National College of Ireland

Configuration Manual

Protection Against Spear Phishing Attacks Using the Ensemble Method of Machine Learning

Jecinta Ifechukwu Fidelis
Student ID: 23148306

School of Computing
National College of Ireland

Supervisor:Imran Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Jecinta Ifechukwu Fidelis …….
Student ID:	23148306
Programme:	Msc in Cyber Security …… **Year:** 2024 …..
Module:	Msc Research Practicum
Lecturer:	Imran Khan
Submission Due Date:	10/08/2024
Project Title:	
Word Count:	…… **Page Count:**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Jecinta Ifechukwu Fidelis ……
Date:	10/08/2024 ……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Jecinta Ifechukwu Fidelis
23148306

1. Introduction

This research utilizes Google Colab, a robust cloud-based platform, to facilitate the development and execution of complex machine learning models. Google Colab provides complimentary access to both CPU and GPU resources, making it an invaluable tool for handling intensive computational tasks associated with machine learning. Additionally, the platform supports efficient management and backup of datasets and codebases, ensuring data integrity and continuity throughout the research process. Google Colab also comes pre-equipped with numerous machine learning libraries, streamlining the implementation process by reducing the need for extensive setup, thereby enabling quicker and more efficient project execution.

2. System Specification

The system configuration used in this project are:

- Operating system: Linux
- Processor: Intel Xeon CPU (2)
- RAM: 13GB

This configuration provides the necessary computational power to efficiently process and analyze large datasets, essential for training and testing the machine learning models employed in this research.

3. Software Tools

The primary software tools used in the implementation of this project are centered around the Python programming language, with Google Colab serving as the development environment. The following tools were integral to the project's success:

- **Python:** The core programming language used for developing machine learning models.
- **Google Colab:** The cloud-based platform that facilitated code execution, model training, and deployment.
- **Jupyter Notebook:** Used within Google Colab for interactive code development and testing.

These tools provided a cohesive environment for the development, testing, and deployment of machine learning models.

4. Implementation

The implementation of this project involved several key Python libraries that provided the necessary functionality for data manipulation, model development, and evaluation:

- **Numpy:** Used for numerical computations and data manipulation.

- **Pandas:** Provided tools for data analysis and manipulation, particularly for handling data frames.
- **Matplotlib:** Facilitated the creation of visualizations for data analysis and model evaluation.
- **Sklearn (Scikit-learn):** A machine learning library used for model training, evaluation, and feature extraction.
- **NLTK (Natural Language Toolkit):** Employed for text preprocessing tasks, including tokenization, stop word removal, and lemmatization.
- **LightGBM:** A gradient boosting framework that was utilized for efficient model training and prediction.
- **XGBoost:** Another gradient boosting library, used alongside LightGBM for model comparison.

These libraries formed the backbone of the project's implementation, enabling sophisticated data processing and model training. Figure Descriptions:

Fig 1: Demonstrates how the Google Colab environment was set up by mounting the necessary drives.
Fig 2: Illustrates the process of importing the required Python libraries.
Fig 3: Shows the function used to calculate and display evaluation metrics for the models.
Fig 4: Depicts the function created for visualizing the confusion matrix, an important tool for evaluating model performance.

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

Fig 1: Mounting the drive in google colab

```
1 ### Libararies for data pre-processing and Visualization
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 import json
7 import os
8 from collections import Counter
9
10 ### Natural Language Processing Libraries
11 import re
12 from nltk.corpus import stopwords
13 from nltk.stem import WordNetLemmatizer
14 from collections import Counter
15 import nltk
16 # Download necessary NLTK data files
17 nltk.download('stopwords')
18 nltk.download('wordnet')
19
20 ## Feature Extraction
21 from sklearn.feature_extraction.text import TfidfVectorizer
22
23 ### Machine Learning Libararies
24 from sklearn.model_selection import train_test_split
25 from sklearn.metrics import accuracy_score, f1_score, matthews_corrcoef, precision_score, recall_score,confusion_matrix
26 from sklearn.linear_model import LogisticRegression
27 from sklearn.naive_bayes import MultinomialNB
28 from sklearn.tree import DecisionTreeClassifier
29 from sklearn.ensemble import RandomForestClassifier
30 import xgboost as xgb
31 import lightgbm as lgb
```

Fig 2: Importing Libraries

```python
def evaluate_model_performance(y_true, y_pred):

    metrics = {
        'accuracy': accuracy_score(y_true, y_pred),
        'f1_score': f1_score(y_true, y_pred, average='weighted'),
        'mcc': matthews_corrcoef(y_true, y_pred),
        'precision': precision_score(y_true, y_pred, average='weighted'),
        'recall': recall_score(y_true, y_pred, average='weighted')
    }

    return metrics
```

Fig 3: Evaluation Metrics function

```python
def plot_confusion_matrix(y_true, y_pred, class_names=None):
    cm = confusion_matrix(y_true, y_pred)
    if class_names is None:
        class_names = [str(i) for i in range(cm.shape[0])]

    fig, ax = plt.subplots(figsize=(8, 6))
    im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    ax.figure.colorbar(im, ax=ax)

    # We want to show all ticks and label them with the respective list entries
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           xticklabels=class_names, yticklabels=class_names,
           title='Confusion Matrix',
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    plt.show()
```

Fig 4: Confusion Matrix visualization function

1 Data Gathering

The process of data gathering involved collecting data points that were instrumental in distinguishing between normal and phishing emails. The steps included accessing the dataset path on Google Drive, creating a function to read each JSON file, converting these files into Pandas data frames, and labeling the data accordingly. Specifically, a new column labeled "Label" was added to identify each email as normal or phishing. The dataset was further refined by removing insignificant columns, such as 'id' and 'sender name,' as shown in the figures below.

Fig 5: Illustrates the process of reading and processing the datasets.
Fig 6: Shows the data gathering process, including the labeling and cleaning of the dataset.

```
path = '/content/drive/MyDrive/Email-project'
```

```
def json_files_to_dataframe(directory):
    # Initialize an empty list to hold the data from each file
    data_list = []

    # Loop through each file in the directory
    for filename in os.listdir(directory):
        if filename.endswith('.json'):
            file_path = os.path.join(directory, filename)

            # Open and read the JSON file
            with open(file_path, 'r') as file:
                data = json.load(file)

                # Append the data to the list
                data_list.append(data)

    # Create a DataFrame from the list of data
    df = pd.DataFrame(data_list)
    return df
```

```
## The path of the dataset
normal_path = path + '/enron_ham'
spam_path = path + '/spear_phishing'
```

```
### Normal (Non-Spam) Email
normal_df = json_files_to_dataframe(normal_path)
### SPam Email
spam_df = json_files_to_dataframe(spam_path)
```

Fig 5: Reading the datasets

```
### Normal Email Target
normal_df['Label'] = 'normal'
```

```
### Phising Email Target
spam_df['Label'] = 'pishing'
```

```
data = pd.concat([normal_df, spam_df]).reset_index(drop=True)
```

```
## Drop insignificant features
data = data.drop(['id','sender_name'], axis=1)
```

Fig 6: Data Gathering and Labelling

2 Data Understanding

Understanding the dataset involved gaining insights into its structure and composition. The dataset's basic characteristics, such as its head, shape, and label frequency, were examined. The dataset comprises 3,334 rows and three features, including the target feature (label). It was noted that approximately 90% of the emails were normal, while 10% were phishing. These insights were critical for guiding the subsequent data processing steps.

