*Report:*

# Phishing URLs detector

by Samay Andes Sisa Ruiz Muenala

**Data Mining and Machine Learning**

Master of Science in Artificial Intelligence and Data Engineering

Department of Information Engineering

University of Pisa

a.y. 2023-2024

# Contents

# 1. Introduction

I have decided to develop software capable of recognising phishing site URLs. At the heart of this project is the creation of a classification model based on machine learning algorithms acquired during the Data Mining and Machine Learning course.

The first part of this document will regard the construction of the machine-learning model: data preprocessing, model selection and evaluation, comparison with state of the art. On the other hand the second part will briefly explain the software implementation side of the detector.

Github repository: https://github.com/sisaruiz/Phishing-URLs-inspector

# 2. Preliminary exploratory analysis

The initial dataset considered for this project has been retrieved from Kaggle[1]. It was created manually by the uploader gathering many datasets from websites offering malicious links[2]. Downloaded and uncompressed, the file's size amounts to 30.1 Mb.

The following evaluations have been obtained via Python scripts reported on the Jupyter notebook associated with the project.

## 2.1. Descriptive statistics

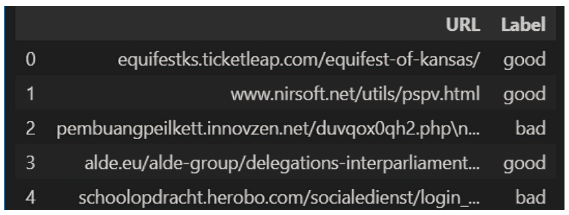
The data file is structured as a very simple matrix. It is made up of 549,346 entries (rows), each with two attributes. The first column, *URL*, contains the full URL address for each entry. The second one, called *Label*, reports the status of the website: *good* for benign pages, *bad* for malicious ones (Fig. 1).

Moreover, the dataset contains no missing values but it does include redundancy as the number of unique values reported for *URL* (507,195) is lower than the total number of rows. Additionally, the datasheet appears very imbalanced. In fact, the presence of harmless sites is

[1] *https://www.kaggle.com/datasets/taruntiwarihp/phishing-site-urls*

[2] not explicitly referenced by the uploader

much more significant compared to that of scamming sites: 392,924 (71.5%) vs 156,422 (28.5%) (Fig. 2).



| | URL | Label |
|---|---|---|
| 0 | equifestks.ticketleap.com/equifest-of-kansas/ | good |
| 1 | www.nirsoft.net/utils/pspv.html | good |
| 2 | pembuangpeilkett.innovzen.net/duvqox0qh2.php\n... | bad |
| 3 | alde.eu/alde-group/delegations-interparliament... | good |
| 4 | schoolopdracht.herobo.com/socialedienst/login_... | bad |

Fig. 1. Result of `df.head()`

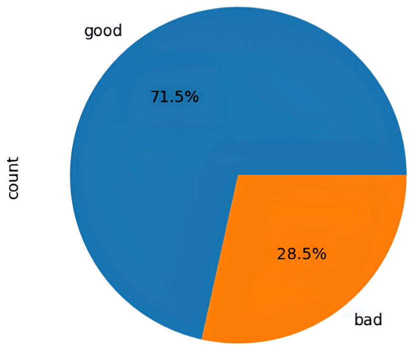


Fig. 2. Distribution of *Label*

Given all the aforementioned peculiarities, a pre-processing phase needs to be performed.

## 3. Data preprocessing

### 3.1. Additional data and undersampling

Firstly, in order to tackle the imbalance issue, I increased the amount of phishing URLs by retrieving recent data from PhishTank[3]. More specifically, I selected the URLs reported (and verified) to be malign inserted in the database from July 26th 2020[4] onwards.
After more processing of this data, I was finally able to merge it with the main one. I then proceeded to remove duplicates and was eventually able to verify that the improvement has been minimal as seen on Fig. 3.

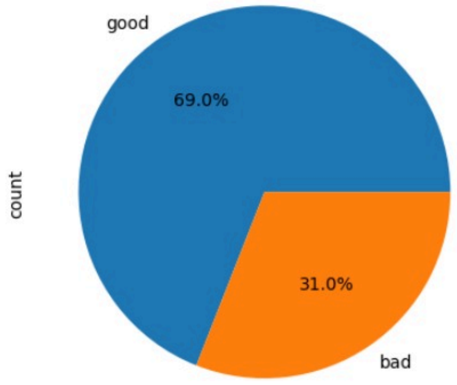


Fig.3. Distribution of *Label* after addition of more data

[3] https://phishtank.org/

[4] On the aforementioned Kaggle page, a youtube video with more explaining is linked. I considered the date on which it was uploaded as time reference for the dataset generation ( https://www.youtube.com/watch?v=zKNXHluHneU )

Before sampling the dataset I have conducted a feature engineering phase to extract possible features to use for training. Notice that this phase is limited only to feature extraction as the features are selected only afterwards on the training set only after splitting.

