URL to analyze:

sites.google.com/youwin/[redacted]

> Prediction for sites.google.com/...

> phishing

Candidates:

Riccardo Fantasia, Leonardo Pantani

What problem we want to solve

Our project was to analyze different classifiers to identify the best parameters and manipulation of data to obtain a model that is able to predict efficiently if a provided URL could be dangerous for the user.
Our classifier has been trained to be able to return four different outputs, based on features extracted by the URLs:

- **benign** → the URL is not malicious
- **defacement** → this website was subject to a defacement attack
- **phishing** → the URL leads to a website that tries to obtain users' info or access to their system by being similar to another website
- **malware** → the website is malicious or tries to download a malware

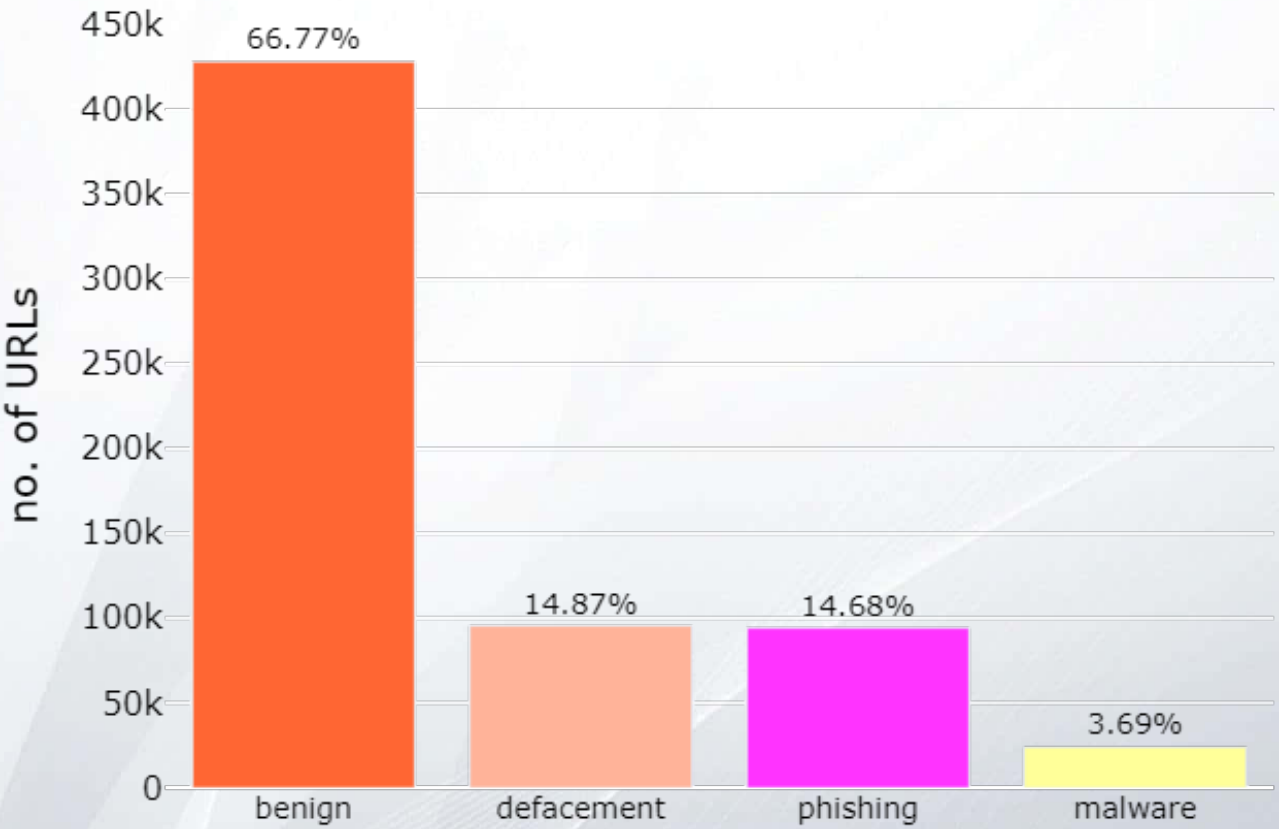
Different strategies were combined to achieve the best model, comparing each of them with f1-score.
Moreover, we assessed whether or not our dataset has a clustering tendency with specific metrics.

Origin of the dataset

The chosen dataset contains around 650,000 URLs compiled from five different sources: ISCX-URL-2016 [5], Malware domain blacklist, a GitHub repository [2], Phishtank [7] and PhishStorm [6].

The dataset has been downloaded from the Kaggle website [3].

Data Exploration - unbalanced dataset



From this visualization, we have a clear representation of how unbalanced the dataset is, thus indicating the need of a balancing approach. For our purposes, we tested two different balancing strategies.

Data Exploration - duplicate data

Given the dataset's size, it is plausible that duplicates exist. We identified around 1.5% of duplicates (approximately 10.000 elements). Because we have four possible classes, we sorted the dataset in this order:

malware > phishing > defacement > benign

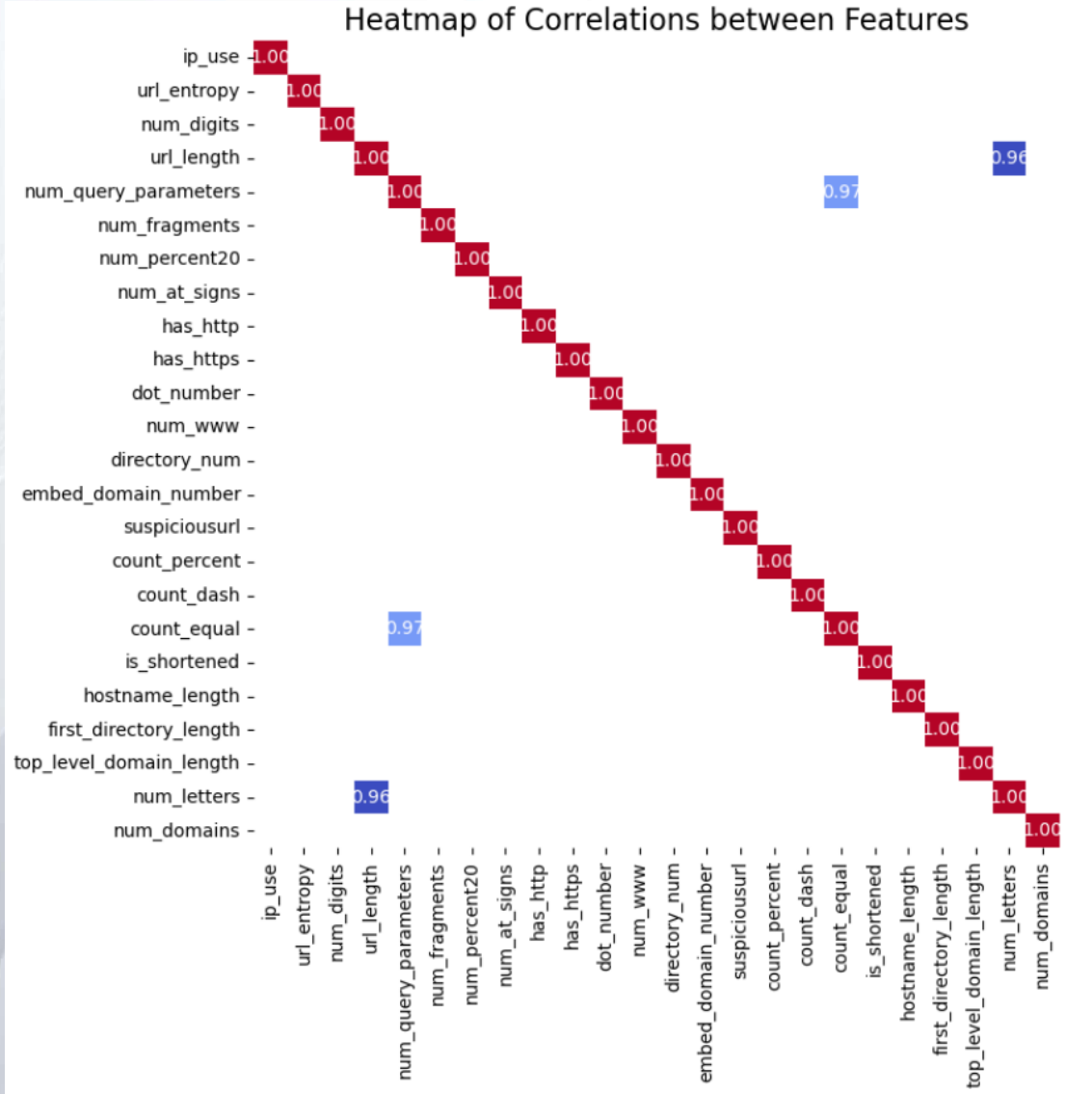
If a URL is labeled both malware and benign, we remove the benign label. This approach is quite aggressive, but we believe it is best for our use case, which is to protect users from potentially malicious sites. It is preferable to block a user by preventing access to a site that could be malicious rather than expose them to an additional risk.

Preprocessing

After cleaning our dataset, based on the found literature [1] [4] [8], we extracted dataset **24 features**, out of which:

- 19 are numerical:
  - `num_digits`, `url_entropy`, `url_length`, `num_query_parameters`, `num_fragments`, `num_percent20`, `num_at_signs`, `dot_number`, `num_www`, `directory_num`, `embed_domain_number`, `count_percent`, `count_dash`, `count_equal`, `hostname_length`, `first_directory_length`, `top_level_domain_length`, `num_letters`, `num_domains`
- and the 5 remaining are binary:
  - `ip_use`, `has_http`, `has_https`, `suspiciousurl`, `is_shortened`

