Phishing Link Identifier

A Machine Learning Approach to Detecting Malicious URLs

Amelia Farrell

DSC680-T302 Applied Data Science

3/17/25

Appendix

Phishing Link Identifier

A Machine Learning Approach to Detecting Malicious URLs

Business Problem

Phishing is the practice of sending messages (emails, texts, etc.) with the purpose of tricking individuals into revealing personal or company information, such as passwords, credit card numbers, and other confidential information. There are four types of phishing, including Smishing (SMS Phishing), Vishing (Voice Phishing), Angler Phishing (Social Media Phishing), and Email Phishing (the most common type). Many phishing attempts succeed by incorporating links that trick users into revealing sensitive information. This is done by redirecting users to fake login pages, (e.g. companyname.com versus companyname-secure.net, hiding the true destination of the link, auto-downloading malware, or exploiting browser vulnerabilities, such as stealing session cookies. It can be difficult for the average user to detect phishing links, as they may look legitimate at a quick glance. These malicious links pose a significant threat to individuals and businesses. If the user engages it can lead to financial losses, data breaches, reputational damage, and operational disruptions.

Background/History

Experi-Metal Inc. (EMI) lost around $1.9 million (with ~$560K recovered) due to a phishing email that contained a fake website (designed to look like a bank's website) (E.D. Mich., 2011)1. An employee trusted the email and site, entering confidential login credentials. This "login" gave the attackers the data they needed to inadvertently access to EMI's bank accounts and transfer the $1.9 million. Another example is the IRS Phone Scam that spanned over four years (2012–2016). Scammers posed as the Internal Revenue Service (IRS) threatening those who did not comply with arrest, imprisonment, and fines (Hauser, 2018)2. They succeeded in stealing hundreds of millions of dollars and personal information from 50,000 individuals over that time period. By creating a machine learning model that can automatically detect phishing links, we can help prevent emails and tests from ever reaching their target.

We are not the first to try to solve this problem. Prior studies have used machine learning models to detect malicious links. Tan and colleagues at Morgan State University proposed a web browser plug which used machine learning to auto detect malicious URLs, helping users avoid such sites. The team chose to train a random forest classifier on "11,000 data points with 30 [URL] features downloaded from phishtank" (Tan, K. H., & MacGregor, A., 2022). The 11,000 observations were classified as malicious or legitimate. After training the model with a 90/10 training split, they were able to achieve 96% accuracy. This was quite an improvement from the built-in web browser detection rate stated in the study (23.75% average accuracy across 4 different popular web browsers). While team's plug-in appears to be a success, it is not clear whether or not it was implemented or effective in real world use. As it was only tested on 10% of the dataset and no cross validation was done. We believe that we can achieve higher accuracy by identifying more features, testing multiple types of models, and cross validating.

Comparison Studies (Models)

Authors/year	Technique Used	Model Accuracy	Weaknesses
Tan, K. H., & MacGregor, A., (2022)3	Random Forest Classifier	96%	Small test set and no CV.
Zhang, H., Liu, G., Chow, T. W. S., Liu, W. (2011)4	Bayesian model (Naive Bayes)	>99%	Trained for 8 specific shopping websites. Will not apply to unrelated sites/links. Trained on text and visual content pull on website. Image detection models use more CPU to implement.
Chandan, C. J., Chheda, H. P., Gosar, D. M., Shah, H. R., & Bhave, P. U. (2014) 5	Neural Network	86%	Only 6 characteristics from phishing URLs were used to train the model.
Shah, B., Dharamshi, K., Patel, M., & Gaikwad, V. (2020)6	Random Forest Classifier	89.6%	16 characteristics were used in training. Model output one of three possibilities; Legitimate, -Suspicious, or Phishing. Leaving room for links that model we unsure of ("Suspicious").

Table 1. *Related research. Benchmark Comparisons for model performance.*

Some of the questions we will be exploring during our exploratory data analysis (EDA) will be, what are the most common characteristics of links marked as phishing? Are we seeing any patterns here? Are there any commonalities between phishing URLs and legitimate ones (potentially through our model)? What phishing URLs are marked as legitimate by our model? How can we improve further?

Data Explanation

Data Source

The dataset used to build and train our phishing link detection model was found on Kaggle (https://www.kaggle.com/datasets/hassaanmustafavi/phishing-urls-dataset?resource=download) It contains over 450,000 URLs, marked as 'legitimate' (77%) or 'phishing' (23%). It provides a robust foundation for training binary classification models. The lack of predefined features also gives us the opportunity to identify potential characteristics as well as build the functions needed to split new URLs into the identified features. This functionality will be needed for model implementation.

Data Preparation (Balancing)

Since our data had an unbalanced number of 'legitimate' (77%) and 'phishing' (23%) URLs (links), we needed to first balancing the dataset (50/50 'legitimate'/ 'phishing'). This is needed to avoid bias in the model towards the majority class. With our majority class being legitimate ULRs/links, this imbalance would likely lead to phishing links being incorrectly identified as legitimate.

