity of examples, only 90 misclassifications is a marker of success, given its relatively low percentage of the over 40,000 test examples (0.13% of True Legitimate URLs were misclassified as False Positives, and 0.27% of True Phishing URLs were misclassified as False Negatives) (Fig. 9).

While Pearson correlation and mutual information were potential indicators of feature importance, Global Feature Importance techniques can be run on the trained models and test sets to gain a more robust ranking of features' predictive values.

Utilizing Permutation Feature Importance (for all features post-preprocessing) (Fig. 10), it is clear that LineOfCode is significantly more valuable than NoOfSelfRef, which is more valuable than LargestLineLength, and

so on. This is evidenced by how much the test performance is hurt by shuffling these features in the test set.

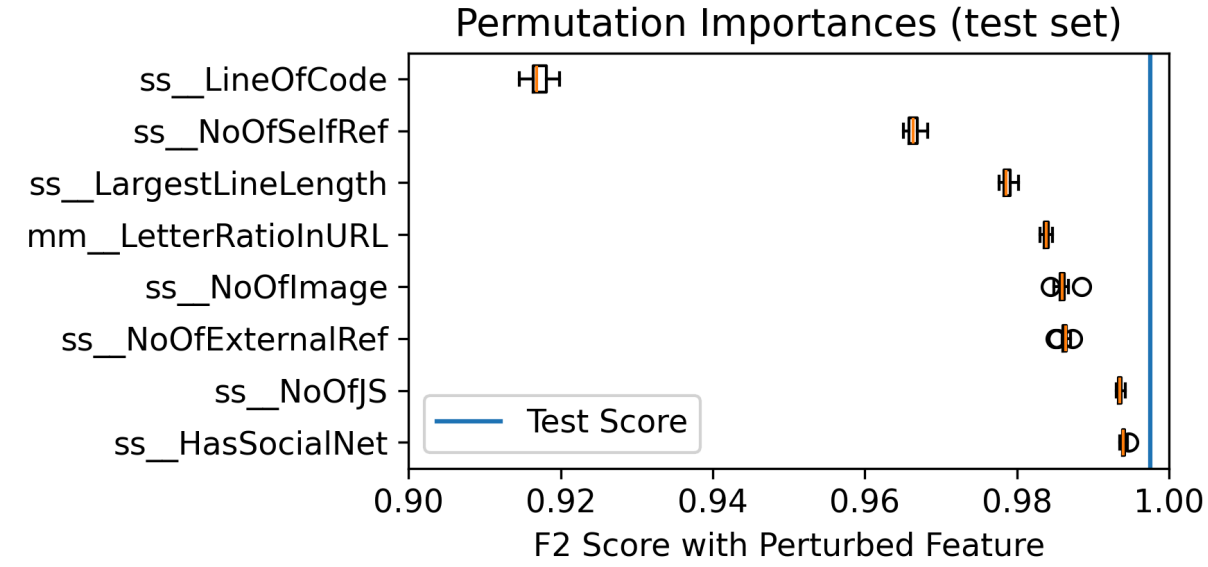

Figure 10: The Permutation Feature Importance of each feature, visualized as the detriment in test score by shuffling that feature for many seeds.

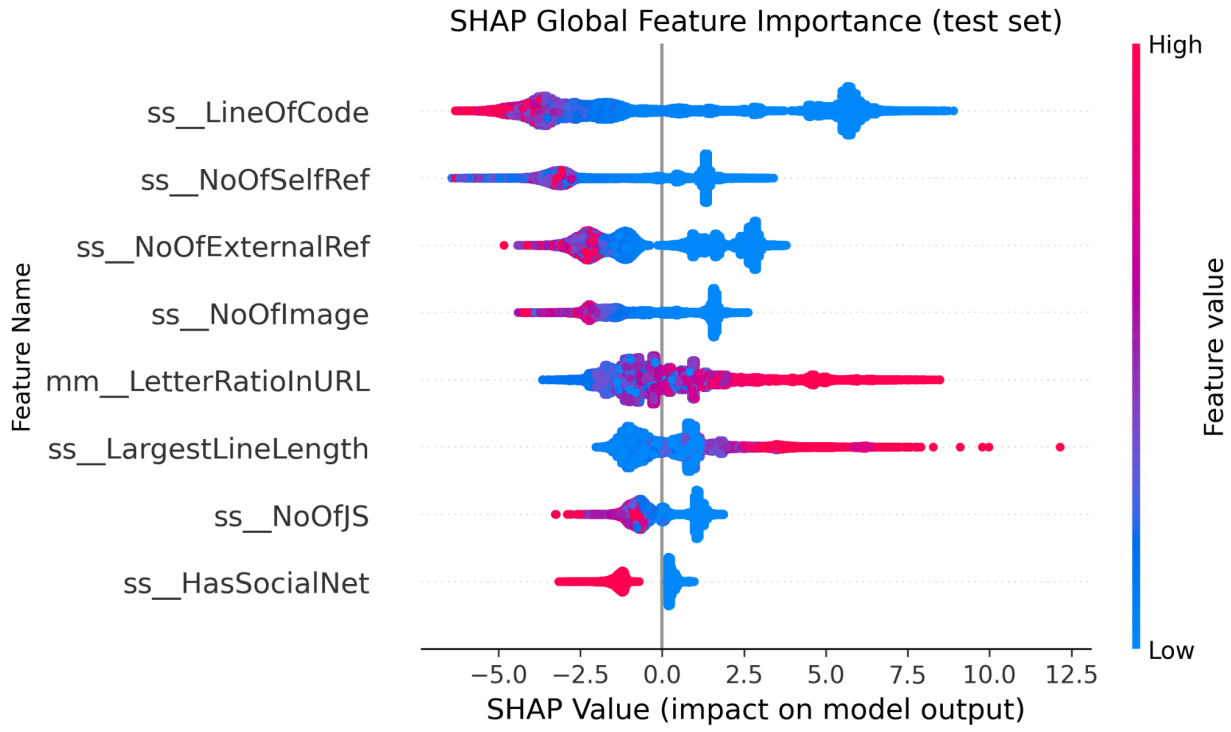

Figure 11: The SHAP Summary Plot representing the distribution of SHAP values for each feature.

These valuable features are further confirmed in the SHAP Summary Plot (Fig. 11) and corresponding bar plot (Fig. 12), which also value LineOfCode and NoOfSelfRef in the highest slots due to average magnitude of impact and assistance in accurately predicting the label for each instance.

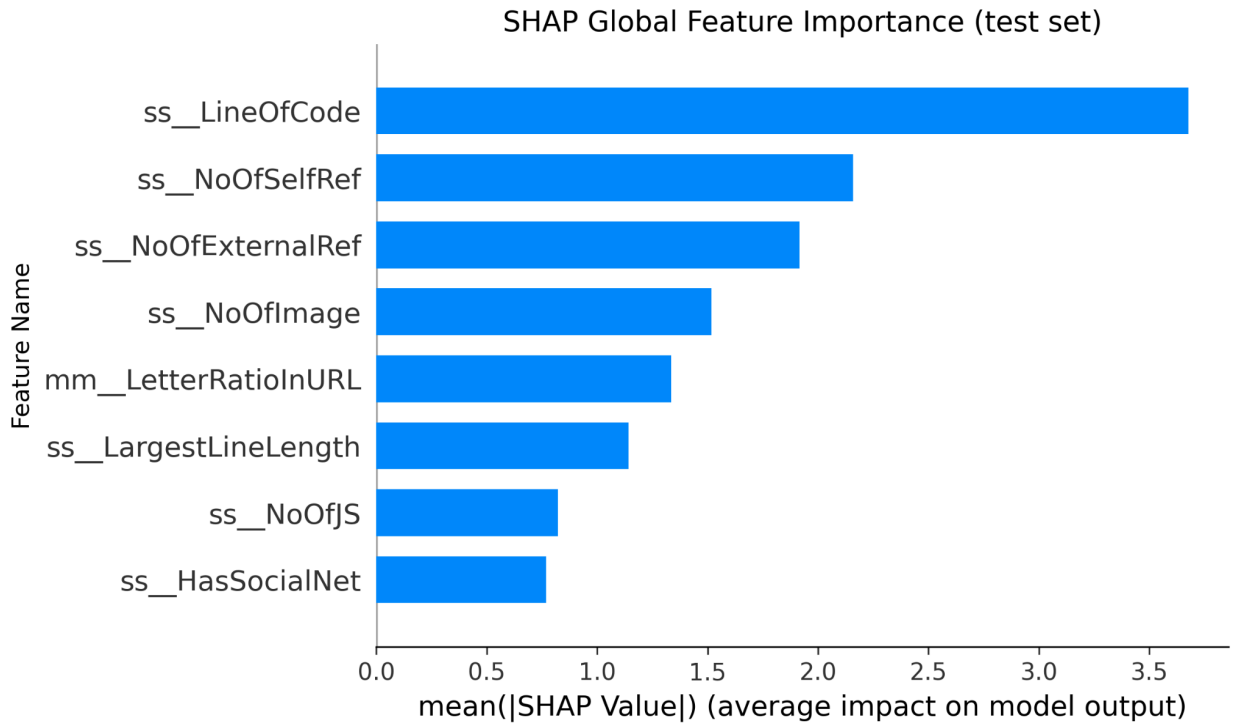

Figure 12: The SHAP Summary Bar Plot, representing the mean SHAP value impact of each feature. The larger the bar, the more 'predictive' the feature.

The use of XGBoost Classifier's global importance metric of total_cover (the aggregate [across all trees] of points that are impacted by each feature) again confirms the value of LineOfCode and NoOfSelfRef (with the other features also ranking themselves in similar orders) (Fig. 13).

Intuitively, the ranking of the features is sound. For example, the number of lines of code on a website (represented by LineOfCode), and thus the website's potential complexity, can be a valuable indicator of its legitimacy. A phishing website with only a couple lines of code attempting to masquerade as a known-to-be complex other website's URL is a clear mismatch.

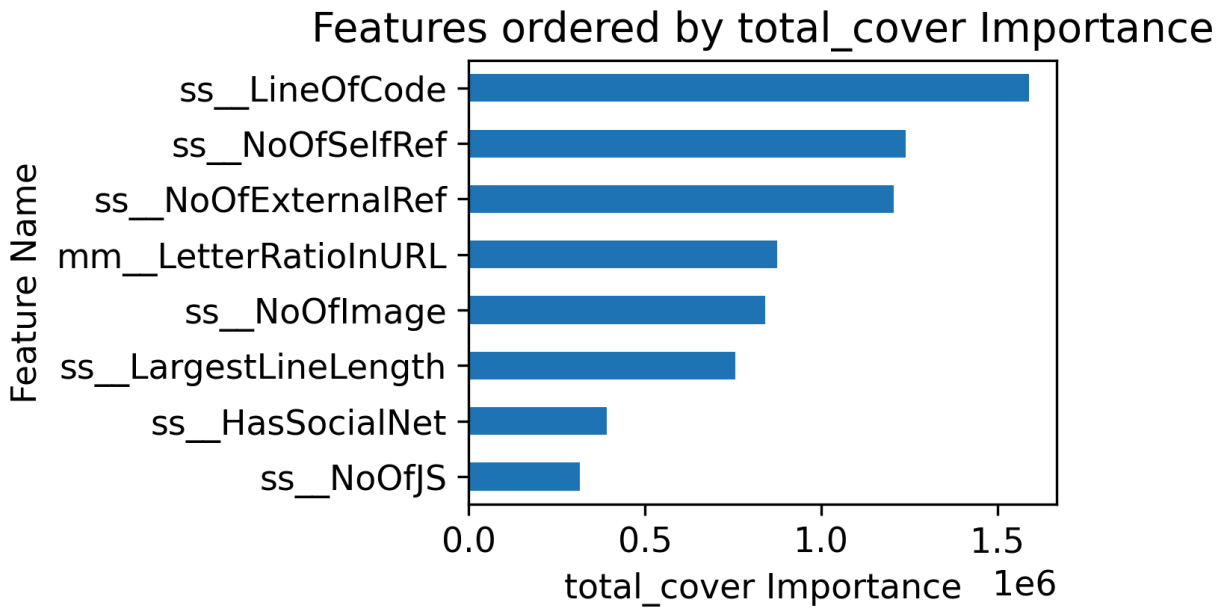

Figure 13: The XGBoost Global Feature Importance metric of total_cover for each feature.

Using SHAP local feature importance metrics, we can visualize the impact of each of the eight features on predicting whether an instance is more likely to be Class 0 (legitimate, leftwards, blue arrows) or Class 1 (phishing, rightwards, red arrows). The force plots are indicating raw logits (that can be transformed to probabilities with the sigmoid function); note the base value of roughly 0, indicating the initial roughly 50/50 odds for an example to be one class or the other (with no feature influence).

In Figure 14, sample 3 has slightly more blue contributors than red, producing a raw logit score of -2.03; in other words, this example is probably Class 0, but might be Class 1. Notably, NoOfSelfRef is visualized as the largest contributor towards Class 1. Sample 131 has many more red contributors than blue, making it more than likely to have a Class 1 ground truth. Samples 41 and 76 are even more extreme, representing (with near predictive certainty) a Class 1 and Class 0 instance, respectively. LineOfCode is within the top two feature contributors for the latter three examples.