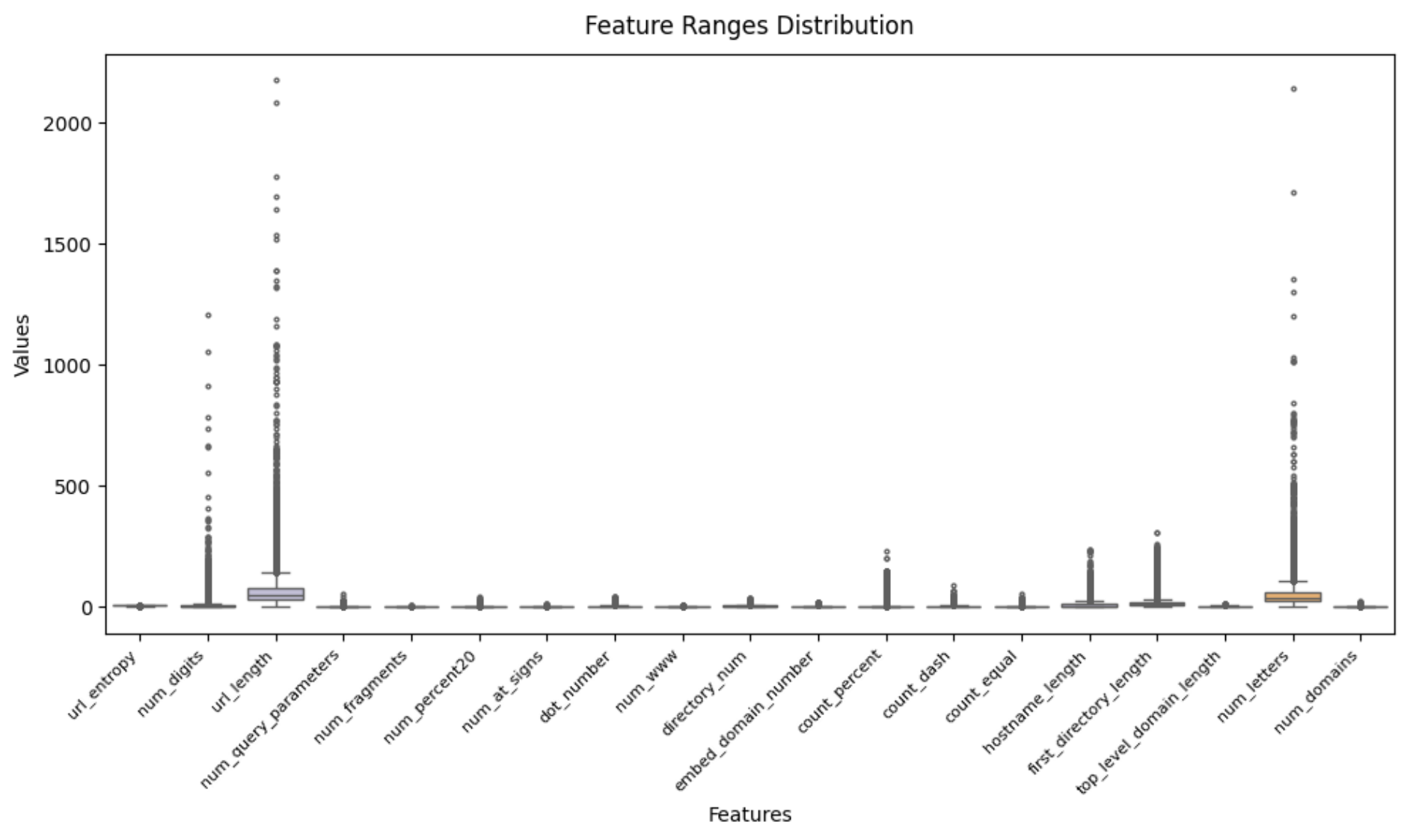
Preprocessing - Redundancy Handling



Using the HeatMap, we decided to remove two features that were strictly correlated to others, thanks to the Pearson's coefficient:

- `count_equal`
- `url_length`

Preprocessing - Redundancy Handling



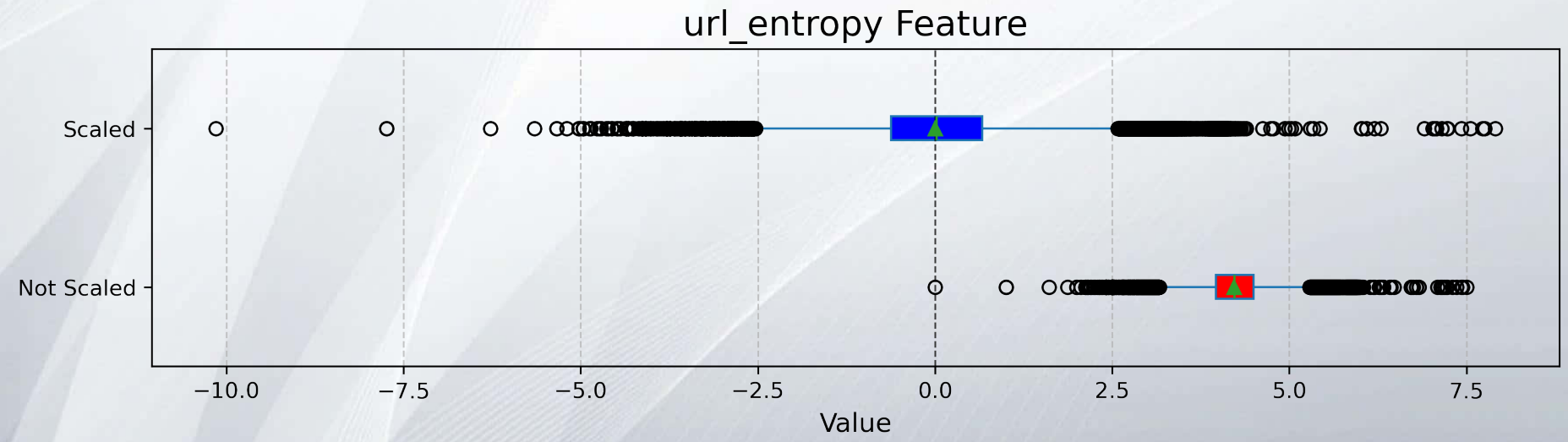
Preprocessing - Outliers and range of features

After splitting the dataset into a **training set** (80%) and a **test set** (20%) we tried to effectively remove the numerous outliers from the training set by using a fairly conservative threshold ($3 \times \texttt{IQR}$) given the nature of the data: the outliers are not the result of measurement errors.

Removing them too aggressively ($1.5 \times \texttt{IQR}$) would risk losing relevant examples for the model. The larger threshold maximizes the retention of potentially informative data while reducing excessive extreme values. Also, given the extreme difference in ranges between features we applied, for logistic regression, a Z-Score normalization.

Preprocessing - Standard Scaler

The Standard Scaler is suitable for logistic regression because it standardizes features by centering them at zero and scaling them to unit variance, ensuring that the model's optimization process converges efficiently and avoids bias due to varying feature magnitudes.



Why we choose Linear Regression

In addition to Random Forest, we also used logistic regression because it provides a more immediate model in terms of interpretability and allows us to evaluate the weight of each individual variable on the probability of malignancy. This is useful for understanding which features actually affect the risk of a URL. Logistic regression, being a linear model, is also easier to calibrate and explain from a statistical point of view. In our case, therefore, it serves as an interpretable and lighter baseline with which to compare the performance and impact of more complex techniques, measuring to what extent Random Forest adds value.

Goal of This Study:

- Try different combination of strategies for each of the two model and justify our results.

To recap: steps done

1. **Duplicate Removal:** Eliminated redundant samples.
2. **Feature Extraction:** Derived meaningful features.
3. **Redundancy Handling:** Removed two highly correlated features (correlation coefficient > 0.9)
4. **Data Splitting:** Partitioned the dataset into training (80%) and test (20%) sets to prevent information leakage during model evaluation.
5. **Outlier Management:** Removed extreme outliers to create a robust version of the dataset while retaining a copy with outliers for comparative analysis.
6. **Class Balancing:** Applied two strategies to address class imbalance and ensure fair model training.
7. **Classification**

To Recap: classifiers trained

Outliers	Balancing	Random Forest	Logistic Regression
No	Under-sampling only	✓	✓
No	Under and over -sampling	✓	✓
Yes	Under-sampling	✓	✓
Yes	Under and over -sampling	✓	✓

A comparison was conducted between two distinct classifiers that had been trained on four training sets. The outlier and balance combinations for these classifiers are presented in the summary table provided above.

F1-Score and Stratified K-Fold

Since our dataset is highly imbalanced, relying on accuracy can be deceptive, as a model that simply predicts the majority class may seem to perform well. The F1-Score, which balances precision and recall, provides a more reliable measure of Random Forest and Logistic Regression models that we trained, by penalizing both false positives and false negatives.

$$F1\text{-}Score = 2 \times \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

For the training phase, due to the unbalanced sample classes, we used StratifiedKFold that ensures that each fold maintains the overall class distribution of the dataset. Thanks to this method, we avoided misleading results that an uneven split could have caused.