We balance the data set by undersampling the majority class (legitimate' links). We chose undersampling as opposed to oversampling (SMOTE) due to the high number of data points we have available. Undersampling avoids the creation of synthetic samples typically done with SMOTE, that are not representative of the true data distribution and can lead to noisy or misleading models.

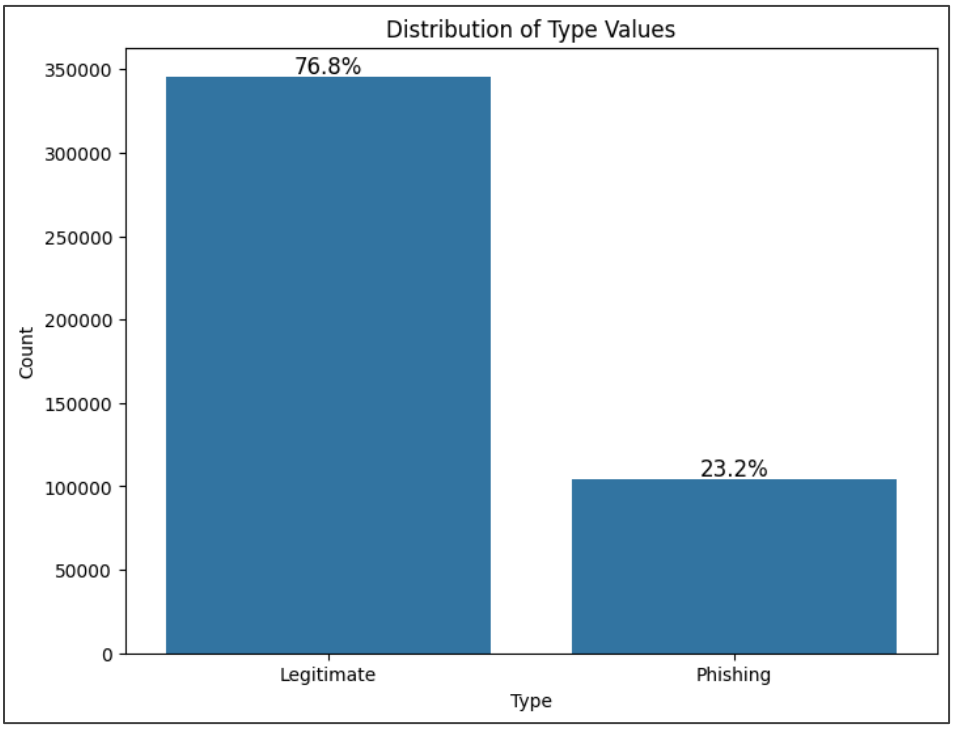

Figure 1. *Data split before undersampling.*

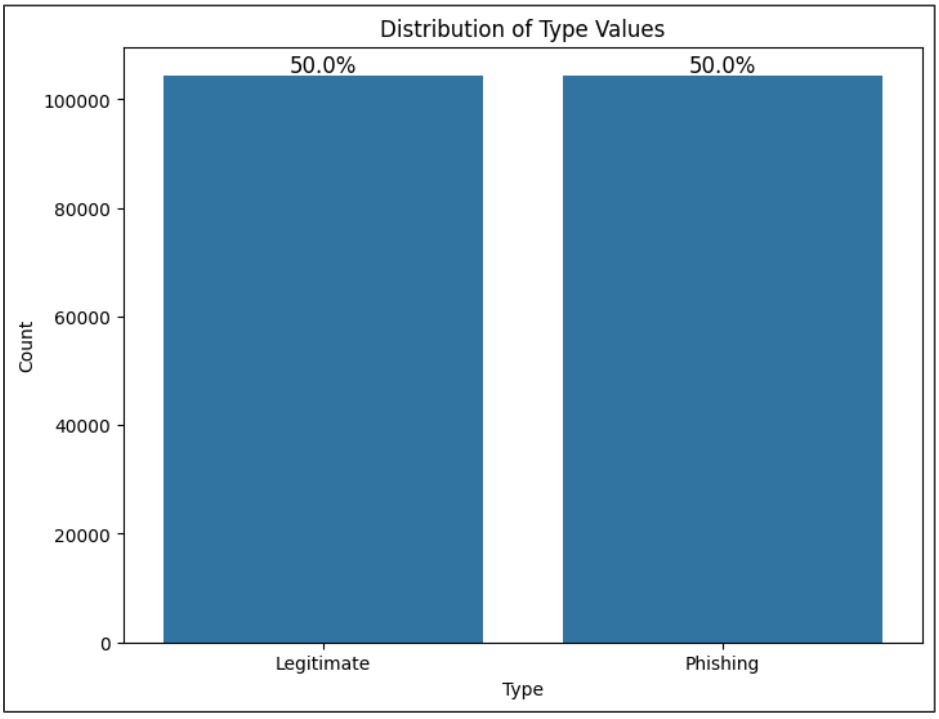

Figure 2. *Data split after undersampling.*

Data Preparation (Feature Extraction)

Next, we performed feature extraction (variable creation) from our remaining 200,000 + URLs. We engineered features based on visual inspection of known phishing links. We also added features that could be possible indicators based on domain knowledge.