## 3.2. Feature engineering

I decided to focus on URL-based attributes only, since gaining data on the webpage content would require access to malicious sites, potentially leading to security risks or exposure to malware which I'm not well equipped to counter.
Given that the original dataset contained no features besides the URL itself, it was necessary to carry out a feature creation phase. To guide this process, I used another phishing websites dataset with 111 features as a reference for which attributes to extract[5].
Its attributes can be broadly split into 9 categories:

1. URL composition: *qty_dot_url*, *qty_hyphen_url*, *qty_underline_url*, etc...
2. Domain level: *qty_dot_domain, qty_hyphen_domain, domain_length,* etc..
3. Directory path: *qty_dot_directory, qty_hyphen_directory, directory_length,* etc…
4. File-level: *qty_dot_file, qty_hyphen_file, file_length,* etc..
5. Parameter: *qty_dot_params, qty_hyphen_params, params_length,* etc…
6. Time: *time_response, time_domain_activation, time_domain_expiration,* etc..
7. Security: *domain_spf, tls_ssl_certificate*
8. IP and server: *asn_ip, qty_ip_resolved, qty_nameservers, ttl_hostname,* etc...
9. Miscellaneous: *url_google_index, domain_google_index, url_shortened*, etc..

Unfortunately some features were not feasible therefore have been filtered out[6]. Thus, the attributes to generate are reduced to 91: *'qty_hyphen_url', 'qty_underline_url', 'qty_slash_url', 'qty_questionmark_url', 'qty_equal_url', 'qty_at_url', 'qty_and_url', 'qty_exclamation_url', 'qty_tilde_url', 'qty_comma_url', 'qty_plus_url', 'qty_asterisk_url', 'qty_hashtag_url', 'qty_dollar_url', 'qty_percent_url', 'qty_tld_url', 'length_url', 'qty_dot_domain', 'qty_hyphen_domain', 'qty_underline_domain', 'qty_slash_domain', 'qty_questionmark_domain', 'qty_equal_domain', 'qty_at_domain', 'qty_and_domain', 'qty_exclamation_domain', 'qty_tilde_domain', 'qty_comma_domain', 'qty_plus_domain', 'qty_asterisk_domain', 'qty_hashtag_domain', 'qty_dollar_domain', 'qty_percent_domain', 'qty_vowels_domain', 'domain_length', 'domain_in_ip', 'server_client_domain', 'qty_dot_directory', 'qty_hyphen_directory', 'qty_underline_directory', 'qty_slash_directory', 'qty_questionmark_directory', 'qty_equal_directory', 'qty_at_directory', 'qty_and_directory', 'qty_exclamation_directory', 'qty_tilde_directory', 'qty_comma_directory', 'qty_plus_directory', 'qty_asterisk_directory', 'qty_hashtag_directory', 'qty_dollar_directory', 'qty_percent_directory', 'directory_length', 'qty_dot_file', 'qty_hyphen_file', 'qty_underline_file', 'qty_slash_file', 'qty_questionmark_file', 'qty_equal_file', 'qty_at_file', 'qty_and_file', 'qty_exclamation_file', 'qty_tilde_file', 'qty_comma_file', 'qty_plus_file', 'qty_asterisk_file', 'qty_hashtag_file', 'qty_dollar_file', 'qty_percent_file', 'file_length', 'qty_dot_params', 'qty_hyphen_params', 'qty_underline_params', 'qty_slash_params', 'qty_questionmark_params', 'qty_equal_params', 'qty_at_params', 'qty_and_params', 'qty_exclamation_params', 'qty_tilde_params', 'qty_comma_params', 'qty_plus_params', 'qty_asterisk_params',*

[5] Vrbančič, Grega (2020), "Phishing Websites Dataset", Mendeley Data, V1, doi: 10.17632/72ptz43s9v.1
[6] These features include google index, security, IP and server related features.

*'qty_hashtag_params', 'qty_dollar_params', 'qty_percent_params', 'params_length', 'tld_present_params', 'qty_params', 'email_in_url'.*

Notice that the absence of a certain URL part (directory path, parameters, file level) is signaled by setting 0 as value in the binary presence features (has_dir, has_params, etc. ). All other features specific to the absent subpart are set to 0 as well.

In addition to all these features, I created one last column for storing the values of shannon entropy, which I thought could be a valuable insight into the URLs structures. The formula used is:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Where:

- $p_i$ is the probability of each unique character $i$ occurring in the URL, which is calculated as the frequency of the character divided by the total length of the URL.
- $n$ is the number of unique characters in the URL.

Note that most of the decimal values in this column fall between 2 and 6.