- **Fig 7:** Provides information about the dataset, including the data types of each feature.
- **Fig 8:** Displays the distribution of the target feature, highlighting the imbalance between normal and phishing emails.

```
 1 data.shape
```

```
(3334, 3)
```

```
 1 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3334 entries, 0 to 3333
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   email_subject  3334 non-null   object
 1   email_body     3334 non-null   object
 2   Label          3334 non-null   object
dtypes: object(3)
memory usage: 78.3+ KB
```

```
 1 data['Label'].value_counts(normalize=True)*100
```

```
Label
normal     89.982004
pishing    10.017996
Name: proportion, dtype: float64
```

Fig 7: Data Info

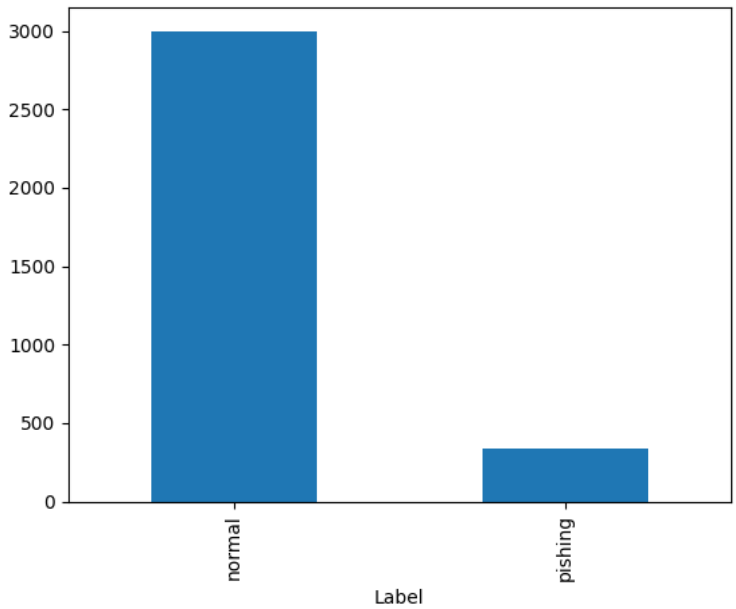


Fig 8: Target feature distribution

3 Data Pre-processing

Data pre-processing is a critical step where the text data from email bodies, which serves as the feature (X) in the dataset, is cleaned and prepared for analysis. This involved applying functions for stop-word removal and lemmatization, converting text to lowercase, and removing unnecessary elements such as URLs, punctuation, special characters, digits, and repetitive words. The pre-processing steps were designed to enhance the quality of the text data, making it more suitable for machine learning model training.

- **Fig 9:** Shows the pre-processing steps applied to the text data, illustrating the improvement in data quality.

```
# Initialize stopwords and lemmatizer
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
```

```
top_10_words = ["subject", "cc", "please", "phillip", "pm", "email", "k", "forwarded", "would", "john"]
underscore_patterns = ['____', '_____', '______']

def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    # Remove punctuation, special characters, and numbers
    text = re.sub(r'\n', ' ', text)  # Replace newline characters with space
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\d+', '', text)  # Remove numbers
    # Remove underscore patterns
    for pattern in underscore_patterns:
        text = text.replace(pattern, '')
    # Remove stopwords and top 10 words
    text = ' '.join([word for word in text.split() if word not in stop_words and word not in top_10_words])
    # Remove repeated words
    text = ' '.join(sorted(set(text.split()), key=text.split().index))
    # Lemmatize words
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split()])

    return text
```

```
data['clean_email_body'] = data['email_body'].apply(preprocess_text)
```

Fig 9: Text Data Pre-processing

4 Data Visualization and Data Transformation

Data visualization was used to identify the most frequent words in the cleaned email bodies, while data transformation involved converting the labels into binary values (0 for normal emails and 1 for phishing emails). The transformed dataset, consisting of the "clean email body" and "Label" features, was then prepared for building the machine learning models.

- **Fig 10:** Visualizes the top words in the cleaned dataset.
- **Fig 11:** Lists the top 10 words, showing their frequency.
- **Fig 12:** Demonstrates the data transformation process, preparing the dataset for model training.

```python
# Combining all email bodies into a single string
all_text = ' '.join(data['clean_email_body'])
# Splitting the text into words
words = all_text.split()

# Counting the frequency of each word
word_counts = Counter(words)
# Getting the top 10 most common words
top_10_words = word_counts.most_common(10)
words, counts = zip(*top_10_words)

# Creating the bar chart
plt.figure(figsize=(10, 5))
plt.bar(words, counts, color='skyblue')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Top 10 Words in Email Bodies')
plt.xticks(rotation=45)
plt.show()
```

