Phishing Simulation and Detection System Documentation

Project Overview

The Phishing Simulation and Detection System aims to simulate phishing attacks and detect phishing emails using machine learning and natural language processing techniques. This system includes sending and receiving emails, processing and extracting features from email text, training a machine learning model, and using the trained model to detect phishing emails.

Project Structure and Explanation

```
phishing_detection/
├── data/
│   ├── emails.csv
├── src/
│   ├── __init__.py
│   ├── data_preprocessing.py
│   ├── email_protocols.py
│   ├── feature_extraction.py
│   ├── model_training.py
│   ├── phishing_detection.py
├── tests/
│   ├── test_email_protocols.py
│   ├── test_model_training.py
│   ├── test_phishing_detection.py
├── main.py
├── requirements.txt
├── README.md
```

Explanation of the Flowchart

1. **Start**: The starting point of the system.
2. **Load Data from emails.csv**: Load the email dataset from the CSV file located in the data directory.
3. **Preprocess Data**: Clean and normalize the email text data by removing HTML tags, converting text to lowercase, etc.
4. **Extract Features using TF-IDF**: Convert the preprocessed text data into numerical features using the TF-IDF (Term Frequency-Inverse Document Frequency) method.
5. **Train Model with Naive Bayes**: Train a Naive Bayes model using the extracted features and their corresponding labels.
6. **Evaluate Model**: Evaluate the trained model on a test set to check its accuracy and performance.
7. **Receive New Emails**: Retrieve new emails from the inbox using IMAP.
8. **Detect Phishing using Trained Model**: Use the trained model to classify new emails as phishing or legitimate.
9. **Is it Phishing?**: A decision point to check if the email is classified as phishing.
10. **Phishing Alert**: If the email is classified as phishing, generate an alert.
11. **Legitimate Email**: If the email is classified as legitimate, proceed without alert.
12. **Send Simulated Phishing Emails**: Send simulated phishing emails using SMTP for testing and training purposes.
13. **Loop Back to Start**: The process repeats as new emails are received and processed.

Detailed Explanation of Each File

1. data/emails.csv

Purpose:

- Contains the email dataset with columns such as text and label.

Significance:

- This dataset is used for training and testing the machine learning model to distinguish between phishing and legitimate emails.

2. src/__init__.py

Purpose:

- This file makes the src directory a Python package.

Significance:

- Allows the directory to be treated as a package, enabling imports from other scripts within the src directory.

3. src/data_preprocessing.py

Code:

```
import pandas as pd

def load_data(filepath):
    return pd.read_csv(filepath)

def preprocess_text(text):
    # Implement text preprocessing (e.g., removing HTML tags, lowercasing)
    text = text.str.replace('<[^<]+?>', '')  # Remove HTML tags
    text = text.str.lower()  # Convert to lowercase
    return text
```

Purpose:

- Contains functions for loading and preprocessing data.

Significance:

- Prepares the raw data for feature extraction and model training by cleaning and normalizing the text.

Explanation:

- load_data(filepath): Reads a CSV file and returns a DataFrame.
- preprocess_text(text): Cleans the text data by removing HTML tags and converting to lowercase, which helps in normalizing the text for better feature extraction.

4. src/email_protocols.py

Code:

```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import imaplib
import email

def send_email(sender_email, receiver_email, subject, body, smtp_server, port, login, password):
    msg = MIMEMultipart()
    msg['From'] = sender_email
    msg['To'] = receiver_email
    msg['Subject'] = subject
    msg.attach(MIMEText(body, 'plain'))
    with smtplib.SMTP(smtp_server, port) as server:
        server.starttls()
        server.login(login, password)
        server.sendmail(sender_email, receiver_email, msg.as_string())

def receive_emails(email_account, password, imap_server):
    mail = imaplib.IMAP4_SSL(imap_server)
    mail.login(email_account, password)
    mail.select('inbox')
```

```
    status, messages = mail.search(None, 'ALL')
    email_ids = messages[0].split()
    emails = []
    for e_id in email_ids:
        status, msg_data = mail.fetch(e_id, '(RFC822)')
        msg = email.message_from_bytes(msg_data[0][1])
        emails.append(msg)
    return emails
```

Purpose:

- Functions for sending and receiving emails.

Significance:

- Simulates phishing attacks by sending emails and retrieves emails for analysis and detection.

Explanation:

- send_email(sender_email, receiver_email, subject, body, smtp_server, port, login, password): Constructs and sends an email using SMTP.
- receive_emails(email_account, password, imap_server): Connects to an IMAP server to fetch emails from the inbox.