### 3.2.1. Feature analysis

Once I obtained all the features, I proceeded to evaluate them before selection. This evaluation has been done on the mutual information score, thus assessing the correlation between each feature and the label target class.
Most than half of the features generated had a score below 0.01. For better visualization Fig. 4 shows only the top 50 features by MI score.
Interestingly, the highest scoring features are *shannon_entropy*, and two binary presence features: *has_dir* and *has_file*. Characters-counting features appear to be less important for discrimination.

### 3.2.2. Features selection

Features selection is done afterwards while evaluating all models. It will still be based on the Mutual Information Score. Notice that each trial will select the same number of features, 35[7], although the actual features selected will be different per fold.

[7] I chose this number based on several papers that I read on this topic. One example: Sakib Shahriar Shafin, An Explainable Feature Selection Framework for Web Phishing Detection with Machine Learning, Data Science and Management, 2024, where the author achieved optimal accuracy results by using 48 (95.47%) and then 26 features (97.18%).

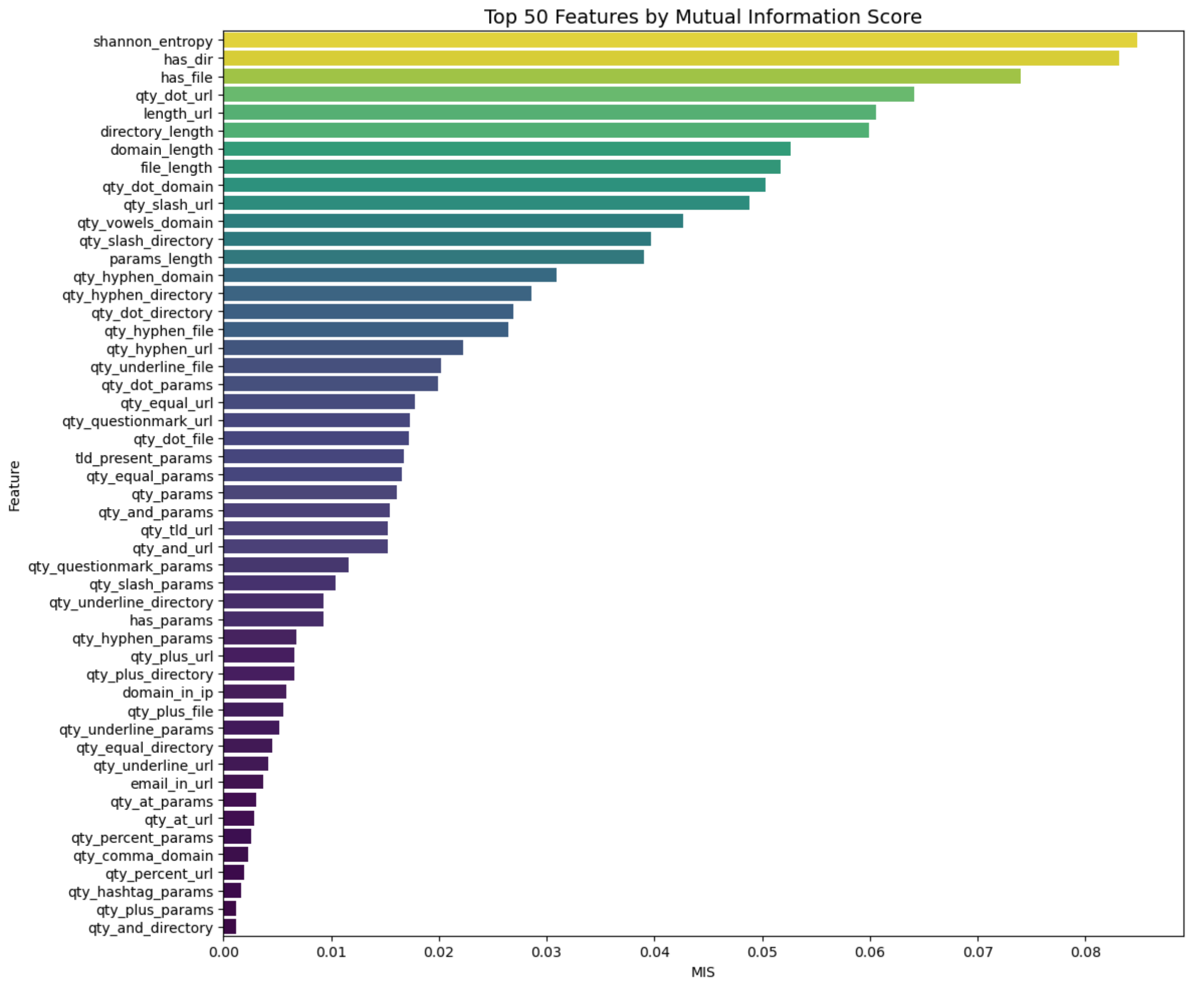


Fig.4. Top 50 features by Mutual Information Score

# 4. Model evaluation

Several models have been chosen for training and then evaluation. These are:

- Random Forest
- Extreme Gradient Boosting (XGBoost)
- Light Gradient-Boosting Machine (LightGBM)
- Adaptive Boosting (AdaBoost) with Decision Tree as base estimator
- Logistic Regression
- Linear SVM
- GaussianNB

Since most methods are negatively affected by imbalance in datasets, I evaluated 2 methods for obtaining equal classes distribution: undersampling and oversampling (SMOTE).
Hence, I first trained all the above-mentioned models on an undersampled dataset, later I trained the same models on an oversampled dataset.
In order to evaluate thoroughly the performance of each ML classifier, I used a 10-fold stratified cross-validation approach. Each fold applies a pipeline which performs, in order :

1. Undersampling (or Oversampling)
2. Feature selection (top 35 by Mutual Information Score)
3. Classifier model selection

```
pipe = Pipeline([
    ('undersample', undersampler),
    ('feature_selection', feature_selector),
    ('classification', model)
])

# cross-validation
cv_results = cross_validate(pipe, X, y, cv=cv, scoring=scoring, return_train_score=False)
```

For each fold I evaluated the models' performances on the following metrics: accuracy, f1, precision, recall, roc-auc.
After obtaining all scores for each fold, I computed the average score and to assess the statistical significance of differences in values, I used the Wilcoxon Test.
To account for multiple pairwise comparisons among models, I applied the Bonferroni correction to adjust the significance threshold. For each pair of models, the Wilcoxon signed-rank test was performed on their F1 scores across the folds, and if the result was statistically significant, a win was assigned to the model with the higher average score. By counting these wins, I identified the model or models that consistently outperformed others in a statistically significant way. Results follow.

## 4.1. Undersampling

The table below shows the average (on 10 folds) performance scores for each model applying undersampling (`RandomUnderSampler(random_state=21)`).

| Models | accuracy | f1 | precision | recall | roc-auc |
|---|---|---|---|---|---|
| Random Forest | 0.883265 | 0.823150 | 0.775522 | 0.877021 | 0.881544 |
| XGBoost | 0.865481 | 0.799164 | 0.743392 | 0.863994 | 0.865071 |
| LightGBM | 0.854995 | 0.784474 | 0.726926 | 0.851925 | 0.854149 |
| AdaBoost | 0.781970 | 0.687768 | 0.618144 | 0.775214 | 0.780108 |
| Logistic Regression | 0.719235 | 0.605392 | 0.536256 | 0.695332 | 0.818319 |
| Linear SVM | 0.720683 | 0.601127 | 0.538996 | 0.679475 | 0.814966 |
| GaussianNB | 0.740640 | 0.396860 | 0.709674 | 0.275557 | 0.781415 |

The Wilcoxon Test assessed the statistical significance of the performance of the Random Forest Model (based on F1 score).

## 4.2. Oversampling (SMOTE)

Below are shown the average results of all models trained on a SMOTE-oversampled dataset (`SMOTE(random_state=21)`).

| Models | accuracy | f1 | precision | recall | roc-auc |
|---|---|---|---|---|---|
| Random Forest | 0.890570 | 0.827561 | 0.808363 | 0.847700 | 0.953740 |
| XGBoost | 0.865486 | 0.798692 | 0.744477 | 0.861430 | 0.944756 |
| LightGBM | 0.853275 | 0.782534 | 0.723401 | 0.852220 | 0.935435 |
| AdaBoost | 0.779235 | 0.689441 | 0.611029 | 0.791077 | 0.874066 |
| Logistic Regression | 0.723771 | 0.610838 | 0.542069 | 0.699863 | 0.815739 |
| Linear SVM | 0.722039 | 0.604646 | 0.540397 | 0.686297 | 0.812167 |
| GaussianNB | 0.742534 | 0.407635 | 0.709450 | 0.286197 | 0.781469 |

Again, the Wilcoxon test confirmed that Random Forest performs the best among all models.

The results show that Random Forest outperforms all other models in both undersampling and SMOTE-based oversampling, especially in terms of F1 score, which is crucial for imbalanced datasets. In the undersampling scenario, it achieved an F1 of 0.823150, and this improved to 0.827561 with SMOTE, along with a notably high ROC AUC of 0.953740.

Logistic Regression, Linear SVM, and GaussianNB had lower F1 and recall scores, indicating weaker performance in identifying minority class instances. This is partly due to model bias: Logistic Regression and Linear SVM are linear models, which makes them less effective when the dataset has non-linear decision boundaries. GaussianNB, on the other hand, assumes feature independence and normally distributed data—assumptions that are particularly unrealistic in this case, as the features are primarily text-based count features, which introduce strong correlations.