Fig 10: Data Visualization

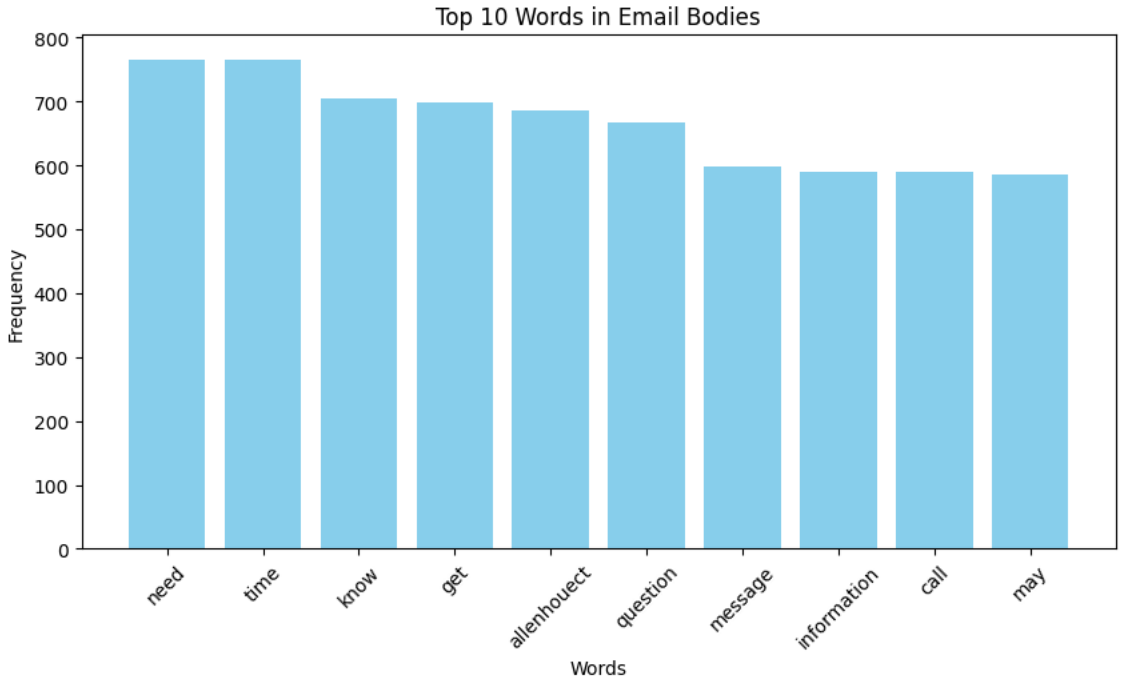


Fig 11: Top 10 words

```python
data['Label'] = data['Label'].replace(to_replace=['normal', 'pishing'], value=[0, 1])
```

```python
data_new = data[['clean_email_body', 'Label']]
```

Fig 12: Data Transformation

5 Data Splitting

The dataset was split into training and testing sets using Scikit-learn's train-test split function. This stratified approach ensured that the label distribution in both the training and testing sets mirrored the original dataset. The training set comprised 70% of the data, while the testing set included the remaining 30%, with a random seed of 42 used for reproducibility.

- **Fig 13:** Illustrates the data splitting process.

```
X = data_new['clean_email_body']
y = data_new['Label']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, shuffle=True,stratify=y)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2333,)
(1001,)
(2333,)
(1001,)
```

Fig 13: Data Splitting

6 Feature Extraction

Feature extraction involved converting the text data into numerical vectors using TF-IDF (Term Frequency-Inverse Document Frequency). This technique was implemented using Scikit-learn libraries and involved setting parameters to filter text, remove rare terms, and handle common words effectively. The TF-IDF was fitted to the training data and subsequently applied to both the training and testing datasets.

- **Fig 14:** Shows the feature extraction process using TF-IDF.

```
tfv = TfidfVectorizer(min_df=5,  max_features=None,
            strip_accents='unicode', analyzer='word',token_pattern=r'\w{1,}',
            ngram_range=(1, 4), use_idf=1,smooth_idf=1,sublinear_tf=1,
            stop_words = 'english')
```

```
# Fitting TF-IDF to both training and test sets
tfv.fit(X_train)
X_train_tfv =  tfv.transform(X_train)
X_test_tfv = tfv.transform(X_test)
```

Fig 14. Feature Extraction using Tf-idf

7 Machine Learning Model

Various Machine learning models are used to predict the test set.

7.1 Logistics Regression

The logistics regression confusion matrix shows that we have two false negatives.

```
lr_model = LogisticRegression() #intialize the logistic regression model
lr_model.fit(X_train_tfv, y_train)# fit the logistic regression model to the training data
predictions = lr_model.predict(X_test_tfv) # make prediction on the test data
lr_eval = evaluate_model_performance(y_test, predictions)# evaluate the performance of the model
plot_confusion_matrix(y_test, predictions, class_names=['normal', 'pishing']) ## Confusion Matrix (TP, FP, FN, TN)
```

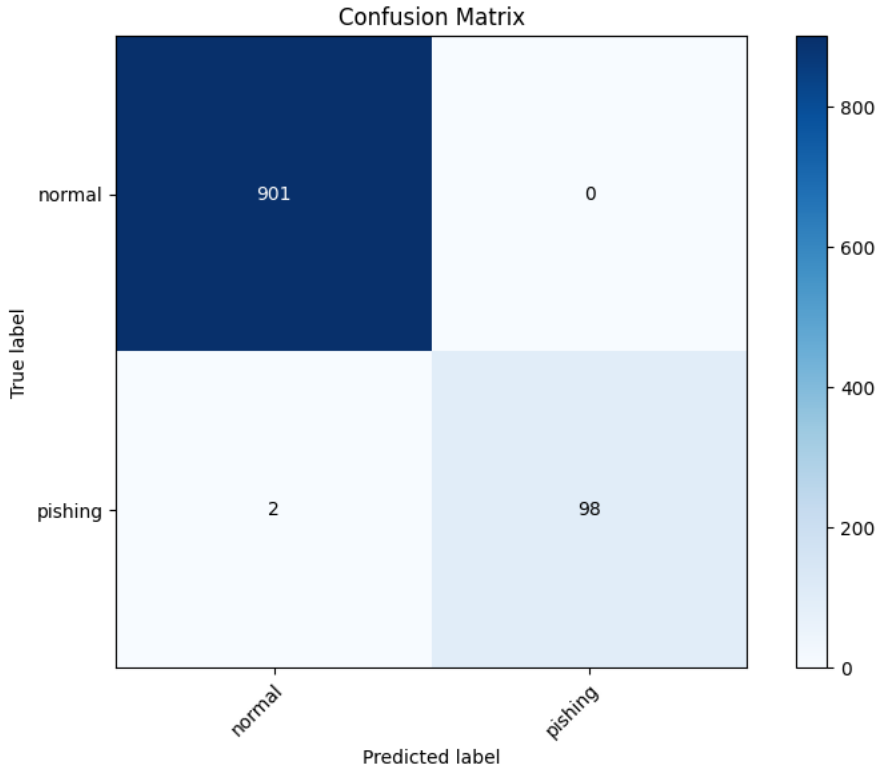


Fig. 15: Logistics Regression Confusion Matrix

7.2 Naïve Bayes

The naïve bayes model confusion matrix shows that we have eight false positives.

```
nb_model = MultinomialNB()
nb_model.fit(X_train_tfv, y_train)
predictions = nb_model.predict(X_test_tfv)
nb_eval = evaluate_model_performance(y_test, predictions)
plot_confusion_matrix(y_test, predictions, class_names=['normal', 'pishing']) ## The TP, FP, FN, TN
```