5. src/feature_extraction.py

Code:

```
from sklearn.feature_extraction.text import TfidfVectorizer

def extract_features(train_texts, test_texts):
    vectorizer = TfidfVectorizer(stop_words='english')
    X_train_tfidf = vectorizer.fit_transform(train_texts)
    X_test_tfidf = vectorizer.transform(test_texts)
    return X_train_tfidf, X_test_tfidf, vectorizer
```

Purpose:

- Functions for extracting features from email text using TF-IDF.

Significance:

- Converts email text into numerical features that can be used for training the machine learning model.

Explanation:

- extract_features(train_texts, test_texts): Uses TF-IDF to convert text data into numerical form, which can be fed into a machine learning model. It returns the transformed training and test data along with the vectorizer.

6. src/model_training.py

Code:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

def train_model(X_train, y_train):
    model = MultinomialNB()
    model.fit(X_train, y_train)
    return model

def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    print('Accuracy:', accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))
```

Purpose:

- Functions for training and evaluating the machine learning model.

Significance:

- Builds a model to classify emails as phishing or legitimate and assesses its performance.

Explanation:

- `train_model(X_train, y_train)`: Trains a Multinomial Naive Bayes model on the training data.
- `evaluate_model(model, X_test, y_test)`: Evaluates the model on test data and prints the accuracy and classification report.

7. src/phishing_detection.py

Code:

```
def detect_phishing(email_text, vectorizer, model):
    email_tfidf = vectorizer.transform([email_text])
    prediction = model.predict(email_tfidf)
    return 'Phishing' if prediction == 1 else 'Legitimate'
```

Purpose:

- Function for detecting phishing emails using the trained model.

Significance:

- Applies the trained model to new emails to identify phishing attempts in real-time.

Explanation:

- `detect_phishing(email_text, vectorizer, model)`: Transforms the email text using the vectorizer and uses the trained model to predict whether it is phishing or legitimate.

8. tests/test_email_protocols.py

Code:

```
import unittest
from src.email_protocols import send_email, receive_emails

class TestEmailProtocols(unittest.TestCase):
    def setUp(self):
        self.sender_email = "your_email@example.com"
        self.receiver_email = "recipient@example.com"
        self.subject = "Test Email"
        self.body = "This is a test email."
        self.smtp_server = "smtp.example.com"
        self.port = 587
        self.login = "your_login"
        self.password = "your_password"
        self.imap_server = "imap.example.com"

    def test_send_email(self):
        try:
            send_email(
                self.sender_email,
                self.receiver_email,
                self.subject,
                self.body,
                self.smtp_server,
                self.port,
                self.login,
                self.password
            )
        except Exception as e:
            self.fail(f"send_email() raised an exception: {e}")

    def test_receive_emails(self):
        try:
            emails = receive_emails(
                self.sender_email,
                self.password,
                self.imap_server
            )
            self.assertIsInstance(emails, list)
        except Exception as e:
            self.fail(f"receive_emails() raised an exception: {e}")

if __name__ == '__main__':
    unittest.main()
```

Purpose:

- Tests for email sending and receiving functions.

Significance:

- Ensures the email protocols are working correctly.

Explanation:

- test_send_email(): Verifies that the send_email function executes without raising exceptions.
- test_receive_emails(): Verifies that the receive_emails function executes and returns a list of emails.

9. tests/test_model_training.py

Code:

```
import unittest
import pandas as pd
from sklearn.model_selection import train_test_split
from src.data_preprocessing import preprocess_text
from src.feature_extraction import extract_features
from src.model_training import train_model, evaluate_model