## 4.2. Random Forest: undersampling or SMOTE?

Given that out of all models trained using undersampling or SMOTE, Random Forest results the one performing best, this is the classifier I will focus now onwards.
In order to fairly compare the undersampled version and the SMOTE one, it is important to observe that the values obtained have been very likely computed on different subsets of training and testing sets. Hence, I proceeded in the following way:

1. Stratified 10-Fold Cross-Validation with a fixed random state (21) was used to ensure both resampling methods operate on the same train/test splits.

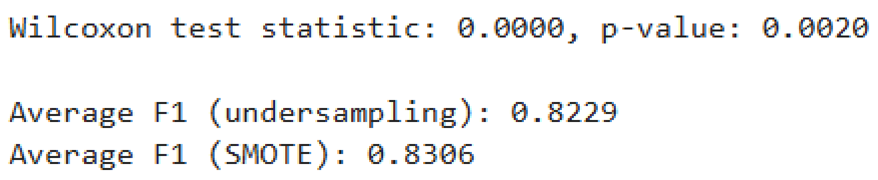
2. Feature selection was applied, selecting the top 35 features (MI score) from the training data and applying the same transformation to the test data.

3. A Random Forest classifier was trained twice per fold: once on data balanced with Random Undersampling, and once on data balanced with SMOTE.

4. Both models were evaluated on the same test set using the F1 score.

5. The F1 scores across folds were compared using the Wilcoxon Signed-Rank Test to assess statistical significance.

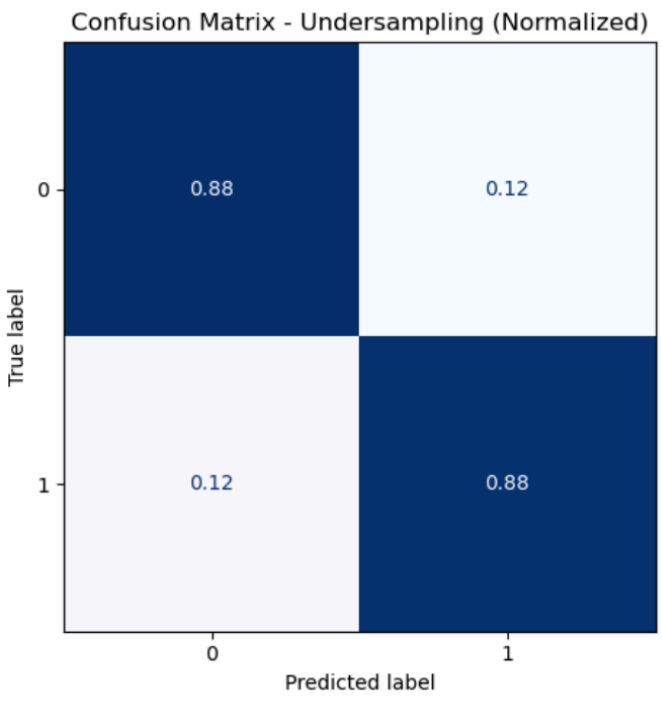
This procedure resulted in:

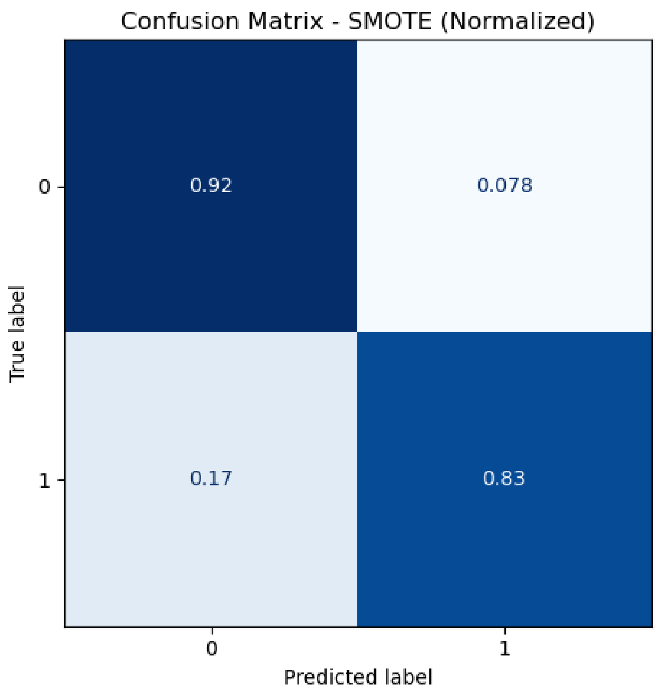


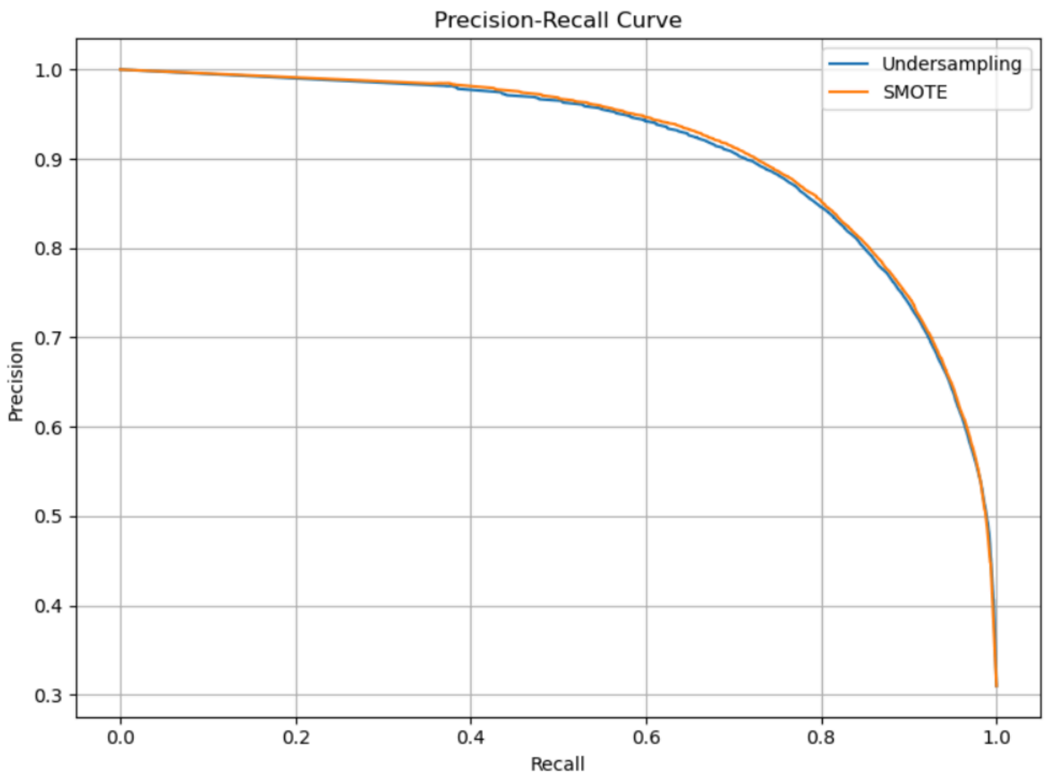
```
Wilcoxon test statistic: 0.0000, p-value: 0.0020

Average F1 (undersampling): 0.8229
Average F1 (SMOTE): 0.8306
```

```
=== Summary Table ===
             Metric  Undersampling      SMOTE
0       Recall (bad)         0.8762     0.8269
1    Precision (bad)         0.7702     0.8270
2           F1 (bad)         0.8198     0.8270
3           F2 (bad)         0.8527     0.8269
4            ROC-AUC         0.9521     0.9530
5    False Negatives      4366.0000  6104.0000
```







While the SMOTE-based model shows slightly better overall metrics (F1, accuracy, ROC-AUC), the undersampled model achieves notably higher recall for the minority class ("bad" URLs), which is critical in phishing detection.

Since our priority is minimizing false negatives, and the next step will focus on tuning hyperparameters to improve recall (with ROC-AUC as a secondary goal), the undersampled model is a more suitable starting point for optimization.

## 4.3. Hyperparameters tuning

Once the model Random Forest-Undersampling had been chosen, I proceeded to further improve its performance by tuning the hyperparameters. A comprehensive grid search with cross-validation was conducted to identify the optimal configuration of the model.
The hyperparameter search was carried out over the following grid:

- `n_estimators: [100, 200, 300]`
- `max_depth: [10, 20, 30]`
- `min_samples_split: [2, 5, 10]`
- `min_samples_leaf: [1, 2, 4]`

Stratified 5-fold cross-validation was employed to maintain consistent class distribution across folds. Two evaluation metrics were considered: **Recall**, to emphasize detection of the minority (malicious) class, and **ROC AUC**, to assess overall classification performance. The model was selected based on the highest **Recall** score.