Feature	Feature Type	Description	Why it was Chosen
num_numbers	Content	Counts the number of digits in the URL (e.g., 0 in https://www.google.com)	Phishing URLs often use digits for randomness or deception (e.g., login123), unlike typical legitimate URLs.
num_alpha_chars	Content	Counts alphabetic characters (e.g., 14 in https://www.google.com)	Helps quantify URL content; phishing URLs may have unusual alphabetic distributions.
repeated_chars	Content	Length of the longest sequence of repeated characters (e.g., 2 for oo in google)	Repeated characters can signal typosquatting (e.g., g00gle) or attempts to mimic legitimate domains.
has_suspicious_keywords	Content	True if the URL contains suspicious keywords (e.g., login, secure), False otherwise	Phishing URLs often include keywords to trick users into trusting the link, making this a strong indicator.
num_suspicious_keywords	Content	Counts the number of suspicious keywords in the URL	Quantifies the extent of deceptive content, enhancing detection of phishing intent.
has_special_chars	Content	True if the URL contains special characters (e.g., @, -, %), False otherwise	Special characters can obscure the true domain (e.g., google.com@phishingsite.com), a common phishing tactic.
num_special_chars	Content	Counts the number of special characters in the URL	Higher counts may indicate obfuscation attempts prevalent in phishing URLs.
has_encoded_chars	Content	True if the URL contains encoded characters (e.g., %20), False otherwise	Encoded characters are used to obfuscate content, a frequent phishing strategy to hide malicious intent.
tld_com	Domain	True if the TLD is .com, False otherwise (only .com in dataset)	TLDs can indicate legitimacy; phishing URLs may use obscure TLDs, though this dataset was limited to .com.
has_subdomains	Domain	True if subdomains are present, False otherwise (e.g., True for www)	Phishing URLs often use subdomains to mimic legitimate sites (e.g., login.google.phishingsite.com).
num_subdomains	Domain	Counts the number of subdomains (e.g., 1 for www)	A higher subdomain count can be a red flag for phishing attempts to deceive users.
has_ip_address	Domain	True if the URL uses an IP address instead of a domain (e.g., 192.168.1.1), False otherwise	Legitimate sites rarely use IPs; phishing URLs may use them to avoid domain registration or detection.
domain_length	Domain	Length of the domain name (e.g., 14 for www.google.com)	Phishing URLs often have longer domains to include deceptive content or mimic legitimate sites.

is_shortened_url	Domain	True if the URL is from a known shortening service (e.g., bit.ly), False otherwise	Phishing URLs often use shortening services to mask their true destination, making this a critical flag.
has_punycode	Domain	True if the domain uses Punycode (e.g., xn--), False otherwise	Punycode can create visually similar domains (e.g., göogle.com) for typosquatting, a phishing technique.
min_levenshtein_distance	Domain	Minimum Levenshtein distance to known legitimate domains (e.g., 0 for google.com)	Measures similarity to legitimate domains; small non-zero distances can indicate typosquatting (e.g., g00gle.com).
has_mixed_case	Domain	True if the domain has mixed uppercase/lowercase letters (e.g., GoOgLe), False otherwise	Mixed case can confuse users (e.g., GoOgLe.com), a subtle phishing tactic not typical in legitimate domains.
entropy	Statistical	Shannon entropy of the URL string, measuring character randomness	High entropy suggests randomness (e.g., x7a9p2q.com), common in phishing URLs to evade detection.
digit_to_letter_ratio	Statistical	Ratio of digits to alphabetic characters in the URL (e.g., 0 if no digits)	Phishing URLs may use more digits for random strings (e.g., login12345), unlike balanced legitimate URLs.
vowel_to_consonant_ratio	Statistical	Ratio of vowels to consonants in the domain (e.g., 0.5 for google.com)	Unusual ratios can signal random strings (e.g., x7a9p2q.com), common in phishing domains versus balanced legitimate ones.
has_https	Structural	True if the URL uses HTTPS, False otherwise	Legitimate sites often use HTTPS, while phishing sites may use HTTP or fake certificates, making it a key distinguishing factor.
num_dots	Structural	Counts the number of dots in the URL (e.g., 3 in www.google.com)	Phishing URLs may have more dots due to subdomains or obfuscation, helping identify structural anomalies.
num_slashes	Structural	Counts the number of slashes in the URL (e.g., 3 in https://www)	Higher slash counts can indicate complex paths or redirects, common in phishing URLs.
total_length	Structural	Total character length of the URL (e.g., 22 in https://www.google.com)	Phishing URLs may be longer due to added subdomains, paths, or deceptive content.
path_length	Structural	Length of the URL path (e.g., 0 for https://www.google.com)	Longer paths can indicate phishing attempts to simulate legitimate pages or hide malicious intent.
has_port_number	Structural	True if the URL specifies a port number (e.g., :8080), False otherwise	Non-standard ports are rare in legitimate URLs but may appear in phishing URLs pointing to malicious servers.
num_query_params	Structural	Counts the number of query parameters (e.g., 0 in https://www.google.com)	High query parameter counts can mimic legitimate login pages or pass malicious data in phishing URLs.