class TestModelTraining(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.df = pd.DataFrame({
            'text': [
                'This is a legitimate email.',
                'Free money!!! Click here to claim your prize.',
                'Your account has been compromised. Update your password.',
                'Meeting schedule for next week.',
                'Congratulations! You won a lottery. Click to claim.'
            ],
            'label': [0, 1, 1, 0, 1]
        })
        cls.df['text'] = preprocess_text(cls.df['text'])
        cls.X_train, cls.X_test, cls
```


Model Selection and Justification

In the Phishing Simulation and Detection System, the Multinomial Naive Bayes model was chosen for the task of classifying emails as phishing or legitimate. This decision was made based on several factors, including the nature of the data, the performance of various models during preliminary testing, and the computational efficiency of the model.

Factors Considered for Model Selection

1. **Nature of the Data**:
    - The data consists of email texts, which are categorical in nature. Naive Bayes models, particularly Multinomial Naive Bayes, are well-suited for text classification tasks as they work effectively with the frequency of word occurrences.
2. **Performance during Preliminary Testing**:
    - Several models were evaluated, including Logistic Regression, Support Vector Machines (SVM), Random Forest, and Multinomial Naive Bayes. The performance metrics used for evaluation were accuracy, precision, recall, and F1-score.
3. **Computational Efficiency**:

- Multinomial Naive Bayes is computationally efficient, making it suitable for large datasets. It requires less training time compared to more complex models like SVM and Random Forest.

Model Evaluation

During the model evaluation phase, the following models were tested:

1. **Logistic Regression**:
   - Pros: Good performance for binary classification, interpretable coefficients.
   - Cons: Slower training on large datasets compared to Naive Bayes, might overfit on smaller datasets.
2. **Support Vector Machines (SVM)**:
   - Pros: High accuracy, especially for text classification tasks.
   - Cons: Computationally expensive, slower training time, requires careful tuning of hyperparameters.
3. **Random Forest**:
   - Pros: Robust to overfitting, can handle large datasets, provides feature importance.
   - Cons: Computationally intensive, slower training and prediction time.
4. **Multinomial Naive Bayes**:
   - Pros: Computationally efficient, performs well on text data, easy to implement, interpretable results.
   - Cons: Assumes independence of features, which might not always hold true.

Performance Comparison

A comparative analysis was conducted using cross-validation to evaluate the models based on accuracy, precision, recall, and F1-score. The results showed that:

- **Multinomial Naive Bayes** achieved competitive accuracy with minimal computational resources.
- **Logistic Regression** had slightly lower accuracy and required more tuning.
- **SVM** performed well but was too slow for real-time detection.
- **Random Forest** provided robust results but was computationally expensive.

Why Multinomial Naive Bayes is the Best Choice

1. **Simplicity and Efficiency**:
   - Multinomial Naive Bayes is straightforward to implement and computationally efficient, making it ideal for real-time phishing detection.
2. **Performance**:
   - Despite its simplicity, Multinomial Naive Bayes achieved competitive performance metrics, making it a reliable choice for this task.
3. **Suitability for Text Data**:
   - The model is particularly well-suited for text classification, leveraging word frequencies effectively to distinguish between phishing and legitimate emails.
4. **Scalability**:
   - The model can handle large datasets efficiently, which is crucial for processing numerous emails in a real-world scenario.

Conclusion

Given the nature of the data, the performance during preliminary testing, and the computational efficiency, Multinomial Naive Bayes was chosen as the optimal model for the Phishing Simulation and Detection System.

It provides a balanced trade-off between performance and efficiency, making it the best choice for detecting phishing emails in real-time.

Features, Advantages, Future Scope, and Limitations

Features

1. **Email Simulation**:
   - Sends simulated phishing emails to test detection capabilities.
2. **Real-time Detection**:
   - Detects phishing emails in real-time using a trained machine learning model.
3. **Text Preprocessing**:
   - Cleans and normalizes email text for better feature extraction.
4. **Feature Extraction**:
   - Uses TF-IDF to convert text data into numerical features.
5. **Model Training and Evaluation**:
   - Trains and evaluates the model, ensuring high accuracy in phishing detection.

Advantages

1. **Efficiency**:
   - The system is computationally efficient, suitable for real-time detection.
2. **High Accuracy**:
   - Achieves competitive performance in classifying phishing emails.
3. **Scalability**:
   - Can handle large datasets and numerous emails.
4. **Ease of Implementation**:
   - Utilizes well-known libraries and methods, making it easy to implement and extend.

Future Scope

1. **Advanced Machine Learning Models**:
   - Incorporate advanced models like Deep Learning for potentially higher accuracy.
2. **Phishing Attack Variants**:
   - Extend the system to detect more sophisticated phishing attacks.
3. **Integration with Email Services**:
   - Integrate with popular email services for broader applicability.
4. **User Interface**:
   - Develop a user-friendly interface for easier interaction and management.

Limitations

1. **Assumption of Feature Independence**:
   - Naive Bayes assumes feature independence, which might not always hold true.
2. **Basic Text Preprocessing**:
   - The current preprocessing might miss some sophisticated text manipulation used in phishing.
3. **Limited Dataset**:
   - Performance depends on the quality and variety of the training dataset.

Conclusion

The Phishing Simulation and Detection System is a robust and efficient solution for identifying phishing emails using machine learning and natural language processