CSE422 Lab Project Report

CSE-422
Lab Project Report
Section: 15
Group: 11

Group Members
Member 1: TASFIA TASNIM PRIOTY
ID: 23241101

Member 2: Maisha Binta Alam
ID: 21201596

Project Title:

Phishing Website Detection For Cybersecurity

Course

CSE422-Artificial Intelligence

Lab Instructor

Shoaib Ahmed Dipu

And

Pollock Nag

Submitted by:

Tasfia Tasnim Prioty-23241101

Maisha Binta Alam-21201596

Date: January 7,2025

Table of Contents

1. Introduction

Aim: The primary aim of this project is to detect whether a website is a phishing website or not by using machine learning models. Our project aims to identify a phishing website by successfully analysing the patterns and features of legitimate websites and phishing websites.

Motivation: Phishing is a harmful online cyber attack, where attackers create fake websites to get sensitive, personal information from the users such as address, phone number, credit card details etc. Many innocent people got scammed because of these fake websites, losing their valuable information, money etc.The motivation behind this project is to protect innocent people from these kinds of malicious activities by creating a safe space for the online users and enhancing the cybersecurity.

2. Dataset Description

Source:

● **Link**: https://www.kaggle.com/datasets/arnavs19/phishing-websites-dataset
● **Reference**: Kaggle Phishing Websites Dataset

Dataset Overview :

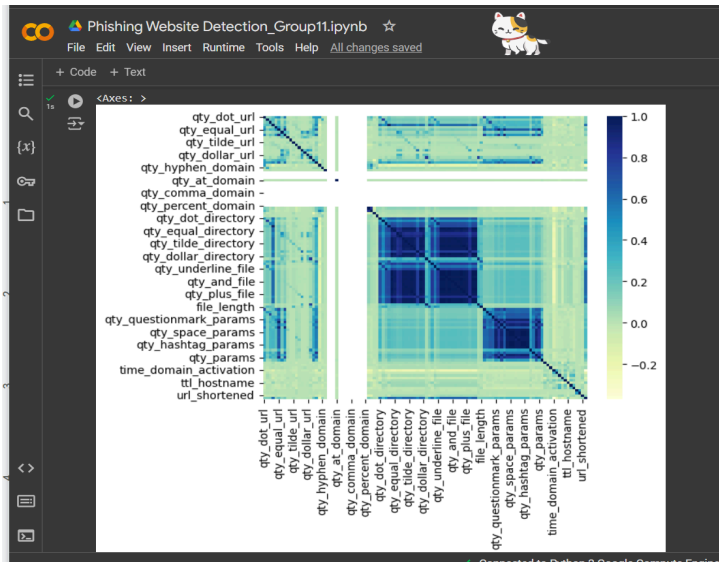
● **number of features:** 111

It is a Classification Problem: Phishing Website Detection is a binary classification problem because the target variable, phishing, is representing two classes, whether a website is phishing (1) or legitimate (0). The goal is to classify websites into one of these two classes based on features like . As there are two classes to classify here, that is why it is a binary classification problem.

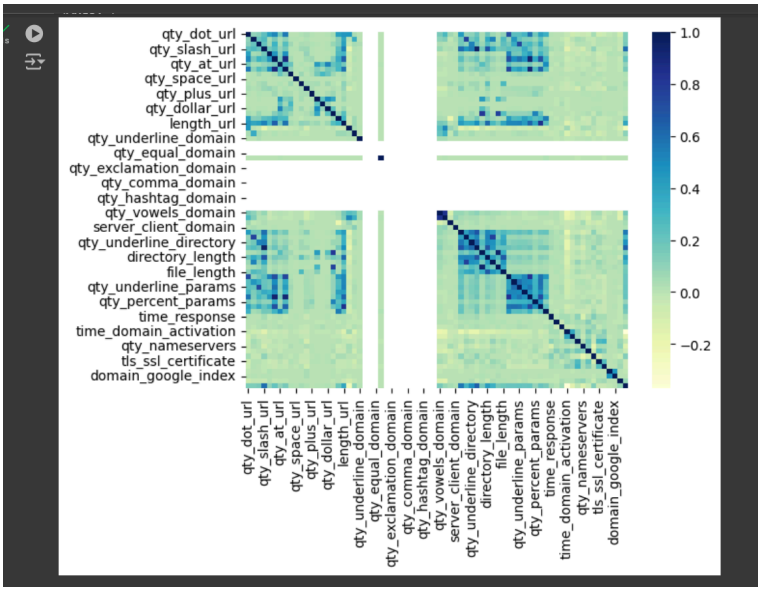
● **Data Points**: 58645
● **Feature Types**:
○ Quantitative: All features are numerical here.
○ Categorical: There are no categorical values in our dataset.

