BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
MUNSHI NAGAR, ANDHERI (WEST)- MUMBAI- 400058- INDIA
(AUTONOMOUS INSTITUTE AFFILIATED TO THE UNIVERSITY OF MUMBAI)

Phishing and Legitimate URL Detection

Author: Somanshu Mahajan

CSE Department, Sardar Patel Institute of Technology, Andheri, India

Corresponding Author: Email: somanshu.mahajan23@spit.ac.in

Biographical Notes on Contributors :

Somanshu Mahajan is an undergraduate student in the Department of Computer Science and Engineering at Bhartiya Vidya Bhavan's Sardar Patel Institute of Technology, Mumbai. Currently in his second year, his academic interests include database systems, computer networking, software development, and artificial intelligence.

Phishing and Legitimate URL Detection

Phishing remains one of the top cybersecurity threats, where users are targeted by spoofed websites imitating authentic websites to capture confidential information like usernames, passwords, and financial information. As the online activity explodes, conventional detection methods fail to detect new and advanced phishing attacks. This project overcomes the challenge through the utilization of machine learning methodologies to predict whether a URL is phishing or legitimate using different extracted features. The model was trained and tested using a dataset of more than 2,35,795 tagged URLs. The data were pre-processed, features selected, and transformed in order to provide maximum input quality to machine learning models.

A number of classification models were used and compared, such as Random Forest, Decision Tree, Extra Trees, XGBoost, and K-Nearest Neighbors. Performance was evaluated using common evaluation metrics: accuracy, precision, recall, and F1-score. The Random Forest classifier performed the best, with an F1-score of 0.96, which shows high reliability in identifying phishing URLs. The overall findings prove that ensemble learning techniques, especially Random Forest and XGBoost, are extremely efficient for phishing detection tasks. This research highlights the power of data-driven solutions to enhance automated cybersecurity systems and paves the way for future research, including incorporating deep learning models or deploying the system in real-world environments.

Introduction

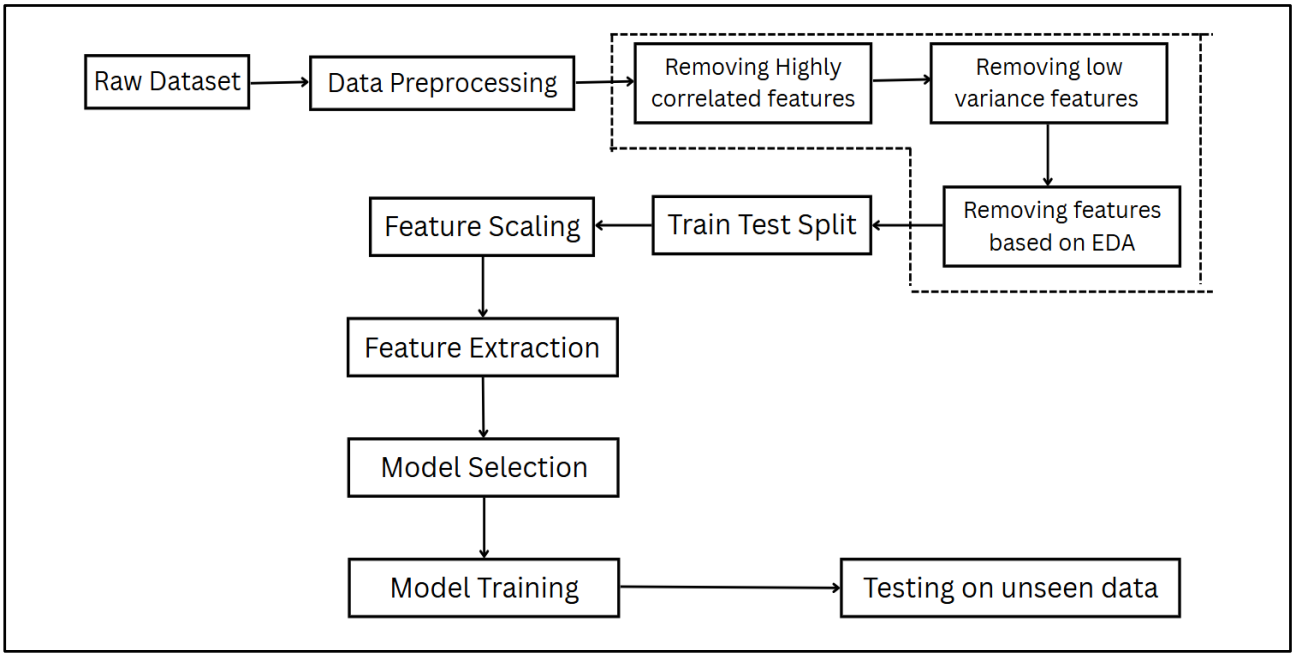
Phishing is one of the most common and dangerous types of cyberattacks seen in the digital world today. It usually involves tricking people using fake websites, misleading links, or emails that appear to be real. The main aim of these attacks is to steal sensitive information such as usernames, passwords, bank account numbers, credit card details, and other personal data. Because of how convincing these fake sources can be, many people fall for them, leading to serious problems like financial loss, stolen identities, and damage to both individuals and companies. In many cases, people don't even realize they've been scammed until it's too late, making phishing even more harmful.

What makes phishing especially dangerous is that it's not easy to recognize just by looking. Attackers create fake web pages and emails that are almost identical to trusted ones, like online banking sites or social media platforms. Even a careful person can be fooled. This is why there is a strong need for automated tools and systems that can detect phishing attempts early and accurately, before someone becomes a victim. Technology like machine learning gives us the chance to build smart systems that can learn patterns from past phishing data and use this knowledge to detect new threats.

In this project, I explored how machine learning can be applied to the problem of phishing detection. I worked with a large dataset that had over 2,35,795 website URLs, each labelled as either phishing or legitimate. I applied feature selection to pick the most important features, and then used them to train different machine learning models. These models included Random Forest, Decision Tree, Extra Trees, XGBoost, and K-Nearest Neighbors (KNN). The goal was to compare how well each model performs in terms of accuracy, precision, recall, and F1-score, and identify the most suitable model for real-world phishing detection. A reliable and accurate

model could be very helpful in increasing online safety and protecting users from falling for online scams in the future.

Methodology



Workflow

To detect phishing websites, I followed a series of steps from data preprocessing to model evaluation. I used a dataset containing over 2 lakh URLs, each labelled as either "phishing" or "legitimate." Here's how I approached the project:

1. Data Preprocessing

First, I started by cleaning the dataset to prepare it for machine learning. I checked for null values and duplicate entries, but the dataset was already clean in those areas, so no rows had to be removed or fixed. However, I did observe that outliers were present in almost a quarter of the dataset. Since these outliers had a noticeable impact on correlation and might represent important edge cases, I decided not to remove them to preserve the original data distribution and variability.

Then, I focused on handling categorical features. Some columns had too many unique values or were not useful for training, so I dropped those columns to simplify the dataset. One important column, TLD (Top-Level Domain), was retained and transformed using frequency encoding, which replaces categories with how often they appear. This helped convert the data into a format suitable for machine learning without increasing the number of features too much.

Additionally, I performed data type casting to make the dataset more memory-efficient. Numeric columns were converted to types like int32 for columns having more than 2 unique values and int8 for columns having values as 0 and 1 only, wherever appropriate to reduce storage space and improve performance. I also carried out Exploratory Data Analysis (EDA)

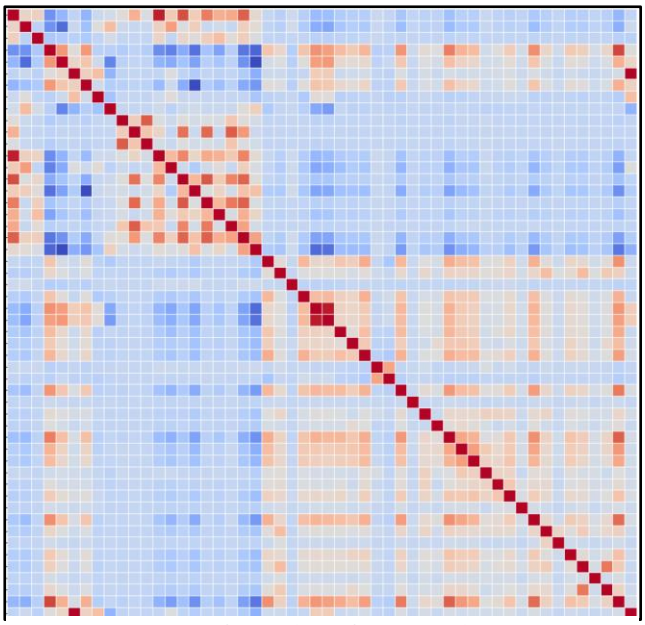
to better understand the feature distributions, identify patterns, and observe relationships between variables before training the models.

2. Redundant Feature Elimination

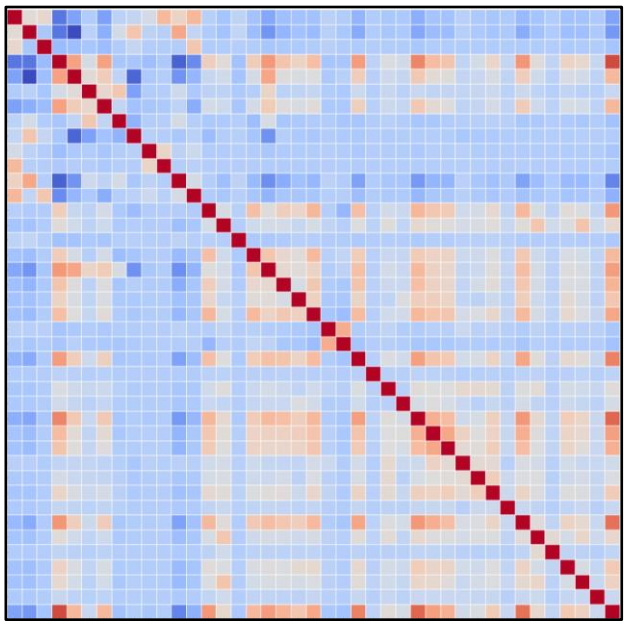
I initially started with 51 features in the dataset. To improve model performance and reduce unnecessary complexity, I applied feature elimination techniques as part of the data preprocessing process.

- *Highly Correlated Features*

  I used a correlation heatmap to identify features that had a correlation coefficient above a threshold of 0.7. These features were likely providing overlapping information to the model, which could lead to overfitting. To address this, I removed one feature from each highly correlated pair. Total of 11 features were eliminated in this process.



Before (51 features)



After (40 features)

- *Low Variance Features*

Features with very little variation across all records were removed because they do not help the model distinguish between phishing and legitimate URLs. I calculated variance for each feature and removed those with near-zero variance. This helped reduce noise in the dataset without affecting performance. Three features were eliminated by this process.

- *EDA-Based Feature Removal*

During exploratory data analysis, I noticed certain features showed inconsistent patterns or were heavily skewed with outliers. Some features had distributions that did not align with phishing behaviour and could confuse the model. Based on these insights, I eliminated such features. Total of 11 features were eliminated in this process.

After the elimination process, I reduced the number of features from 51 to 26, retaining only the most informative ones for model training. The best features will be selected during the feature extraction process to further refine the model's input for optimal performance.

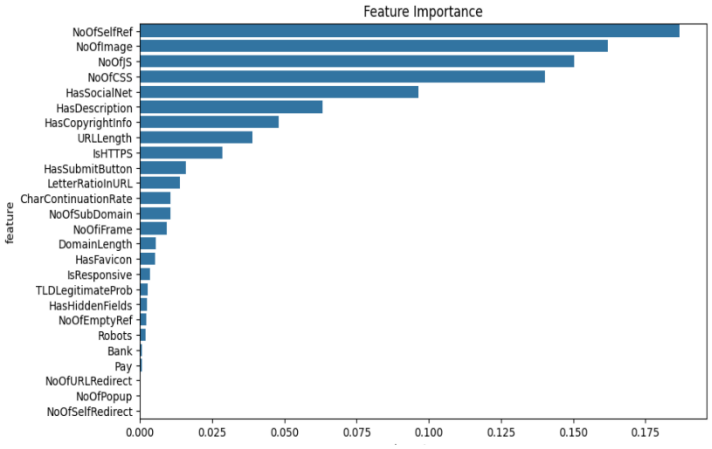
3. Data Splitting

To avoid data leakage, the train-test split was performed prior to feature extraction. This ensures that the model is tested on unseen data and prevents any information from the test set from influencing the feature selection process. The dataset was split into 70% training and 30% testing, providing enough data for training while maintaining a sufficiently large test set for evaluation.

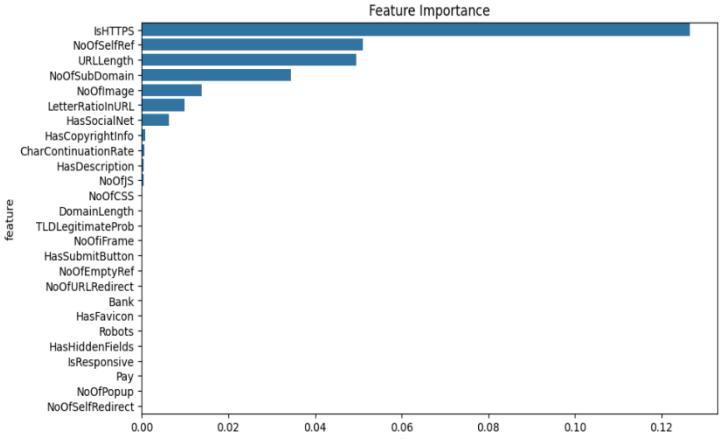
4. Feature Extraction

To begin the feature extraction process, I first scaled the dataset using StandardScaler. This step ensured that all features had a mean of 0 and a standard deviation of 1, preventing features with larger ranges from dominating the model's learning process.

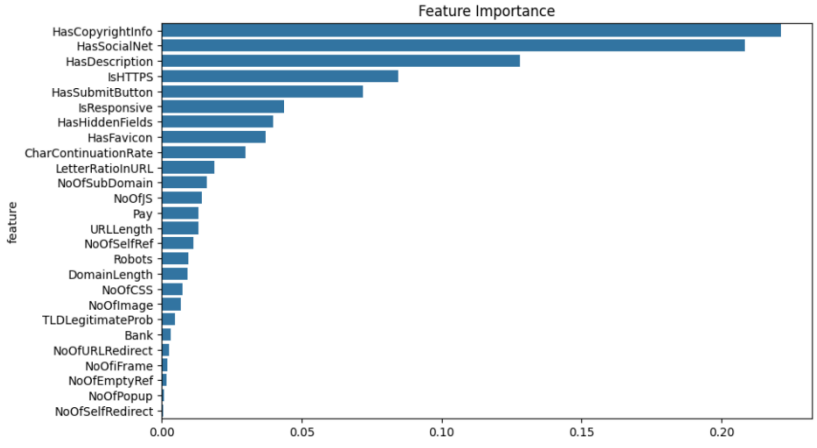
Following the scaling, I used Random Forest to evaluate feature importance, followed by Permutation Importance to further assess the relevance of each feature. Additionally, I employed the Extra Trees Classifier to rank the features based on their contribution to model performance.



Random Forest



Permutation Importance

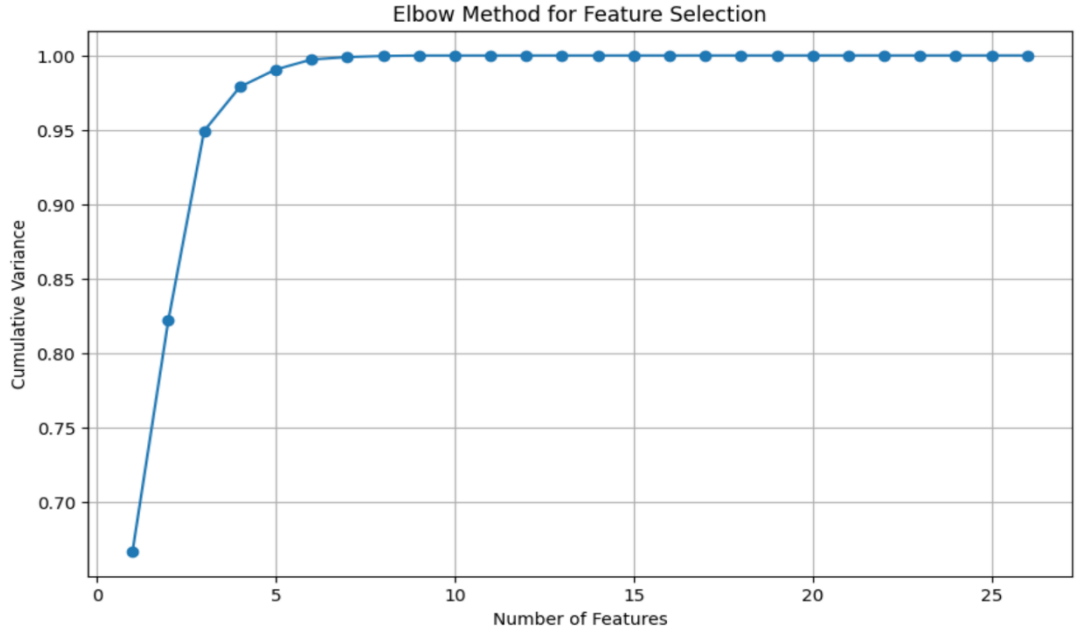


Extra Trees

To consolidate the results from these different methods, I applied Rank Aggregation. The process of Rank Aggregation involves aligning the rankings from all methods and determining the average or aggregated rank for each feature. Features that consistently ranked highly across all methods were given higher importance, while those that ranked lower across most methods were considered less important. This approach ensures that the most informative features are selected, regardless of the slight variations in rankings from different methods.

Top K Features

To determine the optimal number of features to select from the aggregated ranking, I applied the elbow method based on the cumulative variance of the features. I plotted the cumulative variance explained by the top-ranked features and observed the point where the curve started to level off - the "elbow point." This point indicates that adding more features beyond it contributes very little additional information. Based on this, I selected the top 6 features, which captured the majority of the variance in the data. This approach ensured that the final set of features used for model training was both informative and efficient, helping to reduce dimensionality without compromising performance.



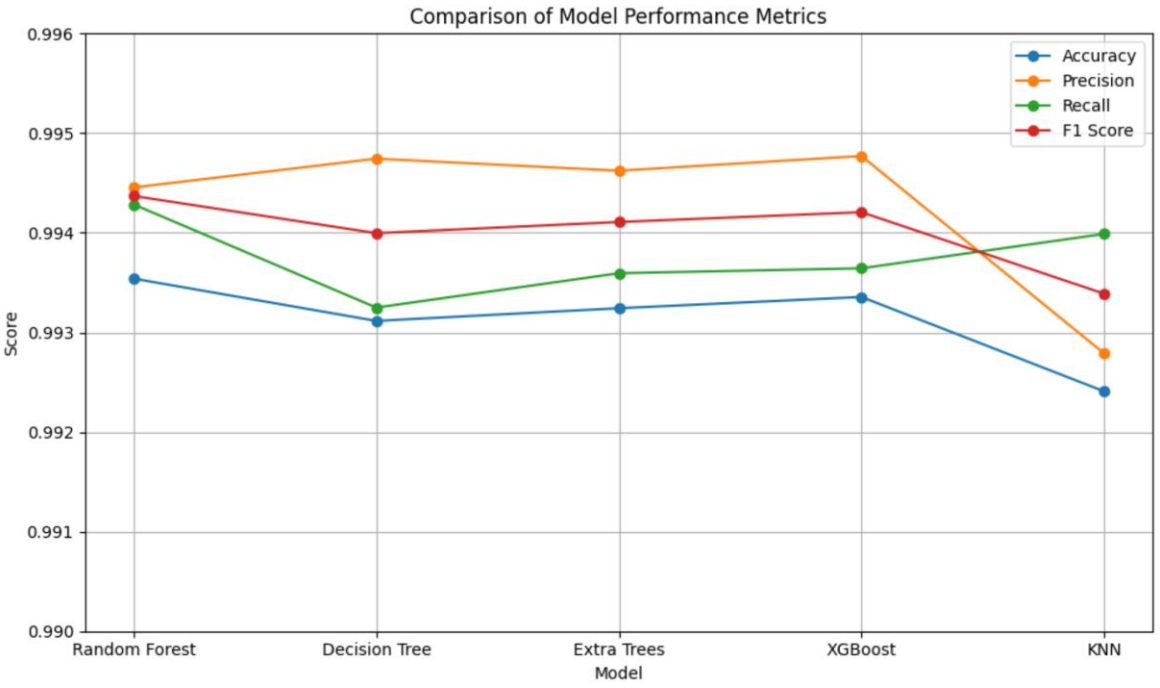
Elbow point at 6 features

5. Model Selection

After selecting the most informative features, I proceeded to train and evaluate several machine learning models to identify the best-performing one for phishing URL detection. The models used included Random Forest, Decision Tree, Extra Trees Classifier, XGBoost, and K-Nearest Neighbors. Each model was trained using the selected top features, and their performance was evaluated based on key classification metrics such as accuracy, precision, recall, and F1-score. The goal was to compare their strengths and weaknesses in identifying phishing URLs. This comparative analysis helped in understanding which model offered the most reliable and balanced performance for the task.

Comparison of Model Performance Metrics

Model	Accuracy	Precision	Recall	F1 Score
Random Forest	0.9935	0.9934	0.9934	0.9934
Decision Tree	0.9931	0.9928	0.9931	0.9930
Extra Trees	0.9933	0.9930	0.9932	0.9931
XGBoost	0.9934	0.9931	0.9933	0.9932
KNN	0.9922	0.9923	0.9921	0.9922



Although both Random Forest and XGBoost demonstrated excellent performance on the phishing detection task, Random Forest was chosen as the final model due to its slightly better balance between precision and recall, as well as its simplicity and faster training time. XGBoost, while powerful and often very accurate, is more complex to tune and requires significantly longer training time, especially on large datasets. Additionally, Random Forest is more interpretable and easier to integrate into production environments where simplicity and efficiency are prioritized. Therefore, despite XGBoost's competitive scores, Random Forest was selected as the preferred model.

6. Model Training

For model training, I performed hyperparameter tuning using randomized search, which helped identify the optimal st of hyperparameters for the model. This process was crucial in improving the model's performance by efficiently exploring a wide range of hyperparameter values.

Additionally, to reduce the false positive rate, I applied threshold tuning. This technique adjusted the decision threshold of the model, allowing for better control over the trade-off between precision and recall, ultimately improving the model's classification accuracy in terms of false positives.

In anomaly detection, reducing the false positive rate is crucial because false positives can lead to significant issues, especially in critical systems where detecting normal behaviour is essential. Anomalies often indicate potential risks or outliers, and if these are incorrectly classified as normal (false positives), it can result in missed opportunities to address genuine issues. For example, in cybersecurity or fraud detection, false positives may cause unnecessary investigations or resource allocation, leading to inefficiencies. By reducing the false positive rate, we ensure that the model focuses on accurately identifying true anomalies, avoiding misclassifications that could undermine the trust and effectiveness of the detection system.

Results

The model's performance was evaluated at a threshold of 0.6 to assess its ability to correctly classify instances while minimizing false positives. The training accuracy was observed to be 0.9949, and the testing accuracy was slightly lower at 0.9938, indicating that the model generalizes well to unseen data.

Testing Classification Report

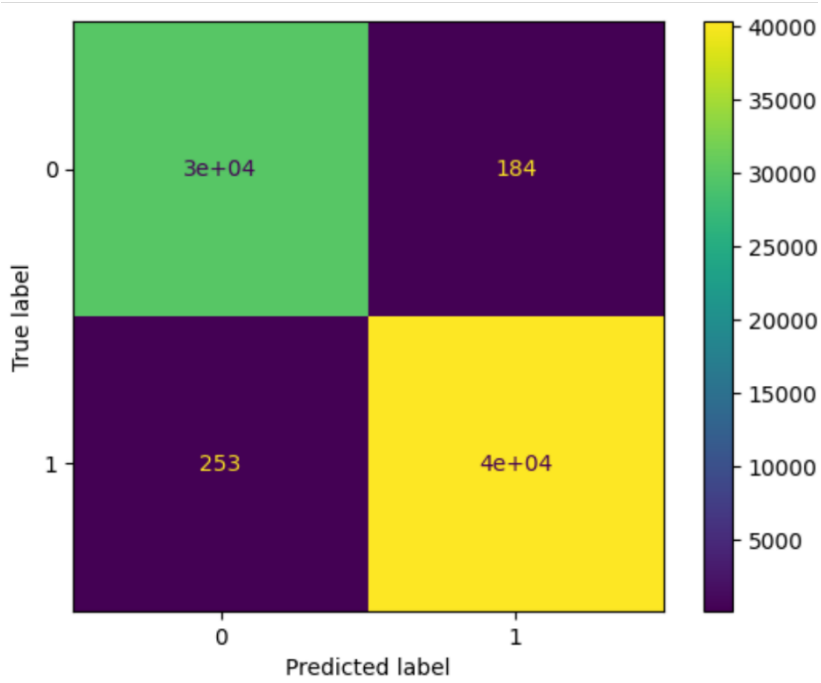
Class	Precision	Recall	F1-Score	Support
0	0.9916	0.9939	0.9928	30,151
1	0.9955	0.9938	0.9946	40,588
Accuracy			**0.9938**	**70,739**

The model achieved a high precision and recall for both classes. Precision for class 1 (anomaly) was 0.9955, while for class 0 (normal), it was 0.9916. The recall values were very similar, with class 0 having a recall of 0.9939 and class 1 at 0.9938. This demonstrates that the model is effectively identifying both normal and anomalous instances, with minimal misclassifications.

