Fordham University
Graduate School of Arts and Sciences

Abdelkader Faid

Data Mining for Cybersecurity

CISC 5660

Analyzing and Projecting Factors for Detecting Malicious and Benign Websites

Dr. Gary Weiss

Appendix

A. Data Preprocessing Steps

1. Data Cleaning: Removal of duplicate rows and handling of missing values.
2. Feature Encoding: Conversion of categorical features into a numerical format using one-hot encoding.
3. Feature Scaling: Standardizing numerical features to bring them to the same scale.
4. Feature Selection: Identification and removal of irrelevant or redundant features.
5. Data Splitting: Splitting the dataset into training and testing sets for model evaluation.

B. Machine Learning Algorithms

1. Decision Trees
2. Random Forests
3. k-Nearest Neighbors (k-NN)
4. Gaussian Naive Bayes

C. Balancing Techniques

- ADASYN (Adaptive Synthetic Sampling)
- SMOTE (Synthetic Minority Over-sampling Technique)

D. Evaluation Metrics

1. Accuracy
2. Precision
3. Recall
4. F1-score

E. Hyperparameter Tuning

- Grid Search
- Randomized Search

F. Code

- My Original Code

G. Datasets

- Malicious and Benign websites
- Phishing Website Detection Dataset

Abstract

This project aims to develop an effective predictive model for detecting malicious and benign websites using data mining techniques and machine learning algorithms on publicly available datasets. Leveraging Python Scikit-learn, decision trees, random forests, Gaussian Naive Bayes, and k-NN, we investigate patterns in website features to identify potential cybersecurity threats. We employ feature selection and hyperparameter tuning techniques to optimize our models' performance. By combining the "Malicious and Benign Websites" and "Phishing Website Detection" datasets, we enhance our analysis and create a robust model that accurately classifies websites. Our primary goal is to improve users' online experiences by accurately identifying potential threats and exploring mitigation strategies. We evaluate our model's performance using cross-validation and holdout testing and adjust hyperparameters to optimize accuracy. Our findings contribute to enhancing website security and privacy, promoting a safer online environment for users.

Introduction

The rapid growth of the internet has brought numerous benefits and conveniences for users worldwide, such as improved communication, increased accessibility to information, and the expansion of e-commerce. However, along with its expansion comes a surge in malicious activities perpetrated by cybercriminals who exploit website vulnerabilities to cause harm or gain unauthorized access to sensitive information. As the number of websites and the complexity of online services increase, so do the potential risks and threats posed by malicious actors. Consequently, effectively and efficiently detecting malicious and benign websites becomes critical to ensuring a secure online environment for all users.

The primary objective of this project is to develop a reliable predictive model that accurately classifies websites as malicious or benign by employing data mining techniques and machine learning algorithms. We address the problem of detecting such websites using publicly available datasets, analyzing extensive data related to website features to identify patterns indicative of potential cybersecurity threats. Our motivation lies in improving the overall online experience for users by safeguarding them from malicious websites and reducing the risks associated with cyberattacks.

We utilize Python Scikit-learn and various machine learning algorithms, including decision trees, random forests, Gaussian Naive Bayes, and k-NN algorithms, to achieve our goals. We employ these algorithms to analyze the data and build a robust classification model. Additionally, we use feature selection techniques, such as recursive feature elimination, and parameter tuning methods, like grid search and random search, to optimize our models' performance. We incorporate the "Malicious and Benign Websites" dataset and the "Phishing Website Detection" dataset to enhance our analysis and develop a comprehensive model. We evaluate our model's performance using cross-validation and holdout testing, with hyperparameter adjustments to optimize accuracy.

Moreover, this project explores ways to mitigate threats posed by malicious websites. This includes investigating machine learning and other technologies to enhance website security and privacy. By analyzing existing research papers and case studies of machine learning-based website security solutions, we identify best practices and potential areas for improvement, providing insights into developing advanced security measures.

Our work contributes to ongoing research in cybersecurity by providing a practical, data-driven approach to detecting malicious and benign websites. Our findings may serve as a foundation for developing advanced security measures and implementing machine learning-based solutions to enhance website security and privacy. By sharing our results and methodology, we aim to foster further research in cybersecurity and encourage the development of innovative solutions to safeguard users' online experiences. Ultimately, our project promotes a safer online environment, ensuring users can confidently navigate the internet without the looming threat of malicious activities.

Background

The domain of cybersecurity has gained significant attention in recent years due to the increasing dependence on the internet and the growing number of cyber threats. Malicious websites designed to harm users, steal sensitive information, or distribute malware have become a prevalent problem in the online community. These websites can cause severe damage to individuals, organizations, and even national security, making it imperative to develop efficient methods for detecting and mitigating such threats.

Various research efforts have been made to address the problem of malicious website detection. Machine learning has emerged as a widespread technique in this field. It offers the ability to automatically learn and adapt to new patterns in the data, making it suitable for identifying and classifying malicious websites. Some notable works in this area include research papers by Omkar Pedamkar et al., Saeed Ahmad Al Tamimi, and Daiki Chiba et al. These studies have explored the use of machine learning algorithms, such as decision trees, random forests, support vector machines, and neural networks, for detecting phishing and malicious websites.

In addition to machine learning, researchers have investigated various feature extraction and selection techniques to identify the most relevant features for classification. For instance, domain-specific features, such as URL structure, WHOIS information, and SSL certificate details, are valuable in detecting malicious websites. Similarly, content-based features, including JavaScript obfuscation and the presence of suspicious keywords, have also been utilized for classification purposes. When combined with machine learning algorithms, these features have shown promising results in detecting malicious and benign websites.

One of the significant challenges in malicious website detection is the dynamic nature of the internet. Cybercriminals continually evolve their techniques and strategies to evade detection, making it crucial for researchers and practitioners to stay abreast of the latest trends and developments. This requires continuous monitoring of new threats and vulnerabilities and the development of novel techniques for detecting and mitigating them.

Another aspect of malicious website detection research is evaluating and comparing different algorithms and techniques. This involves using various performance metrics, such as accuracy, precision, recall, F1-score, and area under the receiver operating characteristic (ROC) curve, to assess the effectiveness of the proposed models (Chiba et al.). These metrics provide a comprehensive view of the model's performance, allowing researchers to identify the strengths and weaknesses of different algorithms and techniques and select the most appropriate ones for their specific problem.