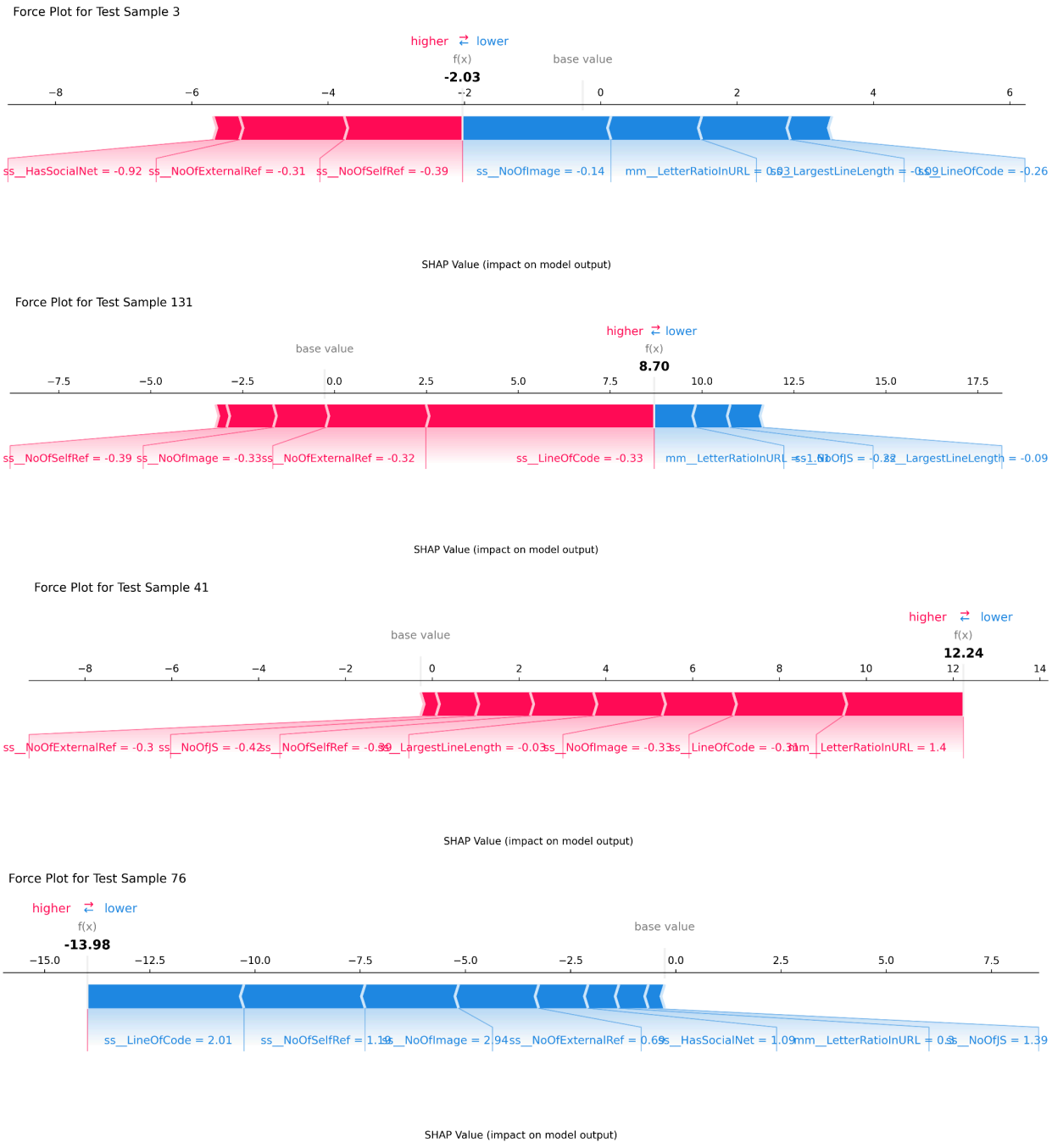

Figure 14: A series of SHAP local feature importance force plots, representing each feature's directional classification impact on each instance.

V. Outlook

Our models excel in performance. If one was looking to further increase predictive power, then any of the following could be employed: (1) use more than 8 of the 53 existing features, (2) search over more hyperparameter combinations, (3) utilize feature engineering (e.g. PolynomialFeatures), (4) use other model architectures (e.g. neural networks), and (5) if possible, collect more training data (e.g. new rows, or new features including more information about the website).

Our models are also highly interpretable. If one was looking for sources of additional interpretability, then they could (1) take advantage of other local explanatory tools (e.g. Local Interpretable Model-Agnostic Explanations [LIME]), and (2) create confusion matrices for filtered subsets of the data (e.g. look at misclassifications for only URLs with a LetterRatioInURL >= 0.5).

Clearly, modern machine learning technologies effectively enable us to counter the latest spike in unique phishing web attacks.

VI. References

[1] Prasad, Arvind and Shalini Chandra, PhiUSIIL: A diverse security profile empowered phishing URL detection framework based on similarity index and incremental learning, Computers & Security, Volume 136, 2024, 103545, ISSN 0167-4048, https://doi.org/10.1016/j.cose.2023.103545. https://www.sciencedirect.com/science/article/pii/S0167404823004558.

[2] "PhiUSIIL Phishing URL (Website)." *UCI Machine Learning Repository*, archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset.

[3] Prasad, Arvind. "PhiUSIIL Phishing URL Dataset." *Kaggle*, 8 Mar. 2024, www.kaggle.com/datasets/ndarvind/phiusiil-phishing-url-dataset.