Correlation Analysis

A heatmap was generated using Seaborn to visualize the correlation between features. Here is the initial heatmap-



After we preprocessed our data, by dropping irrelevant or more correlated features, the resulting heatmap of the cleaned dataset was this-

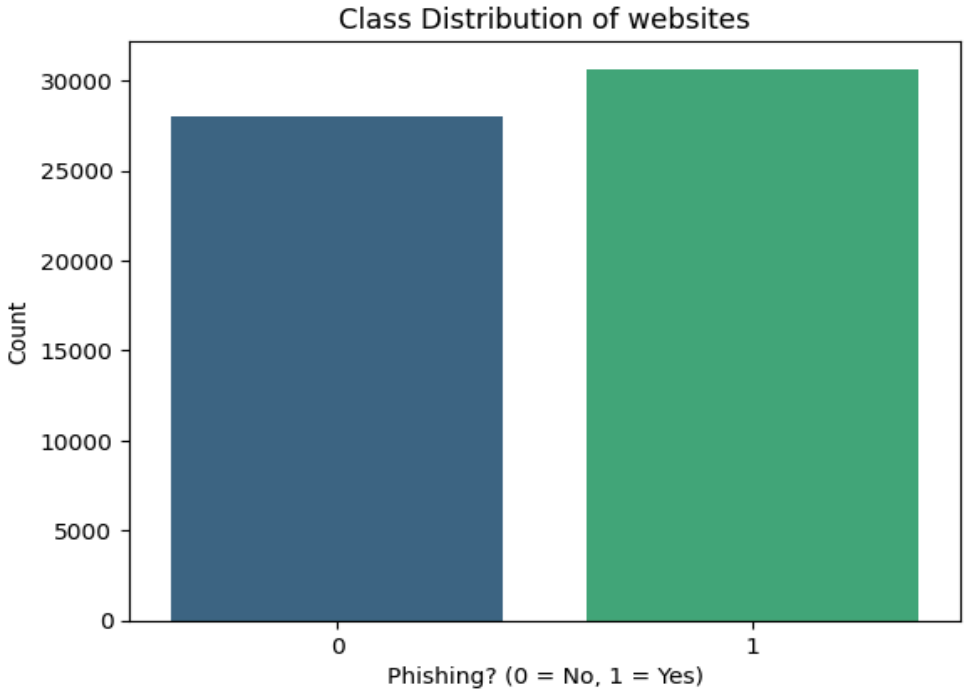


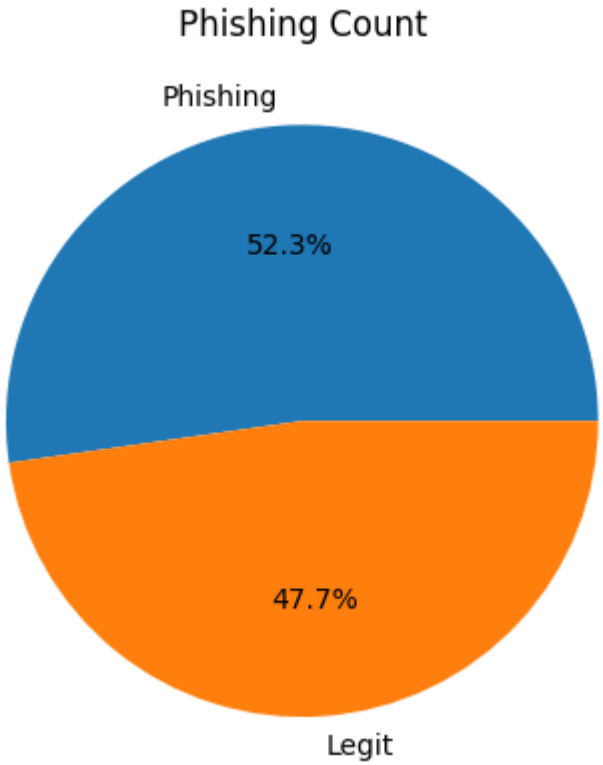
Imbalanced Dataset

The dataset is slightly imbalanced as the number of phishing websites are slightly higher than the number of legitimate websites.

- **Legitimate (0)**: 27998 instances
- **Phishing (1)**: 30647 instances

A bar chart and also a pie chart was plotted to highlight the imbalance. There we plotted the bar chart for legitimate (27998) and phishing(30647) .

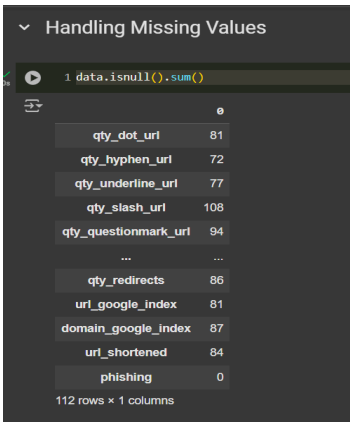

Class Distribution of websites
30000
25000
20000
15000
10000
5000
0
Count
0
1
Phishing? (0 = No, 1 = Yes)


Phishing Count
Phishing
52.3%
47.7%
Legit

3. Dataset Preprocessing

Faults Identified

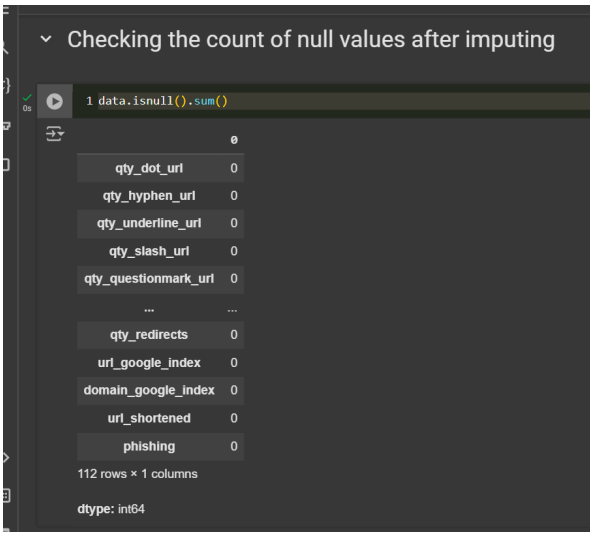
- **Null Values**: There are many null values in the dataset. We can see the number of null values in each feature by data.isnull().sum() command.



Solutions Applied

- **Handling Null Values (Imputing mean)**:

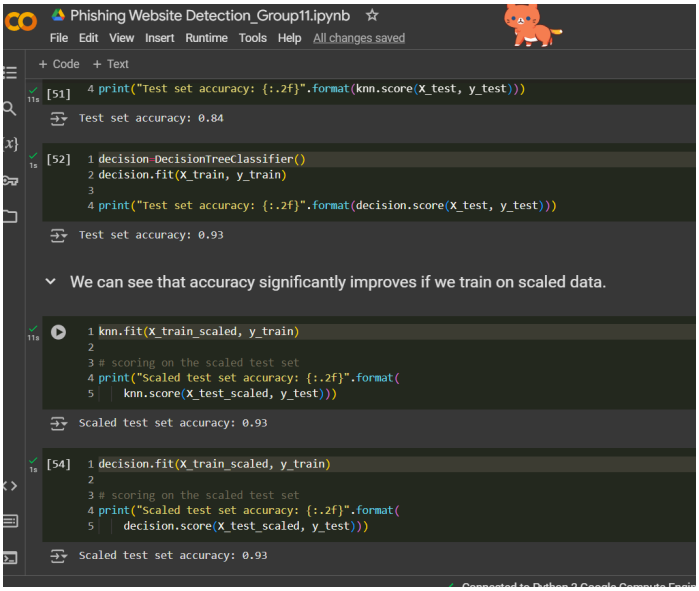
Here we are imputing the null values in each column by the mean of the other values in that particular column. Like this, the dataset would be balanced and precise. We could just drop the rows containing the null values, but in that way, many important values of other columns could have been lost. In our way, no values will be lost.



We can see above, there is no null value after imputing the mean in each column. In this way, the model's performance will be better.

4. Feature Scaling

Min-Max scaling was applied to all the features to normalize them into a range of [0, 1], because initially, the dataset was more scattered. After scaling, the performance of the model improved significantly. For example, in our projects, the KNN model was only giving 84% accuracy initially, but after scaling the dataset, the accuracy of the same model significantly improved, which was 93%.



5. Dataset Splitting

The dataset was split into training (70%) and testing (30%) sets using a random split. According to the 30% of the test data we ran four models then predicted and calculated accuracy of those models.

6. Model Training & Testing

- **K-Nearest Neighbors (KNN):** KNN achieved the highest accuracy of **92.76%** among all the models, with precision, recall, and F1-scores consistently around **0.93** for both classes. This indicates that the model performed well in balancing true positives and true negatives without significant bias toward either class. Its strong performance makes it the

most reliable model for this dataset.

KNN train predict and evaluate

```
# Initialize and train
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)

# Predict and evaluate
y_pred_knn = knn.predict(X_test_scaled)
print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))
```

```
KNN Accuracy: 0.927645788336933
              precision    recall  f1-score   support

           0       0.92      0.93      0.92      8428
           1       0.93      0.93      0.93      9166

    accuracy                           0.93     17594
   macro avg       0.93      0.93      0.93     17594
weighted avg       0.93      0.93      0.93     17594
```

- **Decision Tree Classifier:** The Decision Tree model had an accuracy of 89.95%, showing good overall performance but with some imbalances. For Class 0, it achieved high precision (0.93) but lower recall (0.85), while for Class 1, it had moderate precision (0.88) and high recall (0.94). This suggests that while the model is effective at identifying positives, it occasionally misclassified negatives. Despite being slightly less consistent than KNN, it remains a competitive model.

Decision Tree train predict and evaluate

```
[34] decision = DecisionTreeClassifier(max_depth=5, random_state=42)
     decision.fit(X_train_scaled, y_train)

     # Predict and evaluate
     y_pred_dt = decision.predict(X_test_scaled)
     print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
     print(classification_report(y_test, y_pred_dt))
```

```
Decision Tree Accuracy: 0.8994543594407184
              precision    recall  f1-score   support

           0       0.93      0.85      0.89      8428
           1       0.88      0.94      0.91      9166

    accuracy                           0.90     17594
   macro avg       0.90      0.90      0.90     17594
weighted avg       0.90      0.90      0.90     17594
```

- **Logistic Regression:** Logistic Regression recorded an accuracy of **89.16%**, demonstrating balanced performance across both classes. Precision values ranged from **0.88 to 0.91**, while recall ranged from **0.86 to 0.92**, resulting in good F1-scores. This makes Logistic Regression a solid option for datasets with linear relationships, although it was marginally outperformed by KNN and Decision Tree.

Logistic Regression train predict and evaluate

```
lr = LogisticRegression(random_state=42)
lr.fit(X_train_scaled, y_train)

# Predict and evaluate
y_pred_lr = lr.predict(X_test_scaled)
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_lr))
print(classification_report(y_test, y_pred_lr))
```

```
Logistic Regression Accuracy: 0.8916107764010458
              precision    recall  f1-score   support

           0       0.91      0.86      0.88      8428
           1       0.88      0.92      0.90      9166

    accuracy                           0.89     17594
   macro avg       0.89      0.89      0.89     17594
weighted avg       0.89      0.89      0.89     17594
```

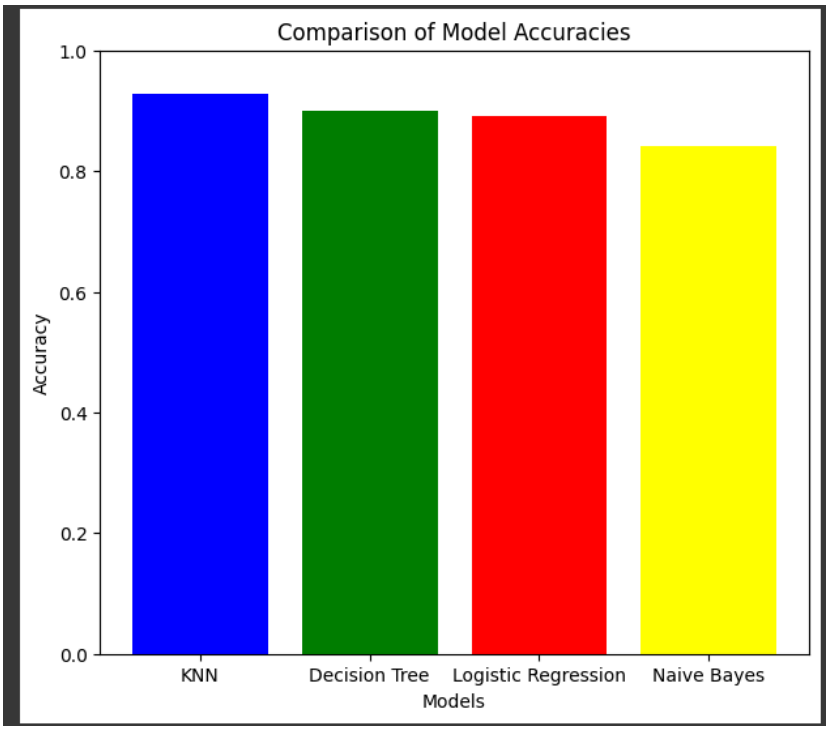
- **Naive Bayes:** Naive Bayes performed poorly with an accuracy of **60.83%**, struggling to handle the dataset's complexity. For Class 0, it achieved high recall (**0.98**) but very low precision (**0.55**), indicating many false positives. For Class 1, while precision was high (**0.93**), recall was very low (**0.27**), suggesting frequent false negatives. The poor results are likely due to the algorithm's assumption of feature independence, which does not align with the dataset.

Naive Bayes train predict and evaluate

```
naive_bayes = GaussianNB()
naive_bayes.fit(X_train_scaled, y_train)

# Predict and evaluate
y_pred_nb = naive_bayes.predict(X_test_scaled)
print("Naive Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
print(classification_report(y_test, y_pred_nb))
```

```
Naive Bayes Accuracy: 0.6082755484824371
              precision    recall  f1-score   support

           0       0.55      0.98      0.70      8428
           1       0.93      0.27      0.42      9166

    accuracy                           0.61     17594
   macro avg       0.74      0.62      0.56     17594
weighted avg       0.75      0.61      0.56     17594
```

7. Model Selection/Comparison Analysis



The comparison shows that KNN achieves the highest accuracy, making it the best-performing model, followed closely by Decision Tree and Logistic Regression, which have comparable and robust results. In contrast, Naive Bayes performs the worst, with significantly lower accuracy, likely due to its inability to handle feature dependencies effectively. This analysis highlights KNN's suitability for the dataset and Naive Bayes' limitations.

8. Conclusion:

The phishing website detection model leverages machine learning to protect users from cyberattacks by identifying phishing websites. KNN proved to be the most effective model, achieving the highest accuracy, while Decision Tree and Logistic Regression performed well but slightly lagged. Naive Bayes struggled due to its limitations. This project highlights the potential of machine learning in enhancing cybersecurity and safeguarding users' sensitive information.