Training and Testing Accuracy at Threshold 0.6

Metric	Value
Training Accuracy	0.9949
Testing Accuracy	0.9938

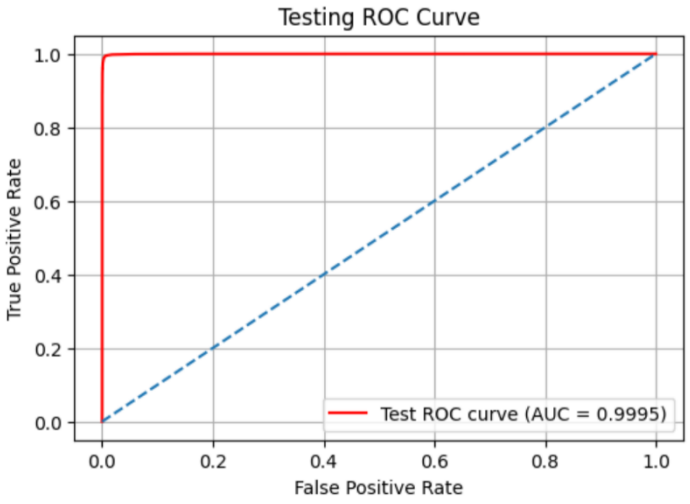
The accuracy values indicate strong performance, with minimal overfitting, as the difference between training and testing accuracy is very small.

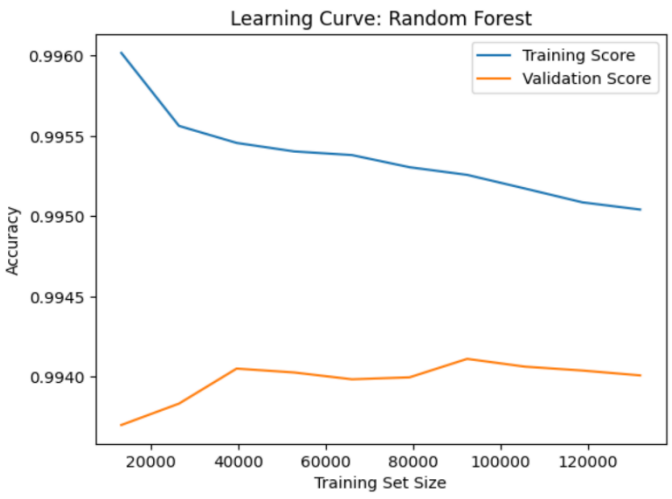


Confusion Matrix

The matrix highlights that the model performs well with a significant number of true positives and true negatives. However, there are still a few false positives and false negatives. The false positive rate is relatively low, indicating that the model doesn't mistakenly classify normal instances as anomalies too often. Similarly, the false negative rate is also manageable, but efforts to reduce it further, especially in anomaly detection, may be necessary to ensure no important anomalies are overlooked.

Other Evaluation Parameters





The red line represents the ROC curve. The ROC curve is very close to the ideal curve, hugging the top-left corner. This shows that the model has a very high ability to distinguish between positive and negative classes.

The decrease in accuracy of training score as the set size increase is showing that model is generalizing well. Validation score is also increasing with set size which indicates it is generalizing good on unseen data. Also gap between both scores is less meaning model is not suffering overfitting.

Conclusion

In this project, I worked on detecting phishing websites using different machine learning models. Phishing is a serious problem that tricks people into sharing their private information through fake URLs. By using a large dataset of labelled URLs, I trained and tested several models to find out which one works best.

After comparing the results, I found that the Random Forest model gave the highest performance in terms of accuracy, precision, recall, and F1-score. Other models like Extra Trees and XGBoost also gave good results, while Decision Tree and K-Nearest Neighbors were slightly less accurate. This shows that ensemble models are more powerful and reliable for solving this type of problem.

Overall, machine learning is a strong tool for detecting phishing URLs and can help make the internet safer. In the future, this kind of system can be used in real-time to alert users before they enter a dangerous site. With further improvements and real-world testing, it can be turned into a helpful application or browser extension.

References

- Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd Edition). O'Reilly Media, 2019.
- PhiUSIIL Phishing URLs (Kaggle) – Phishing Websites Dataset. - https://www.kaggle.com/datasets/joebeachcapital/phiusiil-phishing-url
- PhiUSIIL: A diverse security profile empowered phishing URL detection framework based on similarity index and incremental learning (Introductory Paper) - https://www.sciencedirect.com/science/article/abs/pii/S0167404823004558