Table 2. *List of engineered features used pulled from URLs.*

These features were extracted using 28 custom python functions. The functions were then applied to the data set one by one, adding a new column with each iteration. This method enables the reusability we need for

checking new URLs. We want to be able to enter a random URL, parse out the chosen features, and tell us if it is legitimate or phishing.

After the feature creation, we reviewed the data frame and ensured that all boolean variables were transformed into numeric. This was a challenge for the top-level domain (TLD) feature. As we discovered over 1,400 TLDs in our dataset. Meaning that we needed to create over 1,400 columns for this one measure. After attempting to do so and running out of computer memory, we decided that the number of TLDs was unnecessary and narrowed it down to the top 20 shown below. Meaning that we removed the TLD column and added one for each (20) of the below. Everyone but one of the columns will be TURE (1) for each observation. As each URL can only have 1 (TURE) TLD.

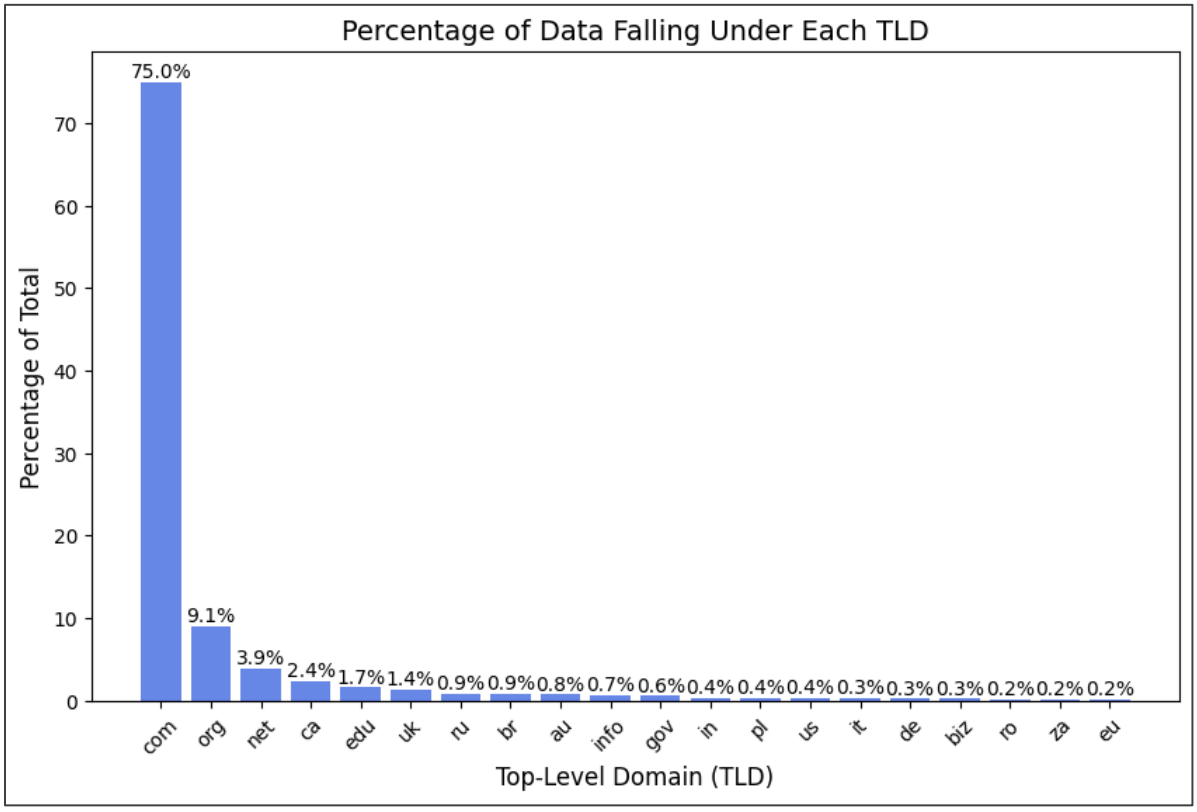

Figure 3. Top 20 TLDs (from extracted TLD feature).

Methods

For predicting phishing (malicious) URLs/links we tested two different binary classification models. Each chosen for their different strengths.

- Logistic Regression: A baseline model for its simplicity and interpretability.
- XGBoost: A gradient-boosting model chosen for its strong performance in classification tasks.

Before training the models, we first identified columns that were highly correlated to one another (multicollinearity) or insignificant to our target variable based on R Person Correlation coefficient and P values. The following we removed due to the below leaving us with 43 prediction features.

