Building a Machine Learning Model for detecting Phishing URLs

Introduction

Phishing URLs are deceptive website links created by cybercriminals to trick users into clicking on them, potentially exposing their sensitive information. These malicious links are often distributed through spam emails, messages, and other fraudulent means.

According to https://aag-it.com/the-latest-phishing-statistics/:

- Phishing is the most common form of cyber-crime, with an estimated 3.4 billion spam emails sent every day.
- 83% of UK businesses that suffered a cyber-attack in 2022 reported the attack type as phishing.
- Phishing was the most common attack type against Asian organizations in 2021.
- The average cost of a data breach against an organization is more than $4 million.

Some of the notable phishing attacks are (https://aag-it.com/the-latest-phishing-statistics/):

- 2015 FACC Whaling Attack: FACC recording losses of 23.4 million euros ($24 million) for the 2015/16 financial year.
- 2014 Sony Pictures Phishing Attack: In total, the attack cost Sony an estimated $100 million to resolve.
- 2021 Colonial Pipeline attack: Colonial Pipeline was forced to pay around $4.4 million to the hackers to regain control of their systems.

The above is only a few. The severity of phishing attack may be catastrophic to an organization.

With advancement in technology, cyber criminals are devising new and sophisticated ways to hack into life of people. In 2023, phishing attacks increased by 173% from the previous quarter.

This project aims to build a machine learning model for detecting phishing URLs by focusing on analyzing the inherent characteristics of phishing URLs rather than relying on external data sources like robots.txt, CSS values, or WHOIS records. Scraping such information can be tedious, and in many cases, it may not even be available. By leveraging a simple yet effective methodology, this approach enables efficient phishing URL detection with minimal computational overhead.

Methodology

This approach focuses on analyzing the inherent characteristics of phishing URLs rather than relying on external data sources like robots.txt, CSS values, or WHOIS records. Scraping such information can be tedious, and in many cases, it may not even be available. By leveraging a simple yet effective methodology, this approach enables efficient phishing URL detection with minimal computational overhead.

Key Methodological Choices:

• URL-Based Feature Engineering: Extracting meaningful attributes from URLs.

• Caching for Performance Optimization: We use Python's @lru_cache(maxsize=None) from the functools module to store function results, improving computational efficiency.

• Machine Learning Classifiers: Multiple models are trained to detect phishing URLs based on extracted features.

Features Used for Detection:

• URL Length

• Top-Level Domain (TLD) Similarity Scores

• Probability Distribution of Special Characters

• Entropy-Based Analysis (Measures randomness in URLs)

- Obfuscation Detection (Encoded characters, hexadecimal IPs, hidden redirections)
- Text Similarity Metrics (To identify deceptive domain names)
- Word-Based Features (Using NLTK's English corpus)
- Log Transformations (To normalize skewed data)

Data Sources

We used a well-documented phishing and legitimate URL dataset from Kaggle, ensuring it was suitable for machine learning.

Data Cleaning

The dataset is of very good conditions. Only a few inconsistencies in the URLs, including some non-ASCII character, occasional non alphanumeric characters at the beginning of the URLS and additional https//:'s or www's.

Feature Engineering

Since the raw dataset contained only URLs and their legitimacy labels, we applied various feature extraction techniques to transform the data into a structured format for machine learning.

Feature Engineering Steps:

1. Extracting URL Components:

   - Fully Qualified Domain Name (FQDN)

   - Top-Level Domain (TLD)

2. Shannon Entropy-Based Features:

   - Phishing URLs tend to be more random than legitimate URLs. We used Shannon Entropy to quantify randomness in URL structures.

3. Text Similarity-Based Features:

   - To detect deceptive domains, we compared TLDs against a legitimate TLD list from Wikipedia, using:

     a. Levenshtein Distance (Fuzzy Score)

     b. Damerau-Levenshtein Normalized Distance

     c. Jaro-Winkler Algorithm

4. Unique Word Chunks:

   - Extracted distinctive word patterns from FQDNs to match against known phishing and legitimate word patterns.

5. URL Obfuscation Detection:

   - Identified techniques like hexadecimal/octal IP representations, encoded characters (%2F instead of /), and hidden redirections.

   - Calculated entropy to detect obfuscation attempts.

6. Log Transformations:

Many ML algorithms perform better when features follow a normal distribution.

- Applied log transformation to reduce skewness in numerical features.

7. Using NLTK corpus
In this project we do not exploit the intricate features of the NLTK. We simply use the English words that are stored in the corpus to match the extracted words from the URLs.

Machine Learning Algorithms

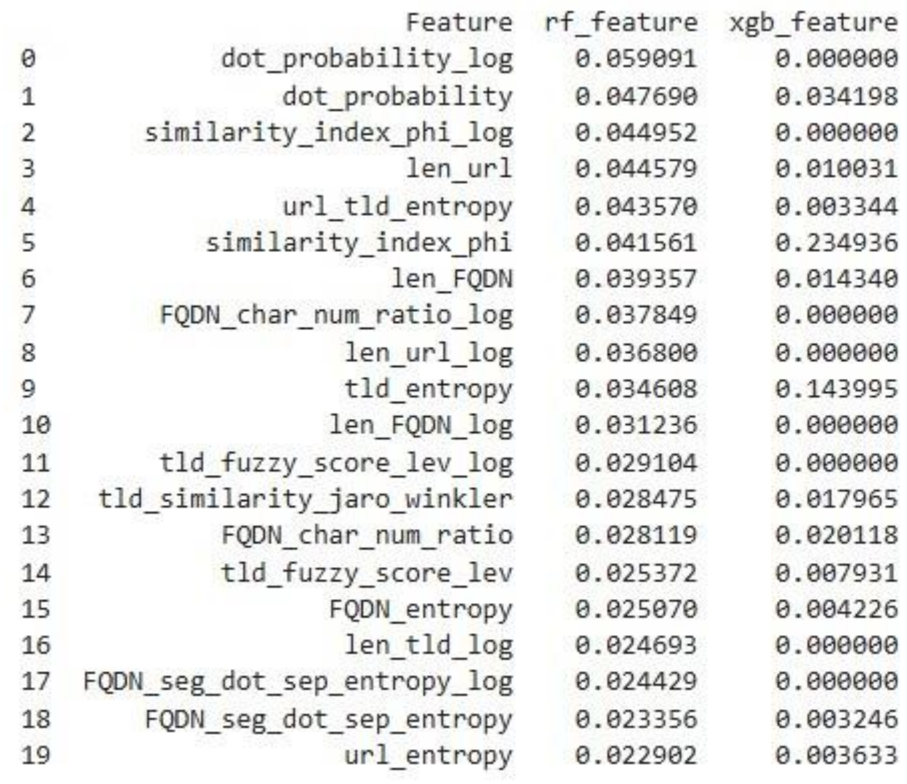
We observed during our model training XGBoost Classifier handles the skewness of the features gracefully than RandomForest Classifier through gradient boosting and tree depth tuning. Therefore, XGBoost didn't gain much from the log transformed featured. As we can see from the table (Fig. 1.1) below:

Many log-transformed features have 0.0000 importance in XGB but nonzero values in RF (e.g., dot_probability_log, similarity_index_phi_log, len_url_log).

RF considers more features significant, with no single feature dominating.

Even log-transformed features like dot_probability_log (0.0591), similarity_index_phi_log (0.0449), and FQDN_char_num_ratio_log (0.0378) contribute.

This indicates that RF benefits from a diverse set of features, unlike XGB, which is more selective.



```
                        Feature  rf_feature  xgb_feature
0           dot_probability_log    0.059091     0.000000
1               dot_probability    0.047690     0.034198
2      similarity_index_phi_log    0.044952     0.000000
3                       len_url    0.044579     0.010031
4               url_tld_entropy    0.043570     0.003344
5          similarity_index_phi    0.041561     0.234936
6                      len_FQDN    0.039357     0.014340
7       FQDN_char_num_ratio_log    0.037849     0.000000
8                   len_url_log    0.036800     0.000000
9                   tld_entropy    0.034608     0.143995
10                 len_FQDN_log    0.031236     0.000000
11      tld_fuzzy_score_lev_log    0.029104     0.000000
12  tld_similarity_jaro_winkler    0.028475     0.017965
13          FQDN_char_num_ratio    0.028119     0.020118
14          tld_fuzzy_score_lev    0.025372     0.007931
15                 FQDN_entropy    0.025070     0.004226
16                  len_tld_log    0.024693     0.000000
17 FQDN_seg_dot_sep_entropy_log    0.024429     0.000000
18     FQDN_seg_dot_sep_entropy    0.023356     0.003246
19                  url_entropy    0.022902     0.003633
```

Fig 1.1: Feature importance for XGBoost Classifier and RandomForest Classifier

We also used PCA on our dataset, but carefully chosen basic feature with some domain knowledge and intuition yields best scores.

Our final decision was to use the basic features as XGBoost is performing much better than RandomForest Classifier.

We used K Nearest Neighbors (K-NN). Tried the elbow method to find the optimal n_neighbors, but it was not obvious from the graph the value of best n_neighbors (Fig. 1.2). Though n_neighbors=5 achieved best scores, we used n_neighbors=10 to minimize variance and to achieve a balance between bias and variance.

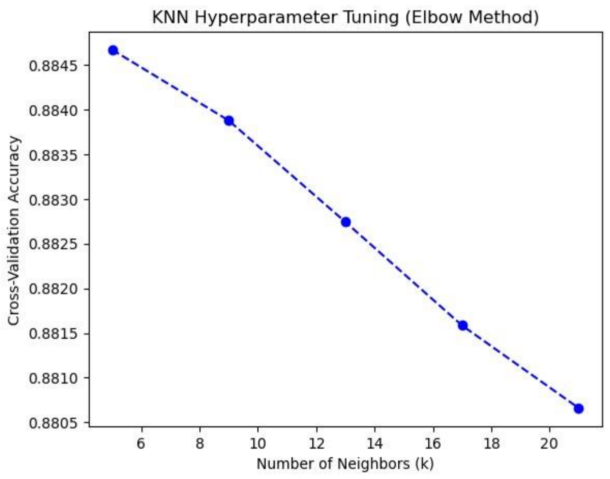


Fig: 1.2: Elbow method

We also used Support Vector Classifier, which prove to be very computational heavy algorithm. So, we downsized our dataset 1:10 while keeping the class balance intact. After optimizing with GridSearchCV on the small dataset, we applied the algorithm on the full dataset with optimized parameters.

To tune the hyperparameters of RandomForest Classifier we used RandomSearchCV. After applying TF-IDF on the 'url' column of our dataset we combined the result with our existing dataset and used the tuned RandomForest Classifier model on the whole dataset with TF-IDF features setting scale_pos_weight=1.5. This yield the highest F1 score of 0.90/0.92 for Legitimate/Phishing URL classification among the three algorithms.

Future Improvements

URLs are primarily text-based, and there are many powerful techniques to extract more meaningful insights. For example, techniques like Word2Vec could be applied to compare URL components in more sophisticated ways. Currently, similarities are calculated based on exact matches due to limitations in computational power, but partial matching could yield better performance.

In future work, instead of using a static similarity index between top-level domains extracted from Wikipedia, we could dynamically compute the similarity between URLs extracted directly from the web and the URLs in our dataset. This would make the approach more adaptive and up-to-date.

Additionally, tools from NLTK, such as Latent Semantic Analysis (LSA), or even more advanced models like BERT, could be applied to improve the comparison and scoring process.

I'm particularly excited to explore TF-GNN from TensorFlow in the future, as it enables building Graph Neural Networks (GNNs). This would be especially useful for problems where the relationships between data points—such as URLs—can be effectively represented as a graph, allowing for deeper insights into phishing patterns and network structures.