Results for Random Forest

Outliers	Balancing Strategy 1	Balancing Strategy 2
Removed	Precision: 0.95 Recall: 0.95 F1 Score: 0.95	Precision: 0.95 Recall: 0.95 F1 Score: 0.95
Kept	Precision: 0.96 Recall: 0.95 F1 Score: 0.95	Precision: 0.96 Recall: 0.96 F1 Score: 0.96

Table: Precision, Recall, and F1 Scores for Random Forest with different combinations of outlier handling and balancing strategies.

Results for Logistic Regression

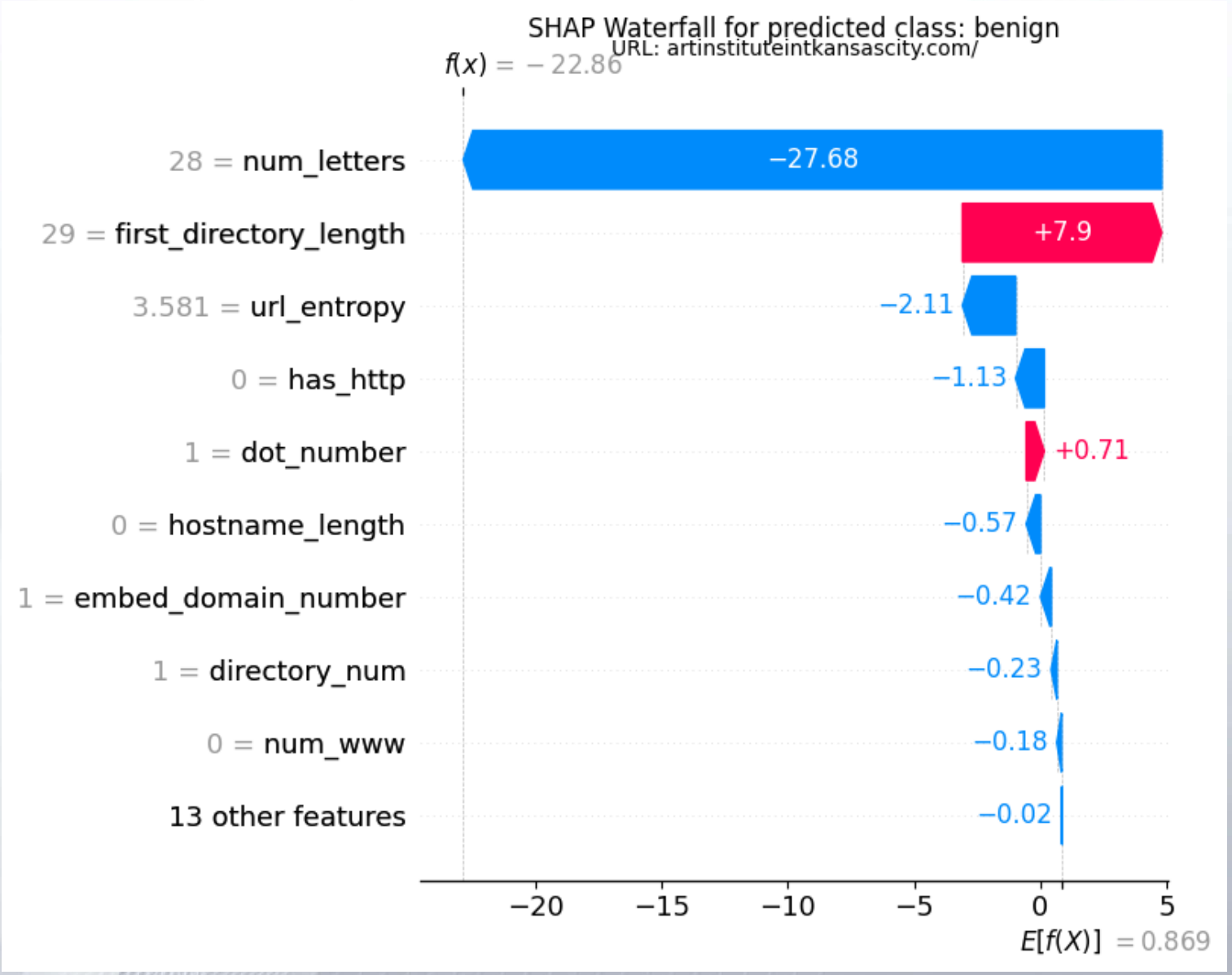
Outliers	Balancing Strategy 1	Balancing Strategy 2
Removed	Precision: 0.89 Recall: 0.89 F1 Score: 0.89	Precision: 0.89 Recall: 0.89 F1 Score: 0.89
Kept	Precision: 0.89 Recall: 0.88 F1 Score: 0.88	Precision: 0.89 Recall: 0.89 F1 Score: 0.89

Table: Precision, Recall, and F1 Scores for Logistic Regression with different combinations of outlier handling and balancing strategies.