- Multicollinearity: ‘num_suspicious_keywords’, ‘total_length’

- Low R coefficient and/or high P values: [‘min_levenshtein_distance’, ‘tld_uk’, ‘tld_us’])

Model Training

For model training, the full ‘url’ column was dropped and the parsed data was split into training (80%) and testing (20%). The StandardScaler() function was then applied to all features to standardize the prediction variables by removing the mean and scaling to unit variance. Given that we have a range of variables (continuous and binary), this ensures that each variable has a mean of 0 and a standard deviation of 1. This is important for Logistic regression as it uses a linear decision boundary and is sensitive to the scale of input features. If features have different scales, the model may assign biased weights, leading to less accurate predictions. While it is not needed for XGBoost, as it is based on decision trees with feature thresholds, the StandardScaler() function was applied to all data before being trained on each model.

A standard Logistic Regression model was trained first giving us an accuracy of 98.80%. To see if we could improve it further, we then applied 5-fold cross validation. Training the final model on 5 different parts of the data set, leading to a slight decrease in overall accuracy (98.62%). XGBoost was trained next and produced an accuracy score of 99.54%. XGBoost also had the highest AUC-ROC (Area Under the Receiver Operating Characteristic Curve) meaning that it can distinguish between positive and negative classes nearly perfectly (1 being perfect).

Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC
Logistic Regression	0.988031	0.996638	0.979366	0.987927	0.997634
5CV Logistic Regression	0.986241	0.995647	0.976752	0.986108	0.997508
XGBoost	0.995356	0.998651	0.992053	0.995341	0.999281

Table 3. *Model performance comparison (table).*

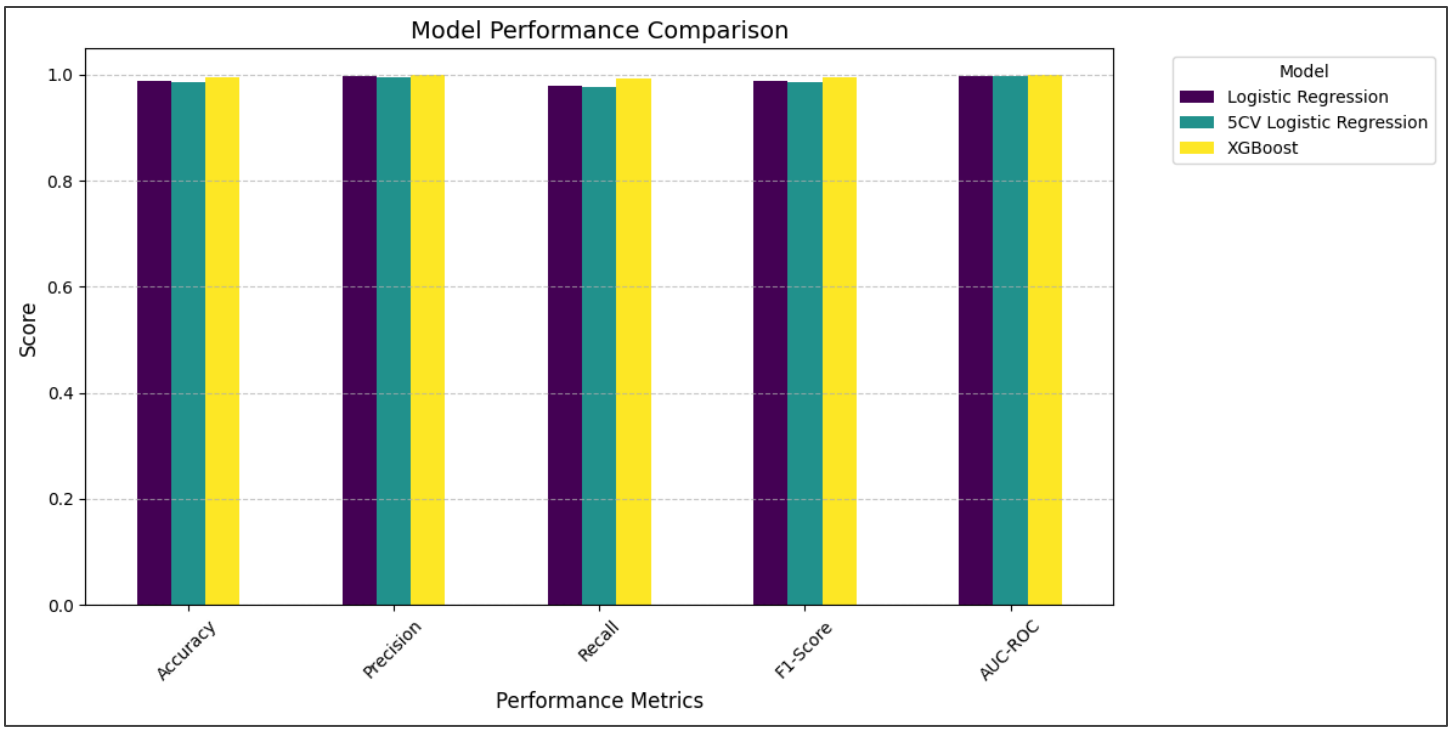

Figure 4. *Model performance comparison (chart).*

XGBoost also had the highest F1-Score (0.9953) meaning that it has the best balance between precision and recall. Precision being the proportion of true positive predictions (true positives / (true positives + false positives). And recall being the number of true positives divided by the total number of actual positives (true positives / (true positives + false negatives).