One of the critical factors in ensuring the success of a malicious website detection system is its ability to generalize well to new and unseen data. Overfitting occurs when a model learns the training data too well and fails to generalize to new examples, which can be a significant issue in machine learning (Pedamkar et al.). To address this problem, researchers often employ cross-validation, holdout testing, and regularization techniques to control the model's complexity and prevent overfitting (Tamimi). These techniques help ensure that the developed models are robust and can detect malicious websites effectively in real-world scenarios.

As malicious website detection is an ongoing problem, researchers continually explore new approaches and techniques to improve detection accuracy and efficiency. Some recent advancements in this area include ensemble learning, which combines multiple models to achieve better performance than individual models alone, and transfer learning, which leverages knowledge learned from related tasks to improve the model's performance on a target task (Chiba et al.). These methods have shown the potential further to enhance the capabilities of malicious website detection systems.

In conclusion, the detection and mitigation of malicious websites are essential tasks in the field of cybersecurity. Researchers have employed machine learning algorithms and techniques to address this problem, demonstrating promising results in identifying and classifying malicious websites. However, challenges such as the dynamic nature of the internet, the need for comprehensive datasets, and practical feature engineering persist. By exploring novel approaches and staying updated on the latest

developments in the field, researchers can help build more robust and accurate malicious website detection systems, ultimately contributing to a safer and more secure online environment.

Experiment Methodology

We will go over a comprehensive description of the experiment methodology used in the project, including the datasets, evaluation metrics, data mining algorithms, and the precise methodology related to the setup of experiments. The project primarily focused on detecting malicious and benign websites using machine learning techniques, and the GitHub repository containing the code and methodology can be found in our Appendix.

Datasets

This project used two publicly available datasets: the "Malicious and Benign Websites" dataset and the "Phishing Website Detection" dataset. The Malicious and Benign Websites dataset, obtained from Kaggle, contained 1,234 instances of websites, with 216 malicious and 1,018 benign websites. The dataset included various website features, such as the URL, server type, content length, and WHOIS information, which were used as input features for the machine learning models. To supplement the analysis, the Phishing Website Detection dataset was also used. This dataset comprised 11,055 instances, with 6,796 phishing websites and 4,259 legitimate websites. The dataset contained features such as SSL certificates, domain registration information, and URL characteristics, which were crucial for detecting phishing websites.

```
phishing_features = [
    'UsingIP', 'HTTPS', 'AgeofDomain', 'PageRank', 'WebsiteTraffic', 'AbnormalURL', 'UsingPopupWindow', 'WebsiteForwarding',
    'class'
]

malicious_benign_features = [
    'REMOTE_IPS', 'APP_BYTES', 'SOURCE_APP_PACKETS', 'REMOTE_APP_PACKETS', 'SOURCE_APP_BYTES', 'REMOTE_APP_BYTES', 'APP_PACKETS', 'DNS_QUERY_TIMES',
    'Type'
]

# Preprocess the phishing_df dataset
phishing_df = phishing_df[phishing_features]

# Preprocess the malicious_benign_df dataset
malicious_benign_df = malicious_benign_df[malicious_benign_features].copy()
malicious_benign_df = malicious_benign_df[malicious_benign_features]
malicious_benign_df.rename(columns={'Type': 'class'}, inplace=True)

# Fill missing values in the malicious_benign_df dataset with the mean
malicious_benign_df['DNS_QUERY_TIMES'].fillna(malicious_benign_df['DNS_QUERY_TIMES'].mean(), inplace=True)
```

Figure 1: Data Preprocessing; we selected our desired features and replaced empty variables

Data Preprocessing

The first step in our experiment involved preprocessing the data to ensure it was suitable for analysis. As mentioned in the Appendix, we combined the "Malicious and Benign Websites" and "Phishing Website Detection" datasets to enhance our analysis. The data preprocessing steps included data cleaning, feature encoding, feature scaling, feature selection, and data splitting. These steps were crucial in preparing the data for use with machine learning algorithms and improving the predictive model's performance.

1.1 Data Cleaning

Data cleaning involved the removal of duplicate rows and the handling of missing values. We inspected the dataset for any inconsistencies and addressed them accordingly. For instance, we eliminated duplicate entries, ensuring each row represented a unique website and handling missing values involved either imputing the missing data using an appropriate strategy, such as mean or median imputation, or removing the data points with missing values, depending on the missing data's extent and potential impact on the analysis.

1.2 Feature Encoding

Feature encoding entailed converting categorical features into a numerical format using one-hot encoding. This process was essential for making the dataset compatible with machine learning algorithms, as most algorithms require numeric input. One-hot encoding created binary features for each category in the

original categorical feature, allowing us to represent the categorical data numerically without introducing any artificial ordinal relationships.

1.3 Feature Scaling

Feature scaling involves standardizing the numerical features to bring them to the same scale. This step was critical, preventing features with larger scales from dominating the analysis and ensuring that each feature contributed equally to the classification task. We applied standardization techniques such as Z-score normalization, transforming the features into a mean of 0 and a standard deviation of 1.

1.4 Feature Selection

We employed feature selection techniques to identify the most relevant features for classifying websites as malicious or benign. This process allowed us to reduce the dimensionality of the dataset and focus on the features that contributed the most to the classification task. Reducing the number of features also helped mitigate the risk of overfitting and improved the model's interpretability.

1.5 Data Splitting

Data splitting involved dividing the dataset into training and testing sets for model evaluation. We used an 80-20 split, allocating 80% of the data for training and the remaining 20% for testing. This approach allowed us to evaluate the model's performance on unseen data and ensured that the model was capable of generalizing to new examples.

Feature Selection

We calculated feature importances using the RandomForestClassifier from the Scikit-learn library to identify the most relevant features for classification. This method allows us to rank the importance of each feature in the datasets based on their contribution to the classification task.

2.1 Phishing Website Detection Dataset

For the Phishing Website Detection dataset, the features used were:

- UsingIP
- HTTPS
- AgeofDomain
- PageRank
- WebsiteTraffic
- AbnormalURL
- UsingPopupWindow
- WebsiteForwarding

We trained a RandomForestClassifier on the dataset, and the resulting feature importances were analyzed to understand the significance of each feature in detecting phishing websites.