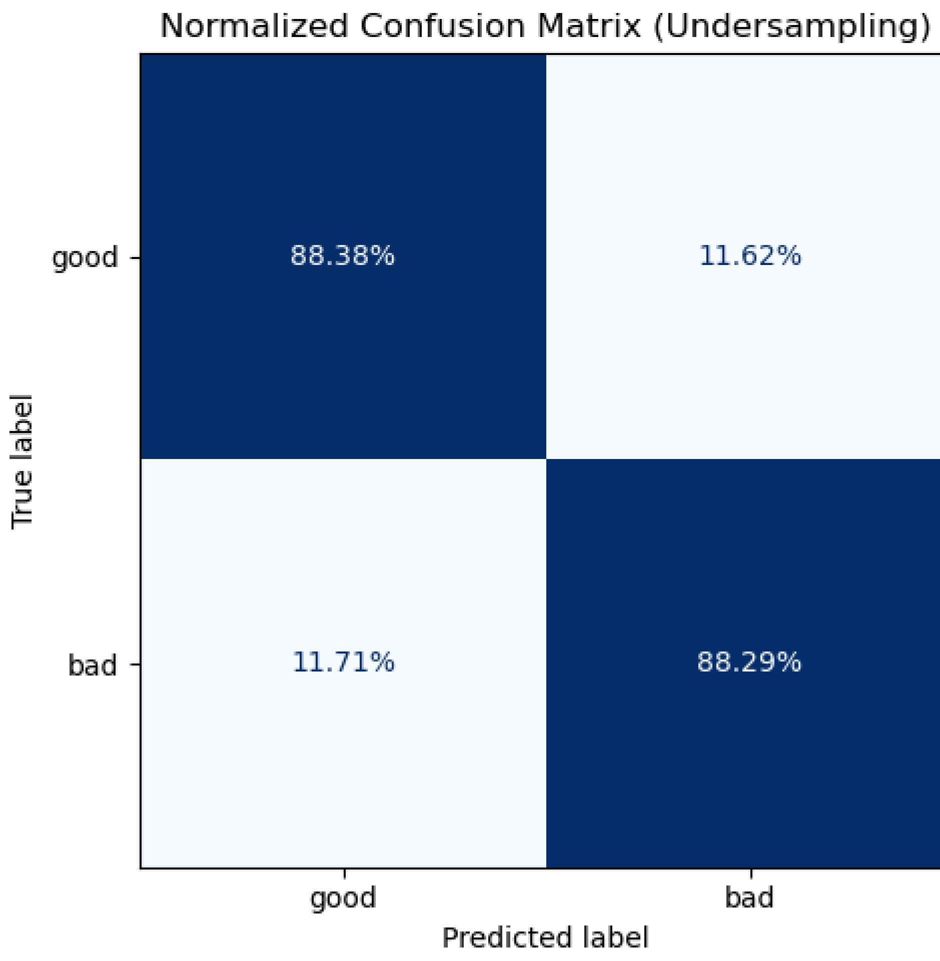
The optimal hyperparameters identified were:

- `n_estimators: 100`
- `max_depth: 30`
- `min_samples_split: 2`
- `min_samples_leaf: 2`

With this configuration, the model achieved a mean cross-validation recall of **0.8837** and a mean ROC AUC of **0.9562**. On the independent test set, the tuned model yielded a recall score of **0.8843**, demonstrating strong generalization in detecting malicious URLs.

The evaluation scores associated to the last model, later used for deployment, are:

| Accuracy | Precision | Recall | F1 Score | ROC-AUC |
|---|---|---|---|---|
| 0.8835 | 0.7732 | 0.8829 | 0.8244 | 0.9568 |



## 5. State of art comparison and further improvements

The results obtained during these trials have not been the best. Many reasons may have affected this decent yet underwhelming performance. For one, the scope limit of the features extracted. I limited my work to only URL-based features, obtained by simply analyzing the text of URLs. As stated by Hannousse and Yahiouche (2020)[8], most research on this topic considers a more varied nature of features, which they collectively divide into three main categories: URL-based features (which my work falls in), content-based features ( examination of the content of visited web pages) and external-based features (extracted by querying external services). Hannousse et al. have worked on each subset of features individually and with different algorithms, namely Decision Tree, Random Forest, Logistic Regression, SVM and Naïve Bayes. For what concerns the category I've worked on, they also did obtain the best results with Random Forest with an accuracy of 91.03% (vs 88.35% in this work) and F1-score of 91.00% (vs 82.44%). Thus, on another run, I would consider the addition of a more diverse set of features.

[8] Hannousse, A., & Yahiouche, S. (2021). Towards benchmark datasets for machine learning based website phishing detection: An experimental study. *Engineering Applications of Artificial Intelligence, 104*, 104347, , Sep. 2021, doi: 10.1016/j.engappai.2021.104347.

# 6. Software implementation

I decided to incorporate the model into a simple web application. The tools used have been: Python and Flask for the backend server, HTML and CSS for the frontend. Besides the model file (`model/rf_model.joblib`), this web tool leverages two other files:

- `app.py`
- `utils/feature_extraction.py`

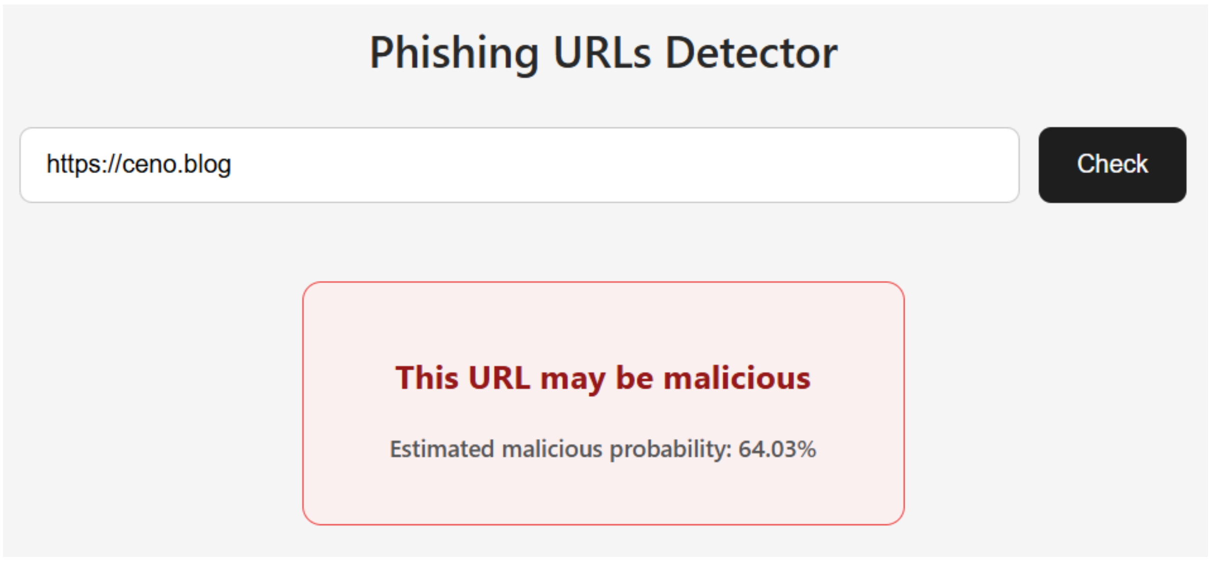
Once a request is sent to the server, *extract_features* is called on the submitted URL. The resultant dictionary is then converted into a DataFrame for prediction. The model then outputs the predicted class and the associated probability of the URL being malicious or legitimate. The result is then rendered on the web page for the user to view.

As for the UI, the User is expected to interact with the System in the following way:

1. User inserts a URL and clicks on the "Check" button.

2. System elaborates the URL and predicts whether it is malicious or not.

3. The result is returned to the User as a danger probability percentage:
    a. If the probability is higher (or equal) than 50%, the system shows a red box with the message "This URL may be malicious" along with the exact percentage (see Fig. 12)

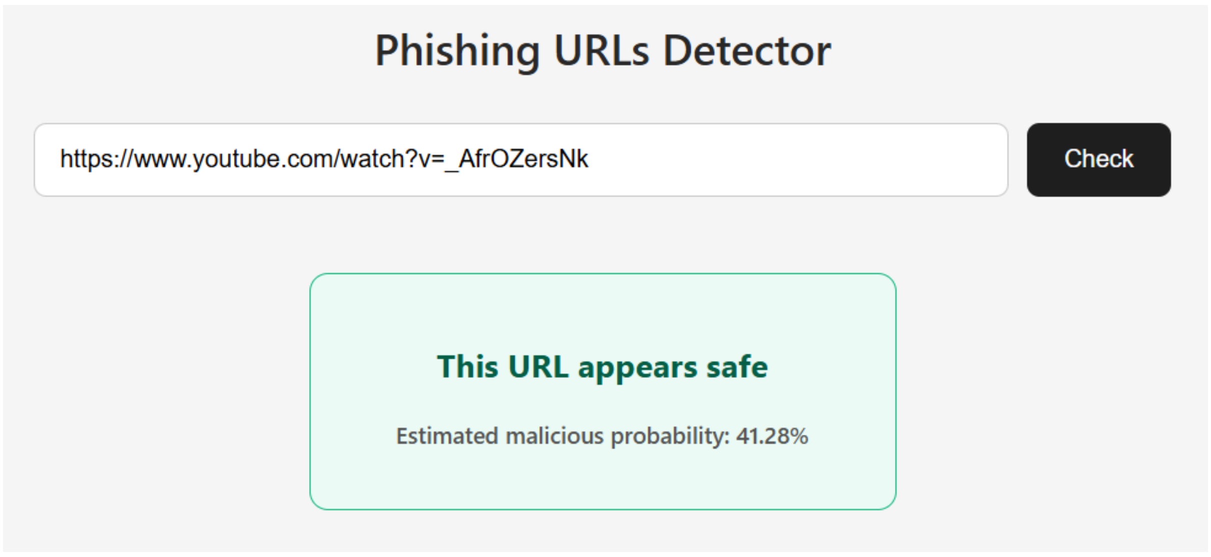

Phishing URLs Detector

https://ceno.blog

This URL may be malicious

Estimated malicious probability: 64.03%

    b. If it is lower, the system shows a green box with the message "This URL appears safe" along with the probability (see Fig.13)


Phishing URLs Detector

https://www.youtube.com/watch?v=_AfrOZersNk

This URL appears safe

Estimated malicious probability: 41.28%