Analysis

After training each model, we took a look at what features impacted the XGBoost model's decision the most. As this is the model that we will be moving forward with based on it's performance. We analyzed feature importance two different ways. Frequence-Based and Gain-based importance.

Weight-Based performance (frequency) calculates the number of times a feature is used in a tree split (within the XGBoost model). A higher weight means the feature has been used more often during the model's training and is generally considered more important for making predictions. Figure 5 shows that the 'vowel_to_consonant_ratio' provided a significant improvement to the model's prediction accuracy, as it was used most often for tree decisions.

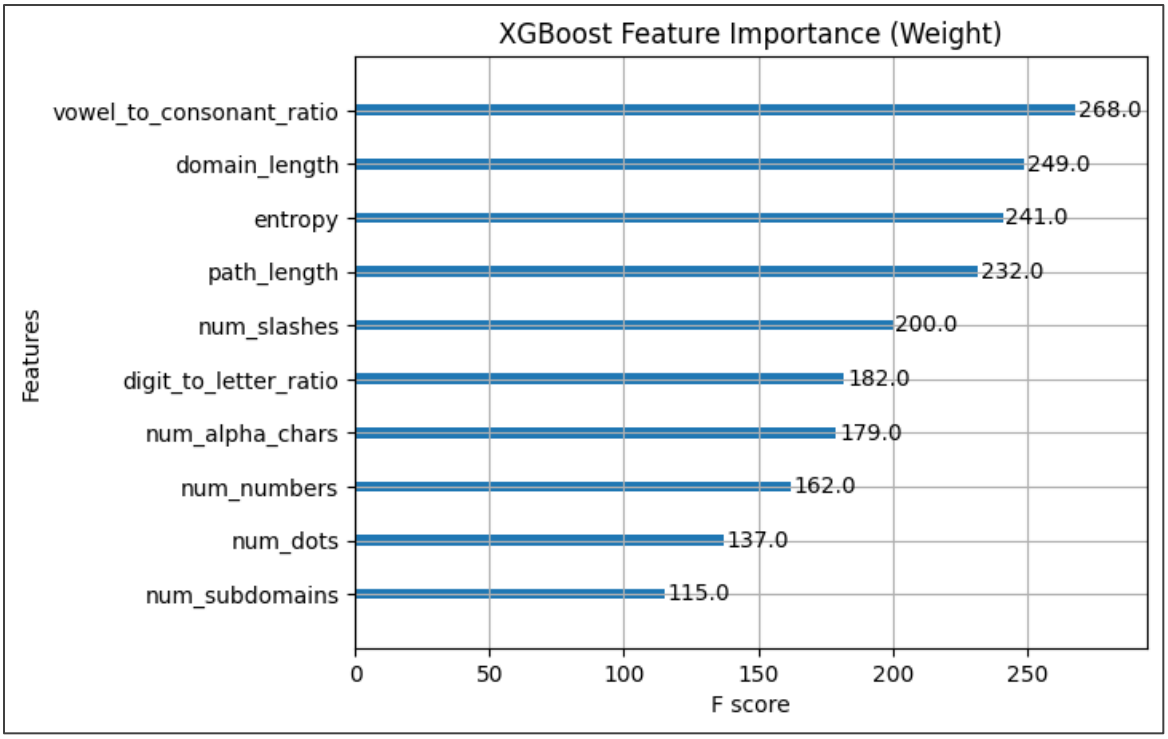

Figure 5. *XGBoost Top 10 Weight-Based performance (frequency) features.*

Gain-Based performance calculates the average improvement in accuracy (reduction in loss function) brought by a feature when it is used in the model. A feature with a high gain provides a significant improvement to the model's prediction accuracy. Figure 6 shows that the presence (or lack of) of an "s" at the end of "http" is an extremely important variable in terms of predicting a phishing URL when compared to the other features.