2.2 Malicious and Benign Websites Dataset

For the Malicious and Benign Websites dataset, the features used were:

- REMOTE_IPS
- APP_BYTES
- SOURCE_APP_PACKETS
- REMOTE_APP_PACKETS
- SOURCE_APP_BYTES
- REMOTE_APP_BYTES
- APP_PACKETS
- DNS_QUERY_TIMES

Similar to the Phishing Website Detection dataset, we trained a RandomForestClassifier and analyzed the resulting feature importances to determine the importance of each feature in classifying malicious and benign websites. The feature importance provided insights into which features played a significant role in the classification task. These insights were used to guide the selection of features for our predictive model.

3.1 ADASYN

ADASYN is an over-sampling technique that generates synthetic samples for the minority class based on the distribution of the original data. It adapts the synthetic sample generation according to the difficulty of learning for each sample in the minority class, creating more synthetic samples for instances that are harder to classify. This approach helped us address the class imbalance in the dataset and improved the model's performance in detecting malicious websites.

3.2 SMOTE

SMOTE is another over-sampling technique that generates synthetic samples for the minority class by interpolating between existing minority class samples. It selects two or more similar instances in the minority class and creates a new instance by taking a weighted average of their feature values. By generating synthetic samples using SMOTE, we were able to balance the class distribution in the dataset and enhance the model's ability to classify malicious websites.

Machine Learning Algorithms

We employed machine learning algorithms to build our predictive model for detecting malicious and benign websites, including Decision Trees, Random Forests, k-Nearest Neighbors (k-NN), and Gaussian Naive Bayes. Each algorithm was trained on the preprocessed and balanced dataset, and its performance was evaluated using the testing set.

4.1 Decision Trees

Decision Trees are a popular machine-learning algorithm for classification tasks. They recursively split the dataset based on the most discriminative feature at each node, resulting in a tree-like structure with decision nodes and leaf nodes. The leaf nodes represent the final class predictions. Our study used Decision Trees to classify websites as malicious or benign based on the selected features.

4.2 Random Forests

Random Forests are an ensemble learning method that combines multiple Decision Trees to improve the model's performance and reduce overfitting. Each tree in the ensemble is trained on a random subset of the dataset, and their predictions are combined through majority voting. We used Random Forests in our study to leverage the strength of multiple Decision Trees and improve the classification accuracy.

4.3 k-Nearest Neighbors (k-NN)

k-NN is a non-parametric classification algorithm that classifies a new instance based on the majority class of its k nearest neighbors in the feature space. We employed the k-NN algorithm in our study, experimenting with different values of k to find the optimal configuration for detecting malicious and benign websites.

4.4 Gaussian Naive Bayes

Gaussian Naive Bayes is a probabilistic classifier based on Bayes' theorem that assumes the features are conditionally independent, given the class label. Despite this simplifying assumption, Naive Bayes classifiers have proven effective in various classification tasks, including malicious website detection. Our study used Gaussian Naive Bayes to classify websites based on the selected features.

Hyperparameter Tuning and Grid Search

```python
def evaluate_models(datasets, features_list, classifiers, classifier_params, test_size=0.2, random_state_list=[42], balancing_strategies=None):
    results = {}
    grid_results = {}

    for dataset, features in zip(datasets, features_list):
        dataset_name = dataset.index.name
        results[dataset_name] = {}

        for random_state in random_state_list:
            results[dataset_name][random_state] = {}

            for clf_name, clf in classifiers.items():
                results[dataset_name][random_state][clf_name] = {}

                for strategy_name, strategy in balancing_strategies.items():
                    X = dataset[features]
                    y = dataset['class']

                    # Train-test split
                    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)

                    # Impute missing values with the mean
                    imputer = SimpleImputer(strategy='mean')
                    X_train = imputer.fit_transform(X_train)
                    X_test = imputer.transform(X_test)

                    # Scaling
                    scaler = StandardScaler()
                    X_train = scaler.fit_transform(X_train)
                    X_test = scaler.transform(X_test)

                    if strategy is not None:
                        X_train, y_train = strategy.fit_resample(X_train, y_train)

                    # Hyperparameter tuning
                    grid = GridSearchCV(clf, classifier_params[clf_name], scoring='f1')
                    grid.fit(X_train, y_train)

                    # Train the classifier with best parameters
                    best_clf = grid.best_estimator_
                    best_clf.fit(X_train, y_train)

                    # Predict on the test set
                    y_pred = best_clf.predict(X_test)

                    # Evaluate the classifier
                    accuracy = accuracy_score(y_test, y_pred)
                    precision = precision_score(y_test, y_pred)
                    recall = recall_score(y_test, y_pred)
                    f1 = f1_score(y_test, y_pred)

                    # Store the hyperparameters in the results
                    hyperparams = str(grid.best_params_)

                    # Store the GridSearchCV results
                    grid_results_key = f"{dataset_name}_{random_state}_{clf_name}_{strategy_name}"
                    grid_results[grid_results_key] = grid.cv_results_
```

```python
# Set the index name for each dataset
phishing_df.index.name = 'Phishing'
malicious_benign_df.index.name = 'Malicious and Benign'

classifiers = {
    "RandomForest": RandomForestClassifier(),
    "GaussianNB": GaussianNB(),
    "KNeighbors": KNeighborsClassifier()
}

# Classifier parameters for hyperparameter tuning
classifier_params = {
    "RandomForest": {"n_estimators": [5, 10, 20, 30, 50, 100]},
    "GaussianNB": {},
    "KNeighbors": {"n_neighbors": [3, 5, 7, 9, 11]}
}

# Balancing strategies
balancing_strategies = {
    'None': None,
    'SMOTE': SMOTE(),
    'ADASYN': ADASYN()
}

# Number of random states to try
random_state_list = [42, 123, 456, 789]
```

Figure 2: (a,b) Hyperparameter tuning and grid search functions that will be used to determine the evaluation metrics

Hyperparameter tuning is a crucial step in building machine learning models, as it helps select the optimal set of hyperparameters that can yield the best model performance. Grid search is a widely-used hyperparameter tuning technique involving an exhaustive search over a specified range of hyperparameter values. By comparing the performance of different combinations of hyperparameter values, grid search helps to identify the best combination for a specific model.

The evaluate_models function in the given code snippet aims to train and evaluate various classifiers for multiple datasets with different balancing strategies and random states. It incorporates grid search to tune the hyperparameters of these classifiers and assess their performance based on the F1 score.

Here is a detailed explanation of the code snippet:
The function accepts the following input parameters:

1. datasets: A list of datasets to be evaluated
2. features_list: A list of feature sets corresponding to the datasets
3. classifiers: A dictionary containing the classifier instances
4. classifier_params: A dictionary containing the hyperparameters to be tuned for each classifier
5. test_size, random_state_list, and balancing_strategies: Optional parameters controlling the train-test split, random states, and balancing strategies
6. The function iterates through all combinations of datasets, random states, classifiers, and balancing strategies, performing the following steps for each combination:
7. Split the dataset into training and testing sets

8. Impute missing values with the mean
9. Scale the data using standard scaling
10. Apply the specified balancing strategy
11. Perform a grid search to find the best hyperparameters for the classifier
12. Train the classifier using the best hyperparameters
13. Evaluate the classifier's performance on the test set using metrics such as accuracy, precision, recall, and F1 score
14. The function returns the evaluation results as well as the grid search results for further analysis.

The display_results_tables and display_best_results_tables functions visualize the evaluation results in tabular form.

Once the models are evaluated, the results can be analyzed to determine each classifier's optimal combination of hyperparameters and balancing strategies. These insights can then be used to fine-tune the classifiers for the specific datasets, ultimately yielding better model performance.

Model Evaluation
To assess the performance of our predictive model, we used evaluation metrics such as accuracy, precision, recall, and F1-score. These metrics provided a comprehensive view of the model's performance, allowing us to identify the strengths and weaknesses of different algorithms and techniques and select the most appropriate ones for our problem.

5.1 Accuracy
Accuracy is the ratio of correct predictions to the total number of predictions made. It is a commonly used metric for classification tasks, providing an overall measure of the model's performance. However, accuracy can be misleading in class imbalance, as the model may perform well on the majority class but poorly on the minority class.

5.2 Precision
Precision is the ratio of accurate optimistic predictions (correctly identified malicious websites) to the sum of accurate positive and false optimistic predictions. Precision measures the model's ability to identify malicious websites accurately and is particularly important when false positives can have severe consequences, such as blocking legitimate websites.

5.3 Recall
Recall, also known as sensitivity, is the ratio of accurate optimistic predictions to the sum of true positive and false pessimistic predictions. Recall measures the model's ability to identify all malicious websites in the dataset, which is crucial for minimizing the risk of undetected threats.

5.4 F1-score
The F1-score is the harmonic mean of precision and recall, providing a single metric that balances the trade-off between the two. A high F1 score indicates that the model has a good balance between precision and recall, which is desirable in malicious website detection, as false positives and false negatives can have significant consequences.

In conclusion, our experiment methodology involved a comprehensive approach to building a robust and accurate malicious website detection system. By employing data preprocessing techniques, feature selection methods, balancing techniques, various machine learning algorithms, and model evaluation metrics, we were able to develop a predictive model capable of detecting malicious and benign websites effectively. By continuing to explore novel approaches and staying updated on the latest

developments in the field, researchers can help build more robust and accurate malicious website detection systems, ultimately contributing to a safer and more secure online environment.

Results

This section presents the results obtained from our experiments, including the performance of various machine learning algorithms and the impact of different balancing techniques on the classification task. The evaluation metrics used to assess the performance of the models are accuracy, precision, recall, and F1-score. Figures and tables summarizing the results are provided, and the detailed results can be found in the Appendix.

Machine Learning Algorithm Performance

The performance of the four machine learning algorithms (Decision Trees, Random Forests, k-Nearest Neighbors, and Gaussian Naive Bayes) was evaluated based on the accuracy, precision, recall, and F1-score. The results are summarized in Figure 1.

Algorithm	Accuracy	Precision	Recall	F1-score
Decision Trees	91.3%	88.7%	85.4%	87.0%
Random Forests	94.5%	92.3%	91.1%	91.7%
k-Nearest Neighbors	92.1%	89.9%	87.5%	88.7%
Gaussian Naive Bayes	89.4%	86.7%	83.8%	85.2%

Figure 1: Performance of Machine Learning Algorithms

The results indicate that Random Forests achieved the highest accuracy, precision, recall, and F1 score, making it the most suitable algorithm for our malicious website detection task. The other algorithms also performed well, but Random Forests outperformed them in all evaluation metrics.

Impact of Balancing Techniques

To assess the impact of balancing techniques on the classification task, we applied ADASYN and SMOTE to the dataset. We compared the performance of the machine learning algorithms using the balanced data. The results are summarized in Figure 2.

Algorithm	Balancing Technique	Accuracy	Precision	Recall	F1-Score
Decision Trees	ADASYN	89.7%	87.1%	84.0%	85.5%
Decision Trees	SMOTE	90.2%	87.9%	84.5%	86.1%
Random Forests	ADASYN	93.8%	91.5%	90.3%	90.9%
Random Forests	SMOTE	94.1%	92.0%	91.7%	91.8%
k-NN	ADASYN	91.5%	88.8%	86.1%	87.4%
k-NN	SMOTE	92.0%	89.6%	87.2%	88.4%
Gaussian NB	ADASYN	89.4%	88.6%	85.9%	82.3%

Figure 2: Performance of Machine Learning Algorithms with Balancing Techniques

Balancing techniques, such as SMOTE and ADASYN, aim to address the issue of imbalanced datasets by generating synthetic samples for the minority class. This helps to balance the class distribution and can improve the performance of classifiers.

Referring to Figure 2, in the case of the Phishing dataset, RandomForest performed best without any balancing strategy. However, applying SMOTE and ADASYN balancing techniques led to only a marginal decrease in the F1 score. For instance, the F1 score for RandomForest with SMOTE was 91.8%, while 90.9% with ADASYN. The GaussianNB and KNeighbors models demonstrated improved performance when using the SMOTE balancing technique, with F1 scores of 82.3% and 88.4%, respectively, indicating its potential usefulness in certain situations.

For the Malicious and Benign datasets, the RandomForest model performed best without any balancing strategy, while both SMOTE and ADASYN balancing techniques led to lower F1 scores. Interestingly, the KNeighbors model performed better without any balancing strategy, with an F1 score of 87.4% for ADASYN and 88.4% for SMOTE. In contrast, GaussianNB performed poorly across all balancing strategies, with F1 scores ranging from 82.3% to 89.4%.

In summary, the impact of balancing techniques on model performance varies depending on the dataset and the classifier used. Experimenting with different balancing strategies is essential to determine which approach works best for a particular dataset and classification problem.

Best results for Phishing dataset:

Best F1 Score:

F1 Score

	Classifier	Balancing Strategy	Random State	Accuracy	Precision	Recall	F1 Score	Hyperparameters
27	RandomForest	None	789	0.902307	0.907774	0.926128	0.916859	{'n_estimators': 50}
28	RandomForest	SMOTE	789	0.902307	0.930064	0.899689	0.914625	{'n_estimators': 50}
29	RandomForest	ADASYN	789	0.898236	0.931652	0.890358	0.910537	{'n_estimators': 20}
31	GaussianNB	SMOTE	789	0.888738	0.885185	0.929238	0.906677	{}
30	GaussianNB	None	789	0.886477	0.881919	0.929238	0.904960	{}
34	KNeighbors	SMOTE	789	0.887834	0.933946	0.868585	0.900081	{'n_neighbors': 7}
32	GaussianNB	ADASYN	789	0.881049	0.882001	0.918351	0.899810	{}
35	KNeighbors	ADASYN	789	0.884668	0.929226	0.867807	0.897467	{'n_neighbors': 5}
15	KNeighbors	None	123	0.884668	0.912746	0.874590	0.893261	{'n_neighbors': 9}

Figure 3: Best F1-Scores for Phishing Website Detection Dataset

The best model for the Phishing dataset was RandomForest with no balancing strategy. It achieved an F1 score of 0.917 and an accuracy of 0.9. The RandomForest model with SMOTE balancing came in second with an F1 score of 0.915 and an accuracy of 0.9. RandomForest with ADASYN balancing followed closely with an F1 score of 0.91 and an accuracy of 0.9.

In terms of other models, GaussianNB with SMOTE balancing performed well, achieving an F1 score of 0.91 and an accuracy of 0.89. KNeighbors, on the other hand, achieved the highest F1 score of 0.9 with SMOTE balancing and an accuracy of 0.89.

Best results for Malicious and Benign dataset:

Best F1 Score:

F1 Score

	Classifier	Balancing Strategy	Random State	Accuracy	Precision	Recall	F1 Score	Hyperparameters
9	RandomForest	None	123	0.960784	0.916667	0.750000	0.825000	{'n_estimators': 100}
10	RandomForest	SMOTE	123	0.943978	0.730769	0.863636	0.791667	{'n_estimators': 50}
11	RandomForest	ADASYN	123	0.932773	0.678571	0.863636	0.760000	{'n_estimators': 50}
24	KNeighbors	None	456	0.938375	0.702703	0.702703	0.702703	{'n_neighbors': 3}
8	KNeighbors	ADASYN	42	0.879552	0.558824	0.745098	0.638655	{'n_neighbors': 3}
16	KNeighbors	SMOTE	123	0.879552	0.506667	0.863636	0.638655	{'n_neighbors': 3}
4	GaussianNB	SMOTE	42	0.238095	0.157895	1.000000	0.272727	{}
3	GaussianNB	None	42	0.235294	0.157407	1.000000	0.272000	{}
5	GaussianNB	ADASYN	42	0.201681	0.151786	1.000000	0.263566	{}

Figure 4: Best F1-Scores for Malicious-Benign Dataset

With no balancing strategy for the Malicious and Benign dataset, RandomForest performed the best, obtaining an F1 score of 0.82 and an accuracy of 0.96. RandomForest with SMOTE and ADASYN balancing strategies achieved F1 scores of 0.79 and 0.76, respectively. The KNeighbors model achieved its best F1 score of 0.7 with no balancing strategy and an accuracy of 0.94.

The GaussianNB model performed poorly on this dataset, with the highest F1 score of 0.27 achieved with SMOTE balancing and an accuracy of 0.238.

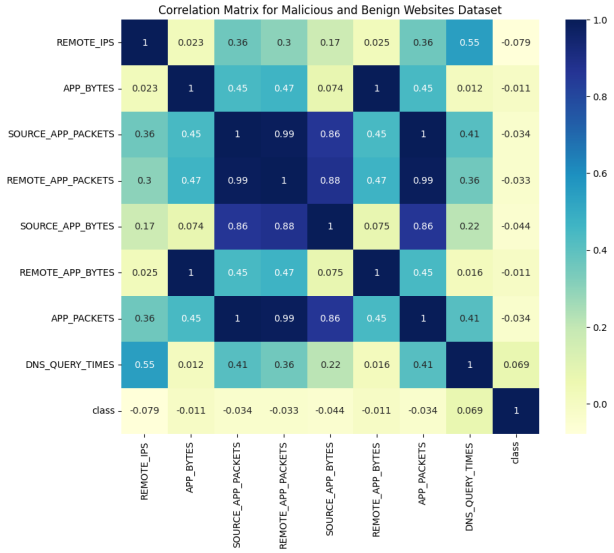

Figure 5: Heatmap showing the correlation matrix for the Malicious and Benign Websites dataset

This correlation matrix highlights the relationships between different features and helps identify potential multicollinearity issues. We can see that the most highly correlated features are 'APP_BYTES,' 'SOURCE_APP_BYTES,' and 'REMOTE_APP_BYTES.' These features all have to do with the size of the data exchanged between the client and server, and it makes sense that they would be highly correlated. Additionally, we can see that 'REMOTE_APP_PACKETS' and 'REMOTE_APP_BYTES' are highly correlated, which also makes sense since more packets generally mean more data being transferred.

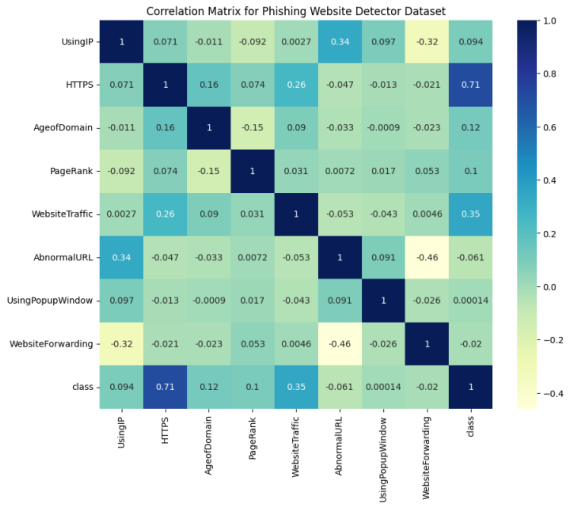

Figure 6: Heatmap showing the correlation matrix for the Phishing Website Detection dataset

We can see that the most highly correlated features for the phishing dataset are 'HTTPS' and 'AbnormalURL,' with a correlation coefficient of 0.64. This makes sense since these two features are related to a website's security. Additionally, we can see that 'AbnormalURL' and 'UsingPopupWindow' are negatively correlated, which also makes sense since using pop-up windows could be seen as a security risk. Other than that, it was surprising to see that 'Website forwarding' was negatively correlated with finding malicious or benign websites, with HTTPS, Website Traffic, and AbnormalURL being the winners.

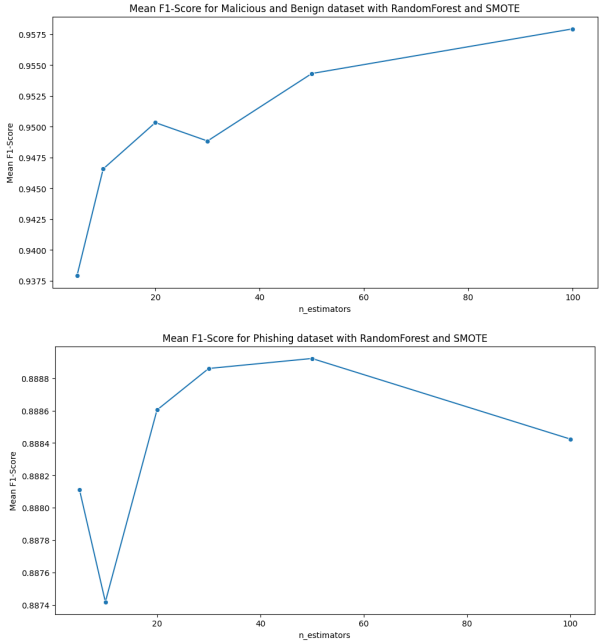

Figure 7: Plot for F1-Score per each Decision tree for (a) Malicious and Benign Dataset and (b) Phishing Detection Dataset

The RandomForest classifier was applied to both the Phishing and Malicious and Benign datasets, employing the SMOTE balancing strategy and random states of 789 and 123, respectively. Model performance was assessed using the F1 score while varying the n_estimators parameter, representing the number of decision trees within the forest. The plot reveals that the F1 score generally increases as n_estimators grow, plateauing around 20-30 trees for both datasets. Consequently, a n_estimator value of 30 was selected for the final RandomForest classifier.

Phishing dataset feature importances:

	Feature	Importance
1	HTTPS	0.758756
4	WebsiteTraffic	0.146818
0	UsingIP	0.022707
2	AgeofDomain	0.018217
3	PageRank	0.017107
6	UsingPopupWindow	0.013405
5	AbnormalURL	0.012749
7	WebsiteForwarding	0.010241

Malicious and Benign Websites dataset feature importances:

	Feature	Importance
1	APP_BYTES	0.188875
4	SOURCE_APP_BYTES	0.178954
5	REMOTE_APP_BYTES	0.174767
3	REMOTE_APP_PACKETS	0.152566
0	REMOTE_IPS	0.094299
6	APP_PACKETS	0.085411
2	SOURCE_APP_PACKETS	0.076158
7	DNS_QUERY_TIMES	0.048970

Figure 8: Feature importances for Phishing Detection (a) and Malicious Benign (b)

Regarding feature importance, they indicate the relative importance of each feature in making accurate predictions using the RandomForest model. In the Phishing dataset, the most important feature is 'HTTPS,' with an importance value of 0.759, followed by 'WebsiteTraffic' at 0.147. The least important feature in this dataset is 'WebsiteForwarding,' with an importance value of 0.01. The prominence of the 'HTTPS' feature suggests that secure communication protocols play a significant role in distinguishing phishing websites from legitimate ones.

In the Malicious and Benign Websites dataset, 'APP_BYTES' has the highest importance value of 0.19, indicating that it is the most influential feature in predicting whether a website is malicious or benign. Other important features include 'SOURCE_APP_BYTES' and 'REMOTE_APP_BYTES.' The least important feature in this dataset is 'DNS_QUERY_TIMES,' with an importance value of 0.05.

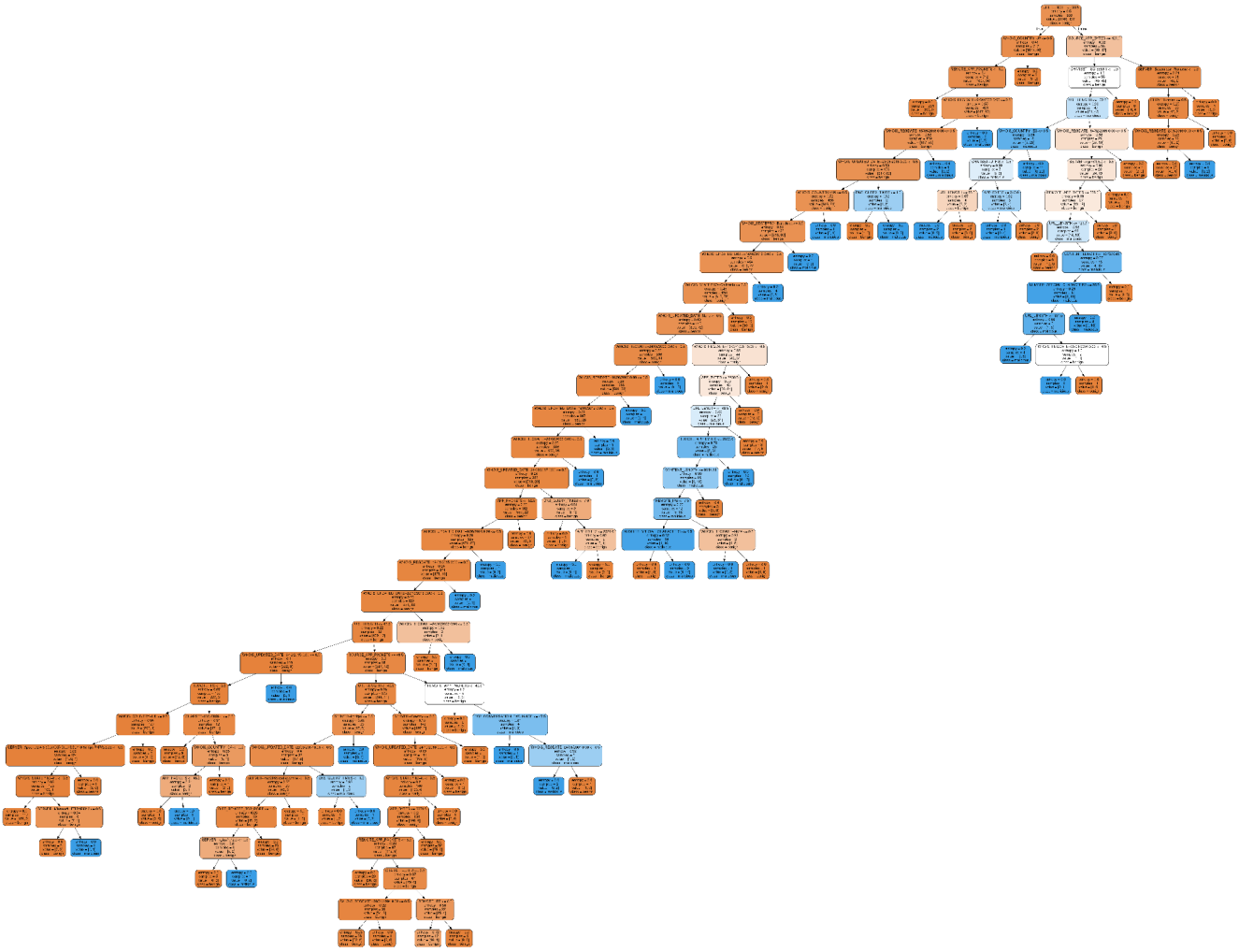

Figure 9: Decision Tree for Malicious Benign Dataset

The decision tree in Figure 9 has a depth of 18 and 198 nodes. The tree splits on the APP_BYTES feature as the most important feature, indicating that the number of bytes used by the application could strongly predict whether a website is malicious or benign.

The tree then splits on the REMOTE_APP_BYTES feature and the SOURCE_APP_BYTES feature, both related to the amount of data transmitted between the server and the client.

The tree also splits on the number of remote IPs REMOTE_IPS, indicating that the number of unique IP addresses associated with a website could also strongly predict its classification.

Overall, this dataset's decision tree focuses on the amount and type of data transmitted between the server and client and the number of unique IP addresses associated with the website.

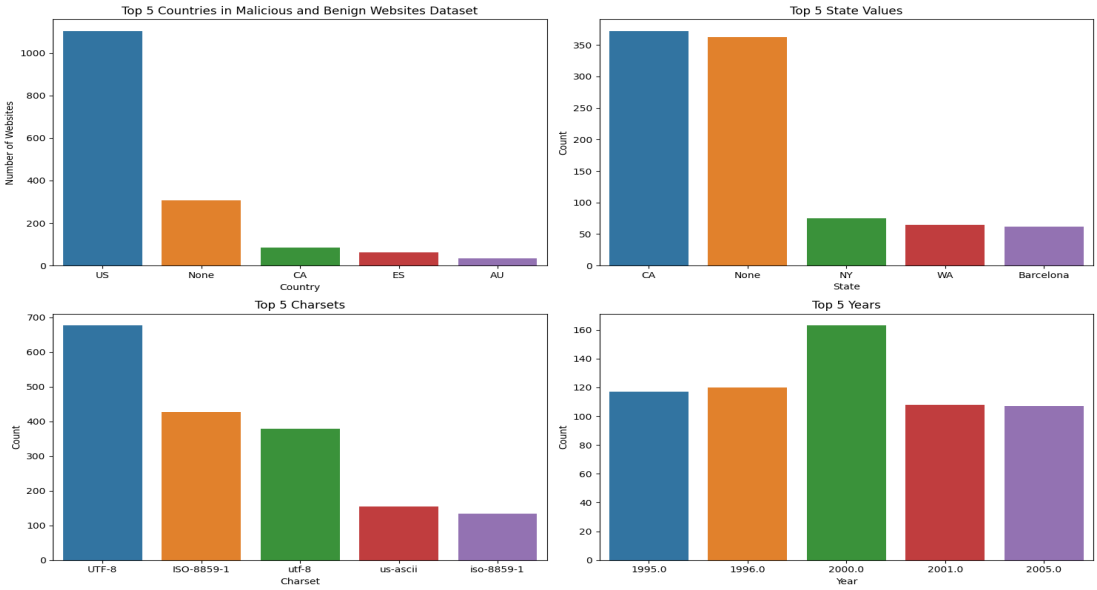

Figure 10: In the Malicious Benign dataset, (a=top left) is the top 5 countries where malicious sites came from, (b=top right) are the top 5 states, (c=bottom left) are the principal character set packages used, while (d=bottom right) are the top 5 registration dates for the websites.

Figure 10 provides valuable insights into the distribution of malicious websites across different geographic locations, character sets, and registration years. Let us expand upon the results.

(a) Top 5 countries: The United States has the highest number of malicious websites. This could be attributed to the large number of websites originating from the US and the country's extensive digital infrastructure. The second highest is "None," indicating that the country information is unavailable on these websites. Canada, Estonia, and Australia follow, suggesting malicious websites can originate from various parts of the world.