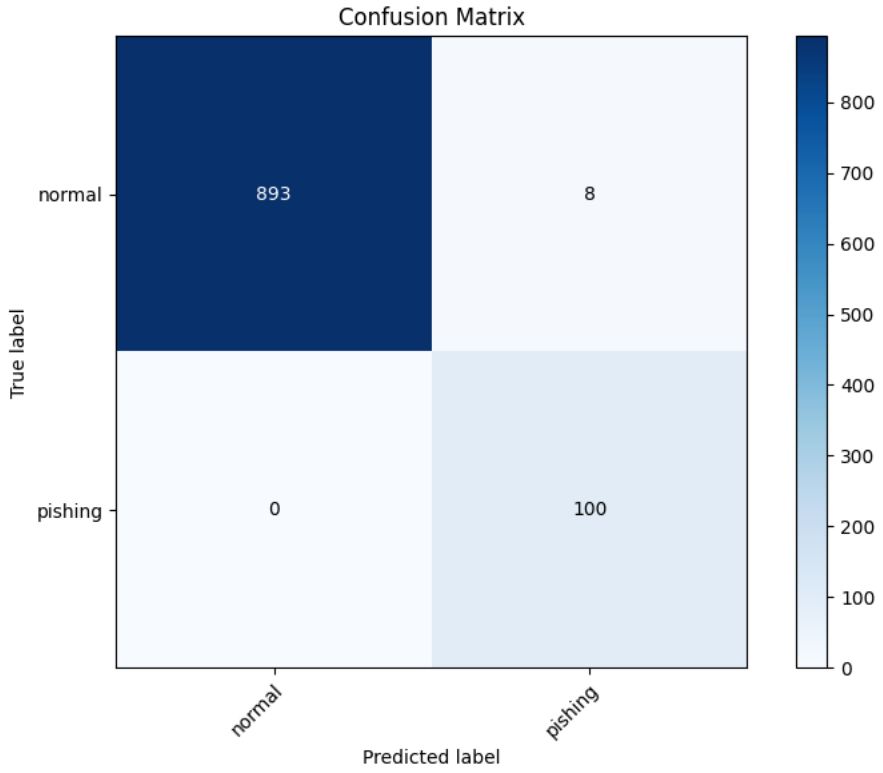


Fig. 16: Naïve Bayes Confusion Matrix

7.3 Decision Tree

The decision tree model confusion matrix shows that we have one false negative.

```
dtr_model = DecisionTreeClassifier(random_state=101)
dtr_model.fit(X_train_tfv, y_train)
predictions = dtr_model.predict(X_test_tfv)
dtr_eval = evaluate_model_performance(y_test, predictions)
plot_confusion_matrix(y_test, predictions, class_names=['normal', 'pishing']) ## The TP, FP, FN, TN
```

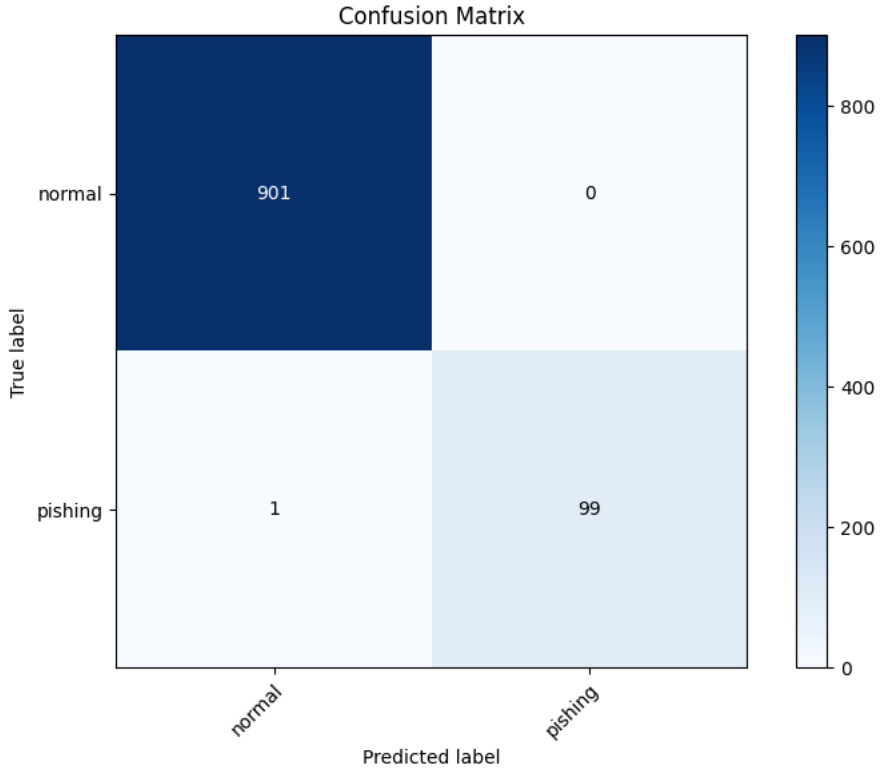


Fig. 17: Decision Tree Confusion Matrix

7.4 Random Forest

The Random Forest model confusion matrix shows that we have three false negative.

```
rf_model = RandomForestClassifier(random_state=0, max_depth=7) ## Restricting the Max depth to handle the Model overfitting
rf_model.fit(X_train_tfv, y_train)
predictions = rf_model.predict(X_test_tfv)
rf_eval = evaluate_model_performance(y_test, predictions)
plot_confusion_matrix(y_test, predictions, class_names=['normal', 'pishing']) ## The TP, FP, FN, TN
```

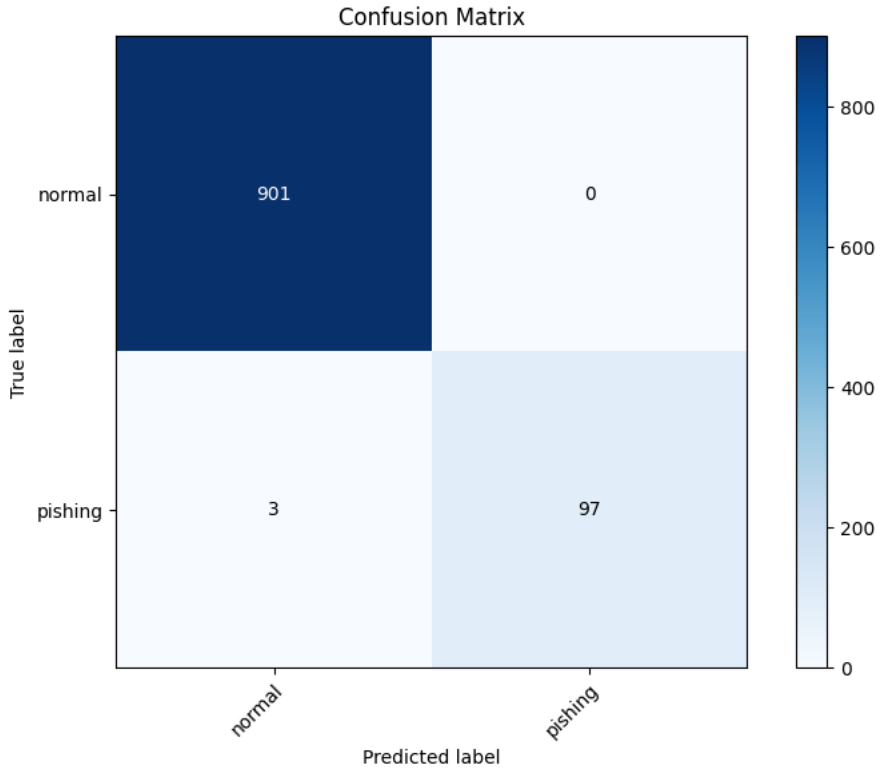


Fig. 18: Random Forest Confusion Matrix

7.5 Xgboost

The Xgboost model confusion matrix shows that we have one false negative.

```
1 xgb_model = xgb.XGBClassifier()
2 xgb_model.fit(X_train_tfv, y_train)
3 predictions = xgb_model.predict(X_test_tfv)
4 xgb_eval = evaluate_model_performance(y_test, predictions)
5 plot_confusion_matrix(y_test, predictions, class_names=['normal', 'pishing']) ## The TP, FP, FN, TN
```

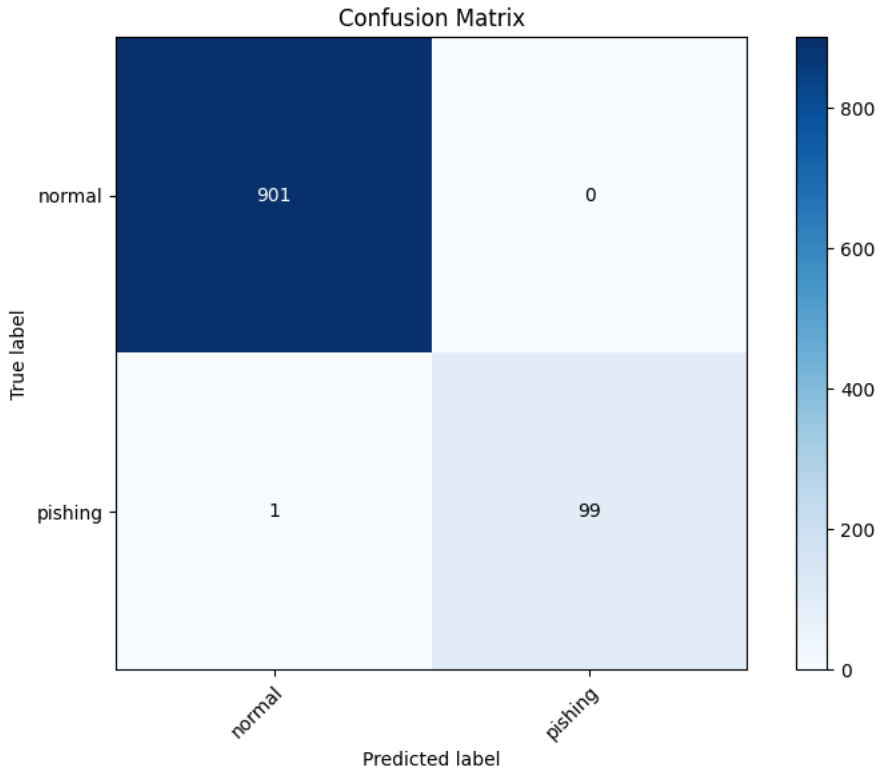


Fig. 19: Xgboost Confusion Matrix

7.6 LightGbm

The Lightgbm model confusion matrix shows that we have one false negative.

```
lgb_model = lgb.LGBMClassifier(silent = True)
lgb_model.fit(X_train_tfv, y_train)
predictions = lgb_model.predict(X_test_tfv)
lgb_eval = evaluate_model_performance(y_test, predictions)
plot_confusion_matrix(y_test, predictions, class_names=['normal', 'pishing']) ## The TP, FP, FN, TN
```

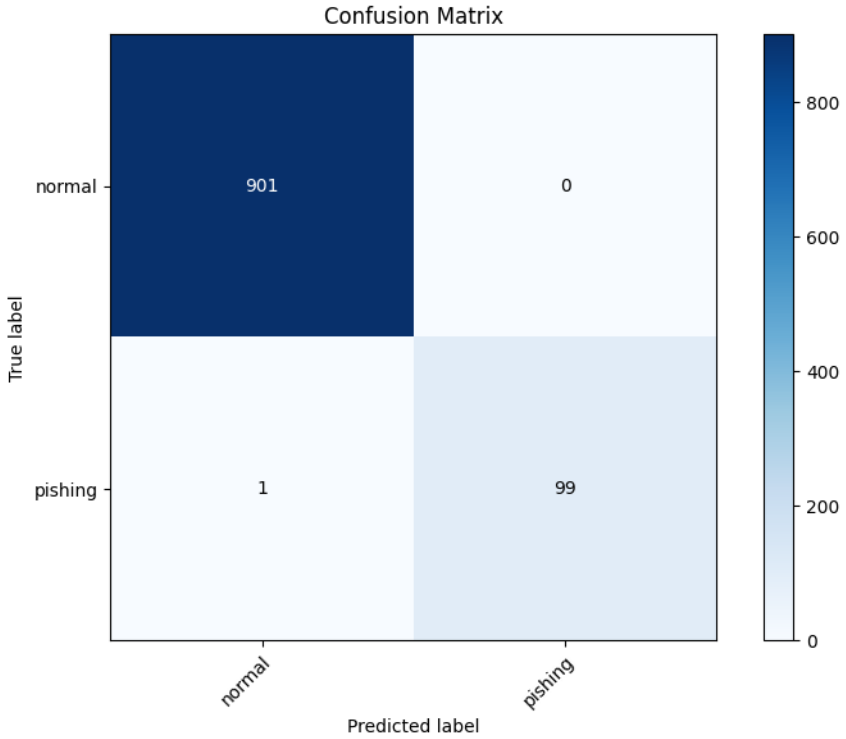


Fig. 20: LightGbm Confusion Matrix

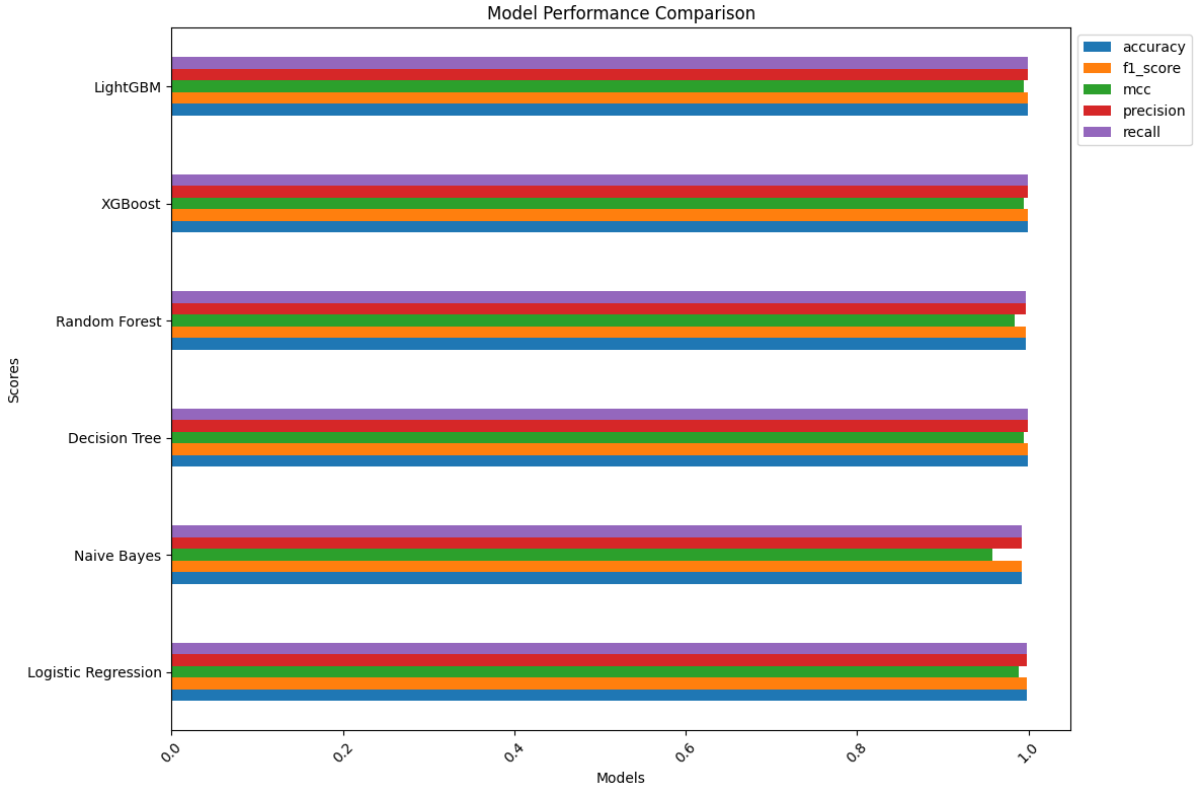
Results and Evaluation

The XGBoost and LightGBM models emerged as the top performers, each achieving an accuracy rate of 99.91%. These models also scored the highest across various evaluation metrics, indicating their effectiveness in classifying phishing emails accurately.

Fig 21: Comparison of model performance results.

	Model	accuracy	f1_score	mcc	precision	recall
0	Logistic Regression	0.998002	0.997993	0.988853	0.998006	0.998002
1	Naive Bayes	0.992008	0.992144	0.957969	0.992600	0.992008
2	Decision Tree	0.999001	0.998999	0.994436	0.999002	0.999001
3	Random Forest	0.997003	0.996983	0.983250	0.997013	0.997003
4	XGBoost	0.999001	0.998999	0.994436	0.999002	0.999001
5	LightGBM	0.999001	0.998999	0.994436	0.999002	0.999001

Fig. 21: Models Comparison results



Email Prediction Interface

The final model was saved using a joblib file, allowing it to be utilized in real-time to predict whether an email is normal or phishing. The interface, built with Gradio, enables users to classify emails interactively.

- **Fig 22:** Shows the real-time email classification interface.
- **Fig 23:** Demonstrates the saved machine learning model and feature extraction function.

The trained model was tested with new email inputs to validate its performance. The results indicated whether the emails were normal or phishing.

- **Fig 24:** Interface displaying a phishing email prediction.
- **Fig 25:** Interface displaying a normal email prediction.

```python
def classify_email(model_path, vectorizer_path, email_text):
    """
    Classifies an input email text to determine if it is a phishing email or not.

    Parameters:
    model_path (str): Path to the saved model.
    vectorizer_path (str): Path to the saved vectorizer.
    email_text (str): The email text to be classified.

    Returns:
    str: 'Phishing' if the email is classified as phishing, 'Not Phishing' otherwise.
    """
    # Load the saved model and vectorizer
    model_saved = joblib.load(model_path)
    vector_saved = joblib.load(vectorizer_path)

    # Preprocess the text
    clean_sample_text = preprocess_text(email_text)

    # Transform the text using the vectorizer
    sample_vec = vector_saved.transform([clean_sample_text])

    # Predict using the model
    prediction = model_saved.predict(sample_vec)

    # Map the prediction to a human-readable label
    if prediction[0] == 1:
        return 'Phishing'
    else:
        return 'Normal'
```

Fig. 22: Email classify near real time prediction

```python
model_saved = joblib.load("lgb_model.joblib")
vector_saved = joblib.load("tfv.joblib")
```

Fig. 23: The saved ML model and feature extraction function

The feature extraction model (TF-IDF) and the saved machine learning model. Whether an email is normal or phishing, the model is utilized to ascertain its input. The email message is phishing, as shown in Fig. 24. When we run the cell and enter a new email message, the text input message will refresh.

Gradio was used to develop the user interface for the email classification system. Its ease of integration with machine learning models and intuitive design features made it an excellent choice for creating a user-friendly interface.

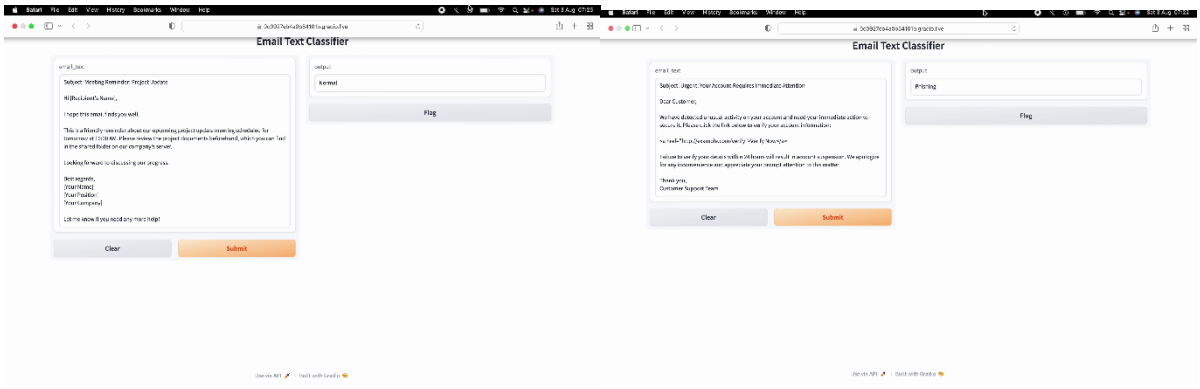


Fig. 24: Phishing Email interface

The architecture comprises several components: user interface, data pre-processing, feature extraction, model training and evaluation, deployment environment, and real-time classification. The system architecture diagram is presented in Figure 25 below.

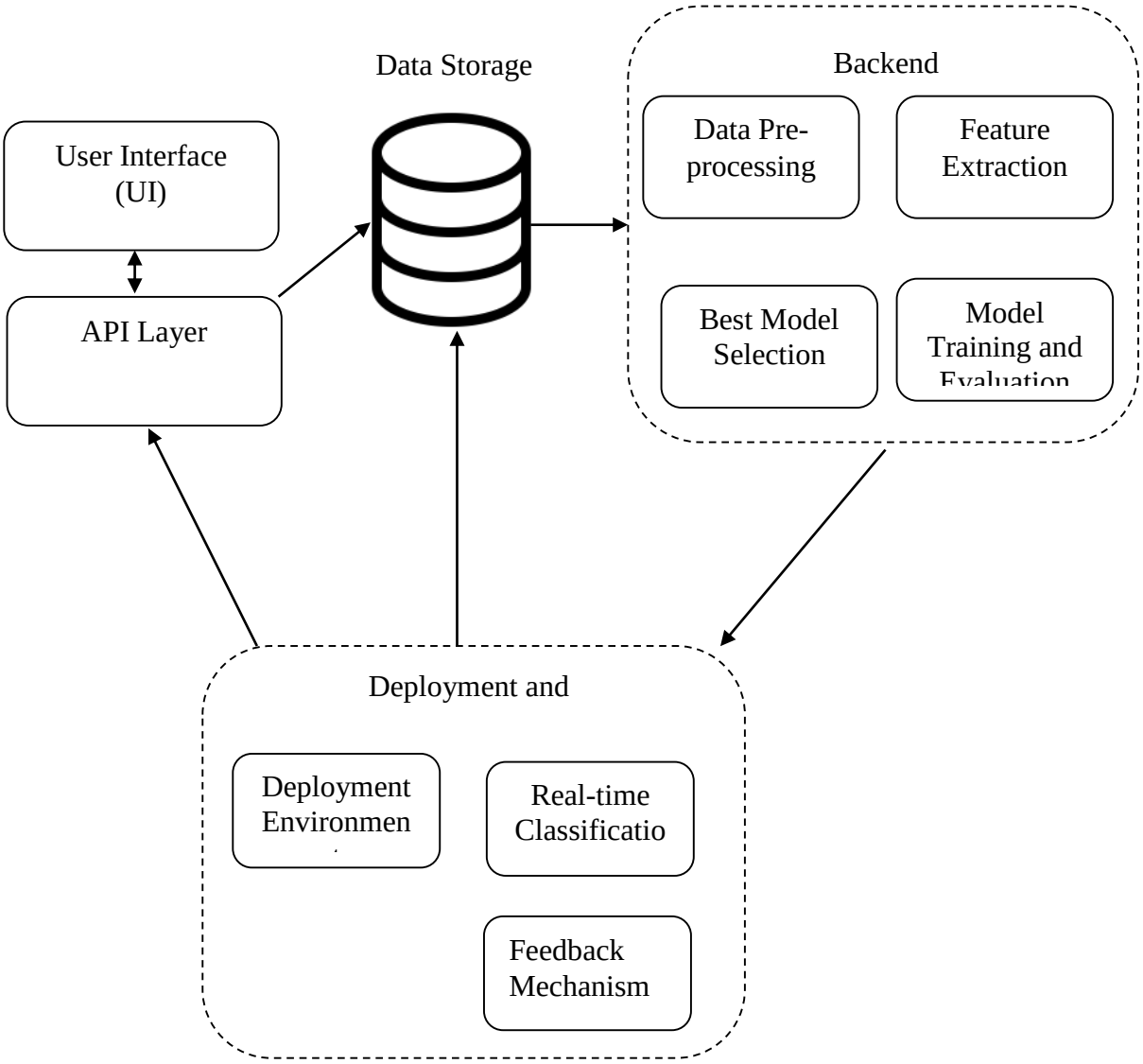


Figure 25: Phishing email classification system architecture

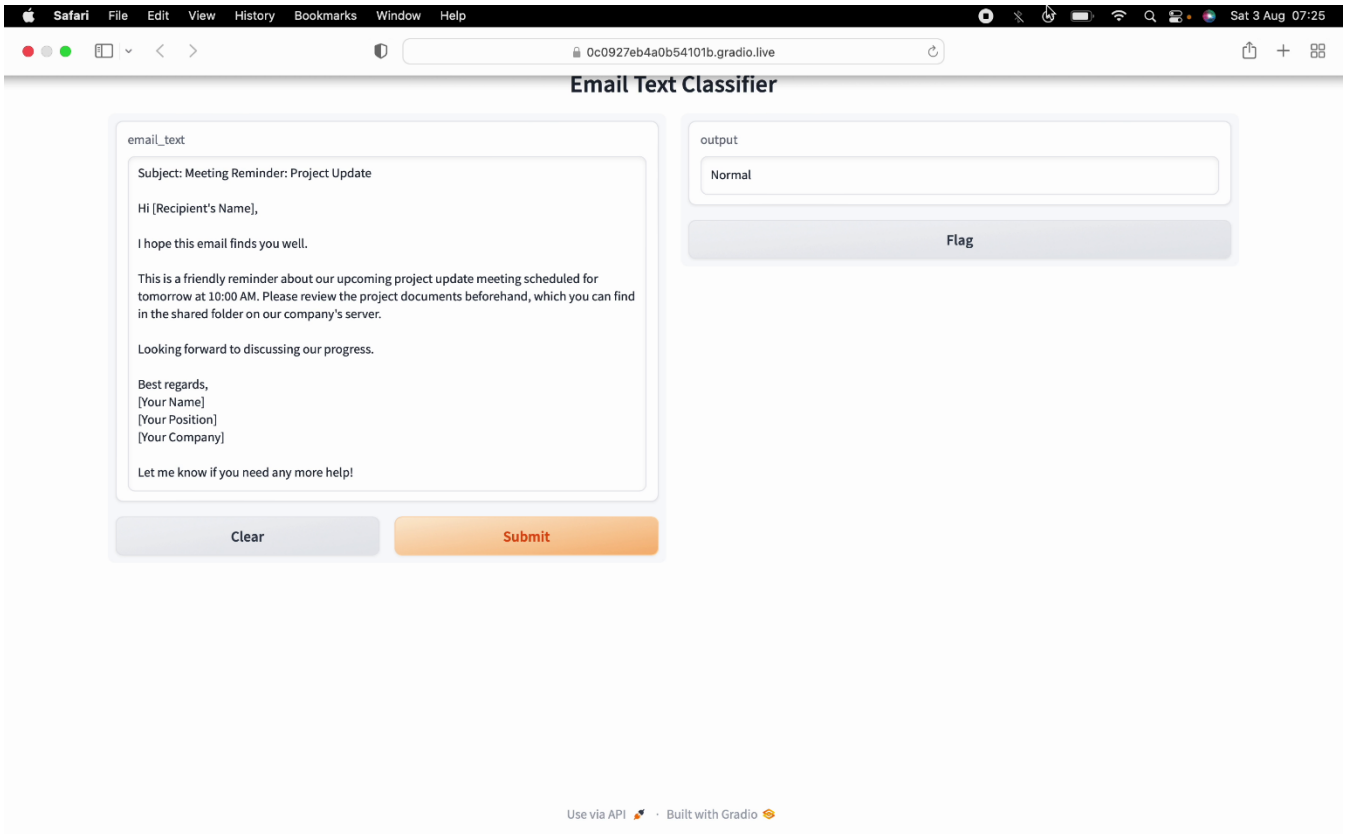
User Interface

The system leverages Gradio to create an intuitive web-based interface, allowing users to input email text, submit it for analysis, and receive classification results immediately. Gradio is an open-source library that simplifies the process of building and deploying user interfaces for machine learning models. It enables developers to create interactive and accessible interfaces without requiring extensive web development experience, making it an ideal choice for the email classification system. A key advantages of using Gradio is its ease of integration with machine learning models. With Gradio, developers can rapidly prototype and deploy

applications by linking pre-trained models directly to user interfaces. This capability is particularly valuable in the email classification system, where the model's ability to process and classify data in real-time is crucial for user satisfaction.

The UI includes features such as:

- **Text Input Field:** Users can enter email content directly into the system.
- **Submit Button:** Initiates the classification process, sending the text to the backend model for analysis.
- **Clear Button:** Resets the input field for new entries.
- **Output Display:** Shows the classification result (e.g., "Phishing" or "Non-Phishing") along with processing time.
- **Feedback Mechanism:** Users can flag emails for further review or feedback, enabling continuous improvement of the model. This integration of user feedback allows the system to learn from real-world usage and refine its detection capabilities.



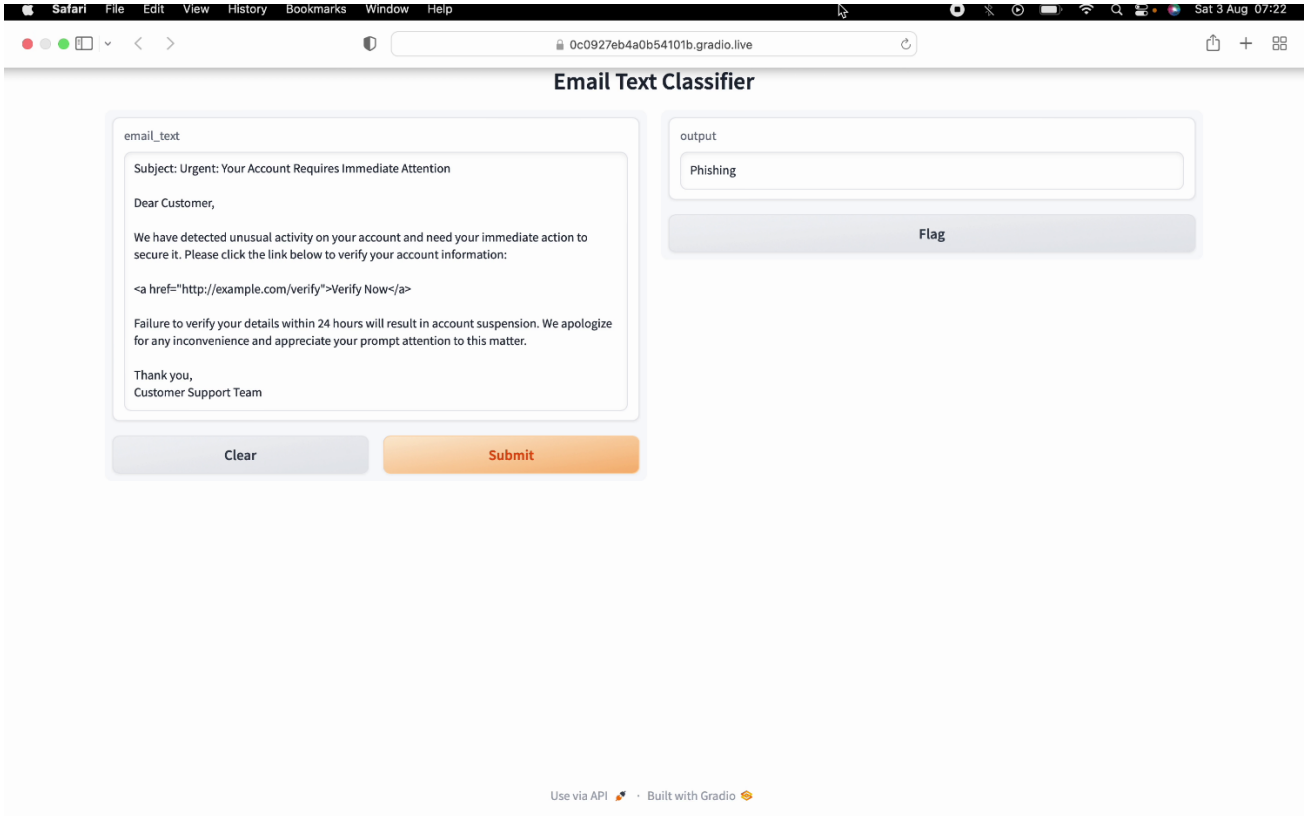


Figure 26: User evaluation with normal and phishing emails