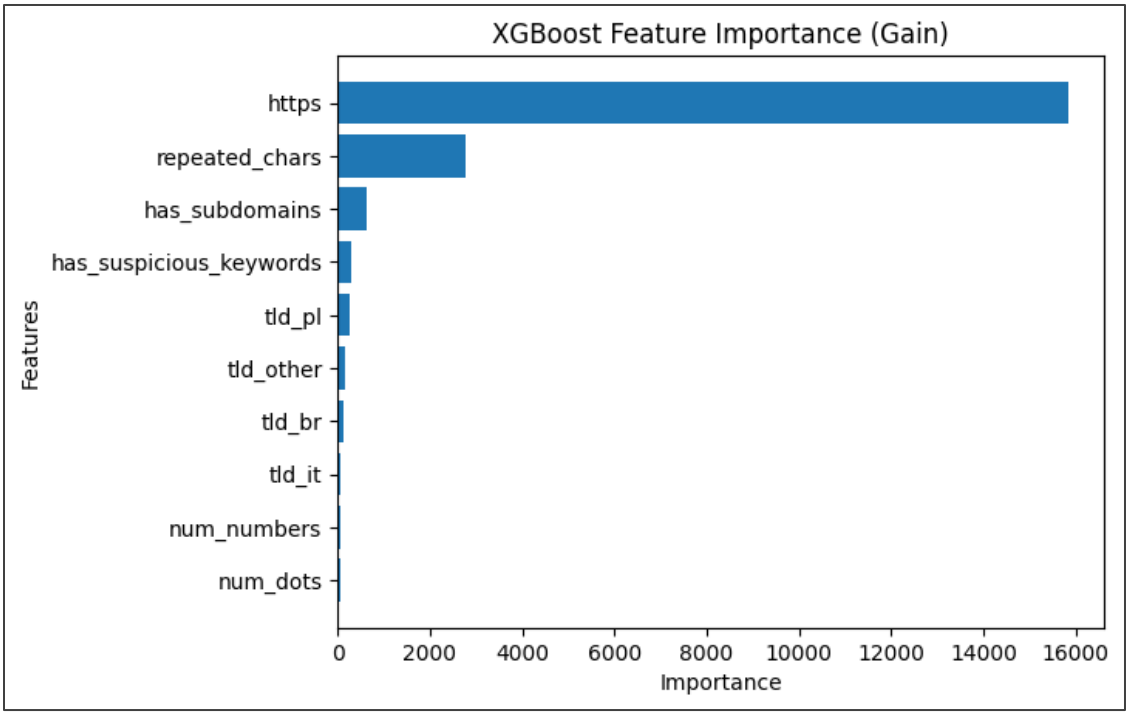

Figure 6. *XGBoost Top 10 Gain-Based performance features.*

Conclusion

To summarize, we succeeded in creating a model (XGBoost) with over 99% accuracy in predicting whether a URL is a phishing attempt or not. This was done by carefully choosing 43 engineered features and training two well-known machine learning models. The XGBoost outperformed the 5-Fold Cross Validation Logistic Regression model and our baseline studies, such as Tan et al.'s Random Forest Classifier (96%). This demonstrated the effectiveness of our feature selection and extraction and gradient-boosting approach. Key insights from feature importance analysis revealed that attributes like the presence of an "s" at the end of "http" (secured website) and the vowel-to-consonant ratio significantly enhance detection, offering valuable clues about phishing tactics.

Assumptions

During the development of our phishing link identification model, some assumptions were made due to the hazards around suspicious links. We assumed that the dataset from Kaggle contained a true list of phishing URLs. We were not able to test whether or not a link identified as "phishing" was truly malicious or not due to security concerns. However, we were able to spot check legitimate links, helping ensure accuracy of the legitimate links. We also assumed that this snapshot of URL patterns will remain representative of current phishing tactics.

Limitations

While our XGBoost model achieved a high accuracy of 99.54%, there are some limitations. First, the model was trained on a static dataset limited to a specific set of features, which may or may not generalize well to other URLs or to emerging phishing techniques (e.g., homograph attacks using non-Latin characters). Second, the reliance on URL-based features alone excludes contextual factors like email content, web-traffic,

website redirect count, actual site text/images, sender reputation, or user behavior, potentially missing sophisticated phishing attempts. Third, the undersampling approach to balance the dataset reduced the number of legitimate URLs, possibly discarding valuable patterns that could improve model robustness. Finally, the model's performance was evaluated in a controlled environment, and real-world deployment may reveal lower accuracy due to evolving phishing strategies.

Challenges

Beyond the challenges already noted (e.g., lack of distinguishing URL characteristics), we encountered difficulties in managing the computational load of processing over 200,000 URLs, particularly during feature extraction and TLD encoding. The initial attempt to create over 1,400 TLD columns exhausted memory resources, forcing a reduction to the top 20 TLDs, which may have omitted niche but relevant phishing patterns. Another challenge is the potential for false positives, where legitimate URLs with unusual structures (e.g., marketing campaign links) are flagged as phishing, risking user trust. Adapting the model to detect zero-day phishing attacks (those using novel tactics not present in the training data) remains an ongoing challenge requiring continuous updates.

Future Uses/Additional Applications

The phishing link identifier has potential beyond a simple model in a Jupiter notebook. It has the potential to be launched as a web-site for individuals to check if a URL is legitimate or not prior to clicking. It could be integrated into email clients, messaging apps, web-browsers, or social media platforms to provide real-time warnings about malicious links. While there are tools out there that check links against a known list of bad URLs, it can be difficult to keep up with and many new sites fall through the cracks. Our model gives enterprises using this preexisting tool, another avenue to protect employees from phishing attempts before they ever make it on the "bad list". The model could also be adapted for educational purposes, training individuals to recognize phishing characteristics by highlighting flagged features (the missing "s" in "http" for example. Additionally, extending the approach to detect other malicious content, such as ransomware download links or deepfake-related URLs, could broaden its impact. Pairing it with natural language processing to analyze email or message text alongside URLs could further enhance detection capabilities.