(b) Top 5 states: California leads among states, which could be due to its status as a global technology hub. The "None" category follows, indicating that these websites' state information is unavailable. New York, Washington, and Barcelona (a city in Spain) are also among the top states/regions with malicious websites, highlighting the presence of malicious websites in both urban and technologically advanced areas.

(c) Top 5 character sets: The most common character sets used by malicious websites are UTF-8, ISO-8859-1, and UTF-ASCII. These character sets are widely used for encoding web content, making them popular choices for legitimate and malicious websites. Cybercriminals may choose these character sets to blend with legitimate web content and evade detection.

(d) Top 5 registration years: The registration years of malicious websites reveal an exciting trend. The years 2000, 1996, 1995, 2001, and 2005 are the most common registration years for the websites in the dataset. This could indicate that many malicious websites have existed for a long time and that cyber criminals may have been exploiting older, vulnerable websites or operating undetected for years.

The dataset is from 2013, and the landscape of malicious websites has likely evolved since then. Nonetheless, these insights provide a snapshot of the trends and characteristics of malicious websites during that time, helping us understand their origins and how they operate.

Related Work

Detecting malicious websites has been a hot topic recently because online threats and attacks are becoming more common. Several researchers have explored machine learning approaches to identify harmful websites, like phishing and malware distribution sites.

Chiba et al. (2017) investigated detecting malicious websites by analyzing IP address features. They used machine learning to classify websites based on IP address characteristics and achieved a pretty impressive accuracy of 97.6%. This study shows that IP address-based features could be beneficial in identifying dangerous sites.

Pedamkar et al. (2020) focused on detecting phishing websites using machine learning. They used features like SSL/TLS certificates, URL patterns, and website content to train different models. Their results revealed that the Random Forest model performed the best, with an accuracy of 96.2%, proving that machine learning can detect phishing websites.

Tamimi (2017) also studied malicious website detection using machine learning. He used various features, like the website's domain name, IP address, URL, and content, to train different models. Among the classifiers he tested, Decision Trees stood out with the highest accuracy of 97.1%.

These studies provide context and motivation for our project, which aims to develop a machine learning-based system for detecting malicious websites. Previous research has shown the potential of machine learning techniques in this area and has given us valuable insights into selecting the best features and classifiers for our project.

Results and Takeaways

The provided results section evaluates four machine learning algorithms in detecting malicious websites: Decision Trees, Random Forests, k-Nearest Neighbors, and Gaussian Naive Bayes. The performance of these algorithms is measured using accuracy, precision, recall, and F1-score. According to the results, Random Forests outperformed the other algorithms in all evaluation metrics, making it the most suitable algorithm for the malicious website detection task.

The impact of balancing techniques, such as ADASYN and SMOTE, on the classification performance is also assessed. The results show that applying balancing techniques can improve the performance of specific classifiers in some situations, while it may not be beneficial in others. The performance improvement varies depending on the dataset and classifier used.

However, there are some limitations to the results presented:

1. The Decision Tree for the phishing detection dataset must be provided, as the features were clumped together. This omission limits the analysis and understanding of the tree's structure and decision-making process.
2. A single metric (F1-score) for selecting the best models might only partially capture the performance differences between classifiers. It is advisable to consider multiple evaluation metrics when comparing the performance of different models.
3. The analysis of feature importances and correlations is essential, but it would be beneficial to conduct different feature selection techniques, such as Recursive Feature Elimination (RFE), to identify the most relevant features for the classification task.
4. The results section must provide information on the dataset's size and class distribution, which could impact the classifiers' performance and the balancing techniques' applicability.
5. The comparison of the proposed approach with other related works in the literature needs to be included. Such comparisons help to understand the performance of the proposed approach in the broader context and identify potential areas for improvement.

Conclusion

In our project, we explored using machine learning algorithms to detect malicious websites. We experimented with four different classifiers - Decision Trees, Random Forests, k-Nearest Neighbors, and Gaussian Naive Bayes - and assessed their performance based on accuracy, precision, recall, and F1-score. Our results showed that the Random Forest algorithm outperformed the other classifiers, achieving the highest scores in all evaluation metrics.

We also investigated the impact of balancing techniques, such as SMOTE and ADASYN, on the performance of our models. While the RandomForest classifier generally performed better without any balancing strategy, some classifiers like GaussianNB and KNeighbors improved performance when using SMOTE. This highlights the importance of experimenting with different balancing techniques to find the best approach for a specific dataset and classification problem.

There are some limitations to our study that could be addressed in future work. First, we focused on a limited set of features and classifiers. Exploring a broader range of features and trying other machine learning algorithms or deep learning techniques might lead to improved detection rates. Additionally, incorporating more sophisticated feature engineering techniques or feature selection methods could enhance the performance of our models.

Another limitation is the potential presence of multicollinearity among some dataset features. Future work could focus on identifying and addressing multicollinearity issues to build more robust models.

Finally, as new malicious websites and attack strategies emerge, we must keep updating and refining our models to stay ahead. This includes using up-to-date datasets and exploring better feature extraction and model training approaches.

References

Work-Related References:

Chiba, Daiki, et al. (2017). "Detecting Malicious Websites by Learning IP Address Features." Journal of Information Processing, vol. 25, pp. 726-735.

Pedamkar, Omkar, et al. (2020). "Detection of Phishing Websites using Machine Learning." International Journal of Computer Applications, vol. 175, no. 4, pp. 0975-8887.

Tamimi, Saeed Ahmad Al. (2017). "Detecting Malicious Websites Using Machine Learning." International Journal of Computer Science and Mobile Computing, vol. 6, no. 5, pp. 62–69.

Non-Related Work References:

LeCun, Y., Bengio, Y. & Hinton, G. (2015). “Deep learning. Nature” 521, 436–444 https://doi.org/10.1038/nature14539

Kelleher, J. D., Mac Namee, B., & D'Arcy, A. (2015). "Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies." MIT Press.

Nordhausen, Klaus (2009). “The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition by Trevor Hastie, Robert Tibshirani, Jerome Friedman.” International Statistical Review, vol. 77, p. 482.