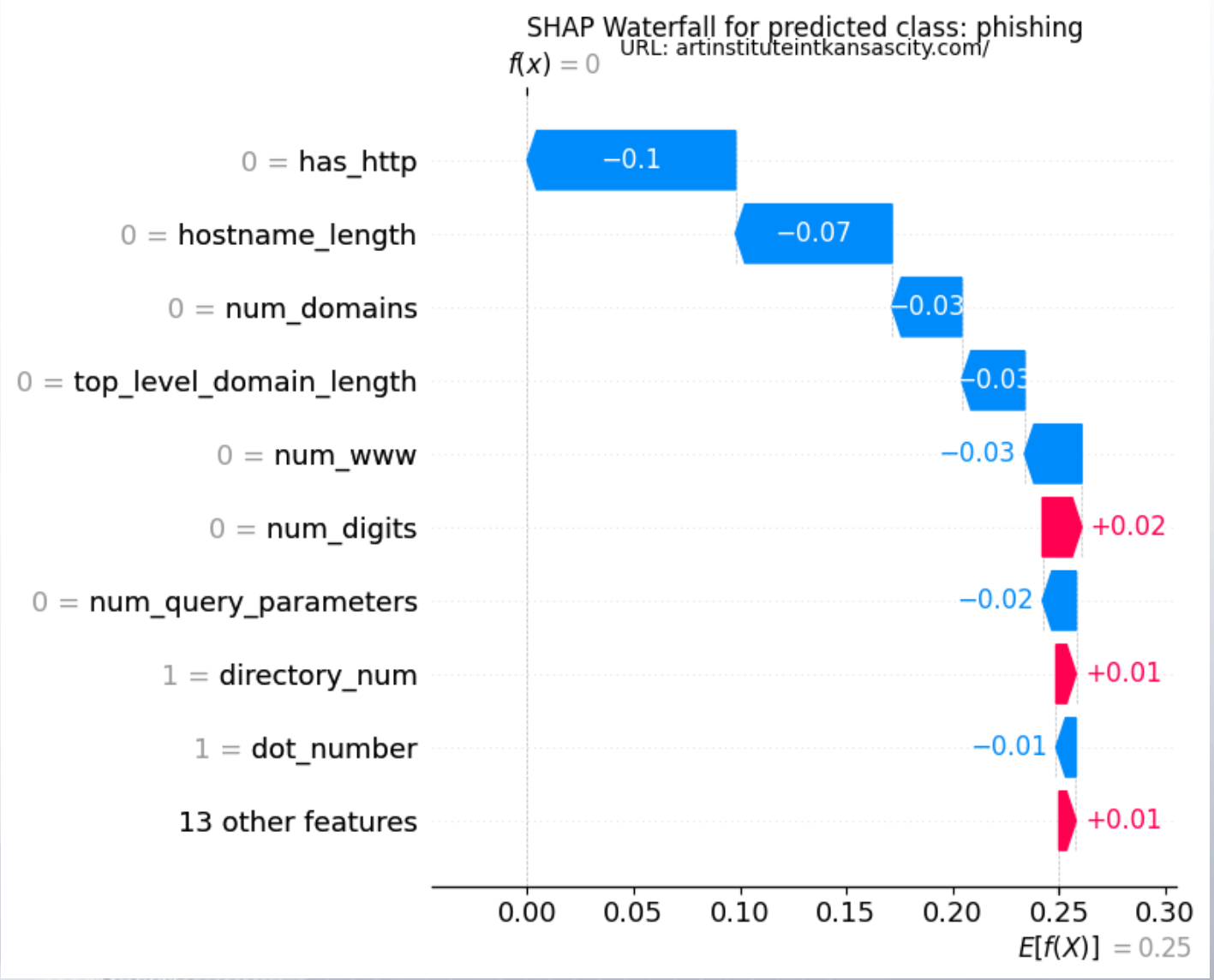
Insights

The **consistent performance** of both Random Forest and Logistic Regression across different data pre-processing techniques suggests a robust underlying data structure. Minor data modifications, such as *class balancing* or *moderate outlier removal*, do not **significantly impact performance**, indicating that core feature relationships drive predictions. Random Forest's inherent resilience to such variations stems from its ensemble methodology. For Logistic Regression, a well-defined separating hyperplane, likely dictated by strong feature signals, might remains stable despite these adjustments.
Essentially, the predictive power in both models derives from dominant data patterns resistant to minor perturbations, leading to consistent outcomes across data variants. Although overall performance may differ between the two algorithms, their shared stability highlights the inherent robustness of the learned decision functions.

Logistic Regression SHAP Values



Random Forest SHAP Values



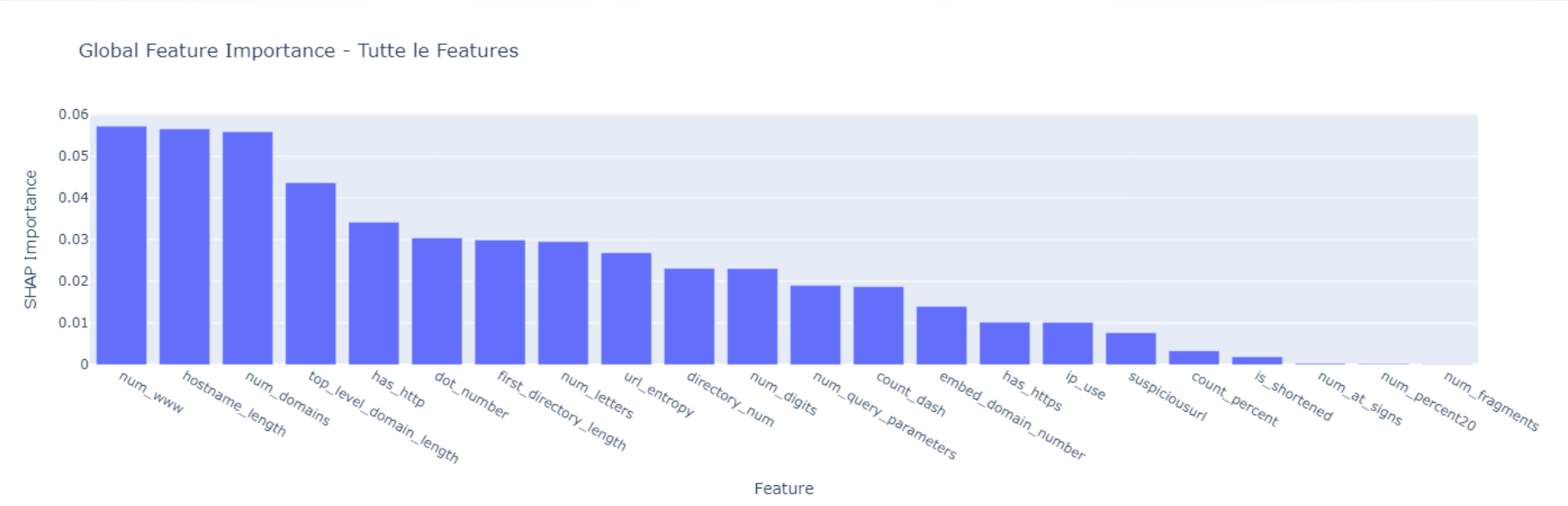
Comparative Analysis: Random Forest vs. Logistic Regression

SHAP analysis indicated that Random Forest effectively captured complex feature interactions, resulting in higher interpretability and improved performance metrics, especially for classes such as phishing and defacement.

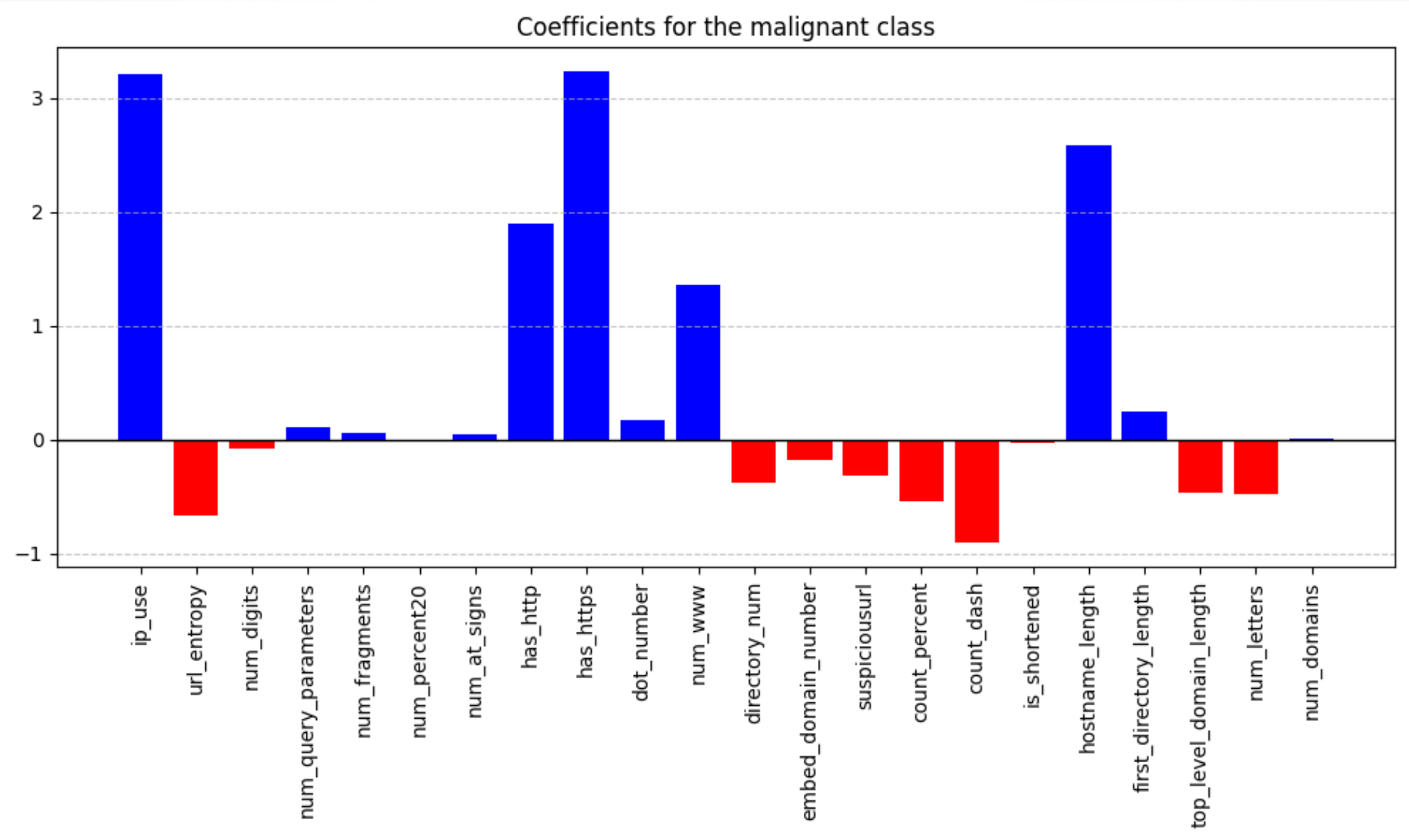
In contrast, Logistic Regression, because of its fixed linear contributions for individual features, has a very limited predicting power even with only two classes. Probably, the merging of malware, phishing, and defacement into a single malignant category created overlapping feature distributions that were difficult to separate with a simplified decision boundary.

The reduced flexibility of the Logistic Regression classifier lead to lower F1 scores, as demonstrated by the difficulty in modeling non-linear effects that Random Forest manages more effectively.

Random Forest Feature Importance



Logistic Regression Feature Importance



Feature Importance Analysis

Since for both models the features `is_shortened`, `num_at_signs`, `num_percent_20`, and `num_fragments` seemed irrelevant, we tested whether their removal impacted performance. We observed that:

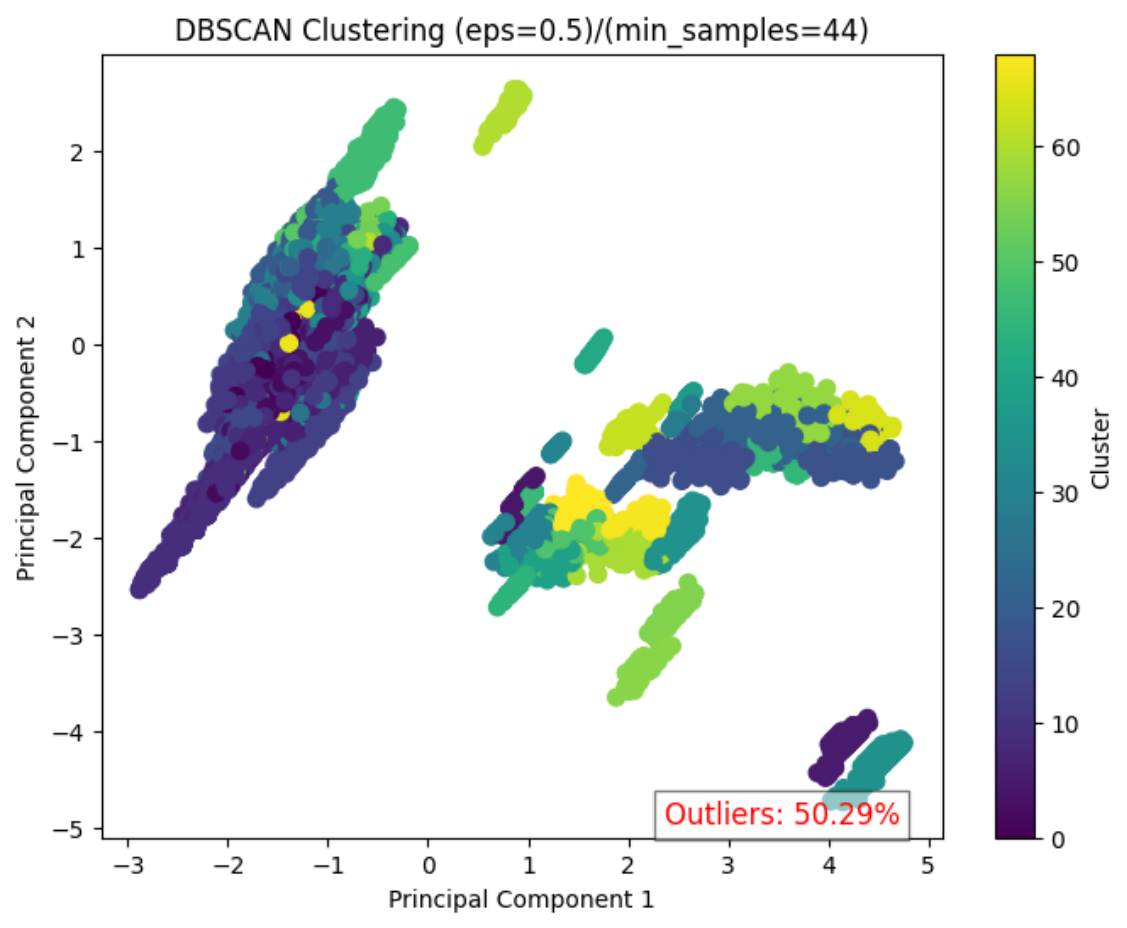
- No significant change in performance was observed.
- Computational time was slightly reduced after removing these features.

Model	F1 Score	Precision	Recall
Log Regression (with removed features)	0.89	0.89	0.89
Random Forest (with removed features)	0.96	0.96	0.96

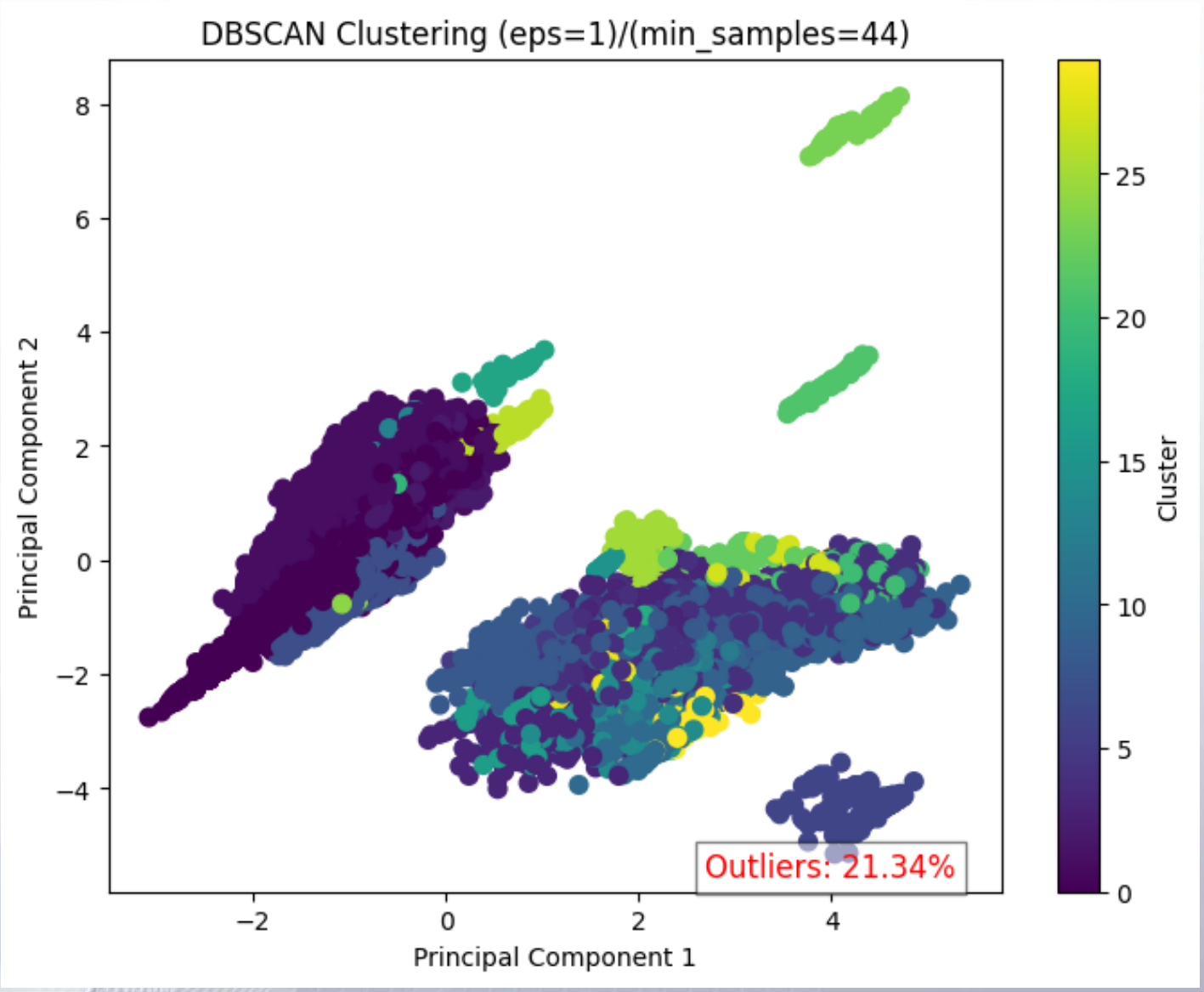
Table: Performance of Logistic Regression and Random Forest after removing irrelevant features.

Clustering

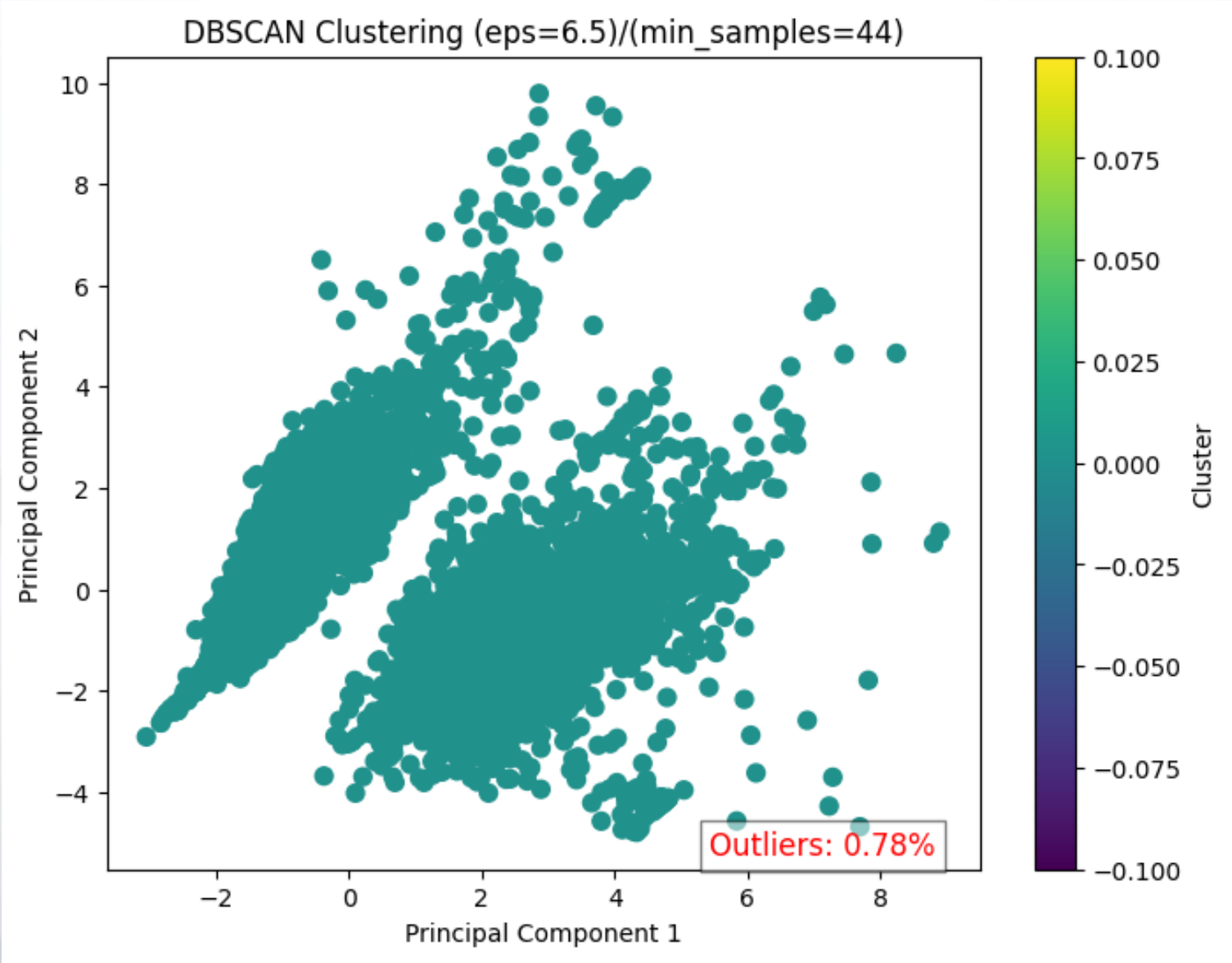
We then applied DBSCAN to test for possible clusters, but results were not really promising...



Clustering


DBSCAN Clustering (eps=1)/(min_samples=44)
Principal Component 2
Principal Component 1
Cluster
Outliers: 21.34%
8
6
4
2
0
−2
−4
−2
0
2
4
25
20
15
10
5
0

Clustering


DBSCAN Clustering (eps=6.5)/(min_samples=44)
Principal Component 2
Principal Component 1
Cluster
Outliers: 0.78%
10
8
6
4
2
0
−2
−4
−2
0
2
4
6
8
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100

Why did we get these results

Then we got our answers

```
import numpy as np
from pyclustertend import hopkins
from tqdm import tqdm

X_sampled_array = X_sampled.to_numpy()
sampling_size = int(0.1*len(X_sampled_array))
hopkins_score = hopkins(X_sampled_array, sampling_size)
n_trials = 10
hopkins_scores = []
print("\nComputing multiple Hopkins Statistics...")
for _ in tqdm(range(n_trials)):
    X_subset = X_sampled.sample(n=sampling_size, replace=False).to_numpy()
    hopkins_scores.append(hopkins(X_subset, sampling_size))
avg_hopkins = np.mean(hopkins_scores)
std_hopkins = np.std(hopkins_scores)
print(f"\nHopkins Statistic (media di {n_trials} prove): {avg_hopkins:.3f} ± {std_hopkins:.3f}")
```

```
Computing multiple Hopkins Statistics...
100%|██████████| 10/10 [11:54<00:00, 71.42s/it]

Hopkins Statistic (media di 10 prove): 0.004 ± 0.001
```

Next steps

Among all the classes, the malware class exhibited the lowest F1-score. This suggests that the selected features may not fully capture the distinguishing characteristics of this class, resulting in suboptimal performance compared to the other classes. Future research could focus on identifying additional or more relevant features to improve the classification accuracy for this particular class and address this performance gap.

Thank you!

https://github.com/LeonardoPantani/UNIPI-IA

Sources (1/2)

[1] Hoa Dinh Nguyen Cho Do Xuan and Tisenko Victor Nikolaevic. *Malicious URL Detection based on Machine Learning*. 2020. URL: http://dx.doi.org/10.14569/IJACSA.2020.0110119.

[2] *Github*. URL: https://github.com/faizann24/Using-machine-learning-to-detect-malicious-URLs/tree/master/data.

[3] *Kaggle*. URL: https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset/data.

[4] Cicchini Lorenzo. *Malware detection based on lexical features of domain names in DNS traffic*. 2019. URL: https://tesi.univpm.it/retrieve/ffa888b6-1ff2-468e-9aa7-88f1102d0a01/Tesi_pdfA_CicchiniLorenzo.pdf.

Sources (2/2)

[5] Mohammad Saiful Islam Mamun et al. *Detecting Malicious URLs Using Lexical Analysis", Network and System Security, Springer International Publishing, pp. 467-482, 2016.* 2016. URL: https://www.unb.ca/cic/datasets/url-2016.html (visited on 01/21/2025).

[6] *PhishStorm*. URL: https://research.aalto.fi/en/datasets/phishstorm-phishing-legitimate-url-dataset.

[7] *Phishtank*. URL: https://www.phishtank.com/developer_info.php.

[8] Abad S, Gholamy H, and Aslani M. *Classification of Malicious URLs Using Machine Learning*. 2023. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC10537824/#sec3-sensors-23-07760.