Recommendations

Based on the findings, we recommend deploying the XGBoost model as a browser extension or API service, allowing users to check URLs in real time. To improve accuracy, we suggest incorporating additional features, such as website metadata (e.g., registration age, SSL certificate validity) or network traffic patterns, and regularly updating the training dataset with new phishing examples. Cross-validation on the XGBoost model with diverse datasets beyond Kaggle's should be conducted to ensure generalizability. For practical use, we advise providing users with clear explanations of why a URL was flagged (e.g., missing "s" in "http" or contains suspicious keywords) to foster trust and education. Finally, collaboration with cybersecurity firms could help refine the model and align it with industry standards.

Implementation Plan

1. Development Phase (1-2 months): Finalize the XGBoost model by optimizing hyperparameters and integrating it into a lightweight Python script compatible with web or desktop applications.
2. Integration Phase (2-3 months): Develop a browser extension (e.g., for Chrome, Firefox) and an API endpoint, ensuring low-latency URL parsing and classification. Partner with a cloud provider for scalable hosting.
3. Testing Phase (1 month): Conduct beta testing with a small user group to identify false positives/negatives and gather feedback on usability.
4. Deployment Phase (1 month): Release the extension on browser stores and the API for public use, accompanied by documentation and a feedback portal.
5. Maintenance Phase (Ongoing): Update the model quarterly with new phishing data, monitor performance metrics (e.g., accuracy, AUC-ROC), and address user-reported issues.
6. A budget for cloud hosting, developer time, and marketing should be secured, alongside compliance with data privacy laws (e.g., GDPR).

Ethical Assessment

The development and potential deployment of our phishing link identifier raises several ethical considerations that must be addressed to ensure responsible use. Firstly, the model's ability to classify URLs as phishing or legitimate carries the risk of false positives and negatives, which could impact user trust and safety. A false positive might flag a legitimate business URL, potentially causing financial or reputational harm to the affected entity, while a false negative could allow a malicious link to reach users, leading to data breaches or financial loss. To mitigate this, we must prioritize transparency by informing users of the model's probabilistic nature and providing a mechanism (e.g., a feedback form) to report misclassifications, which can then be used to refine the model.

Privacy concerns also need to considered. As we want to empower users to check URLs, we need to ensure that these submissions do not contain personal information. If any data is stored, users will be clearly informed about what data is analyzed and how it is handled before using our tool.

The decision to balance the dataset also introduces an ethical trade-off. By undersampling legitimate URLs (from 77% to 50%), we reduced bias toward the majority class but may have discarded valuable patterns in legitimate URLs at the same time. Potentially skewing the model's perception of "normal" behavior. We must acknowledge this limitation to users and researchers, ensuring that the model's design choices are not misrepresented as inherently neutral or infallible.

Lastly, equitable access and education are critical. If deployed, the tool should be accessible to non-technical users and accompanied by resources explaining phishing risks and the reasoning behind model outputs (why the link was flagged or not, what features flagged a suspicious link). This empowers users to make informed decisions rather than relying solely on automated judgments.

By addressing these ethical dimensions accuracy, privacy, transparency, and accessibility, we can offer a tool that enhances cybersecurity without compromising user rights or trust.

References

1 *Experi-Metal Inc v. Comerica Bank, No. 2:2009cv14890 - Document 66 (E.D. Mich.)*. (2011). Justia Law. https://law.justia.com/cases/federal/district-courts/michigan/miedce/2:2009cv14890/244989/66/

2 Hauser, C. (2018, July 26). U.S. breaks up vast I.R.S. phone scam. *The New York Times*. https://www.nytimes.com/2018/07/23/business/irs-phone-scams-jeff-sessions.html

Benchmark Comparisons (for model performance)

3 Tan, K. H., & MacGregor, A. (2022). An efficient phishing website detection plugin service for existing web browsers using random forest classifier. Authorea. https://doi.org/10.22541/au.166792122.20565582/v1

4 Zhang, H., Liu, G., Chow, T. W. S., Liu, W. (2011). Textual and visual content-based anti-phishing: A Bayesian approach. IEEE Transactions on Neural Networks, 22(10), 1532–1546. DOI: 10.1109/TNN.2011.2161999

5 Chandan, C. J., Chheda, H. P., Gosar, D. M., Shah, H. R., & Bhave, P. U. (2014). A machine learning approach for detection of phished websites using neural networks. International Journal on Recent and Innovation Trends in Computing and Communication, 2(12), 4205–4209. DOI: https://doi.org/10.17762/ijritcc.v2i12.3639

6 Shah, B., Dharamshi, K., Patel, M., & Gaikwad, V. (2020). Chrome extension for detecting phishing websites. International Research Journal of Engineering and Technology (IRJET), 7(3), 2958–2962. https://www.irjet.net/archives/V7/i3/IRJET-V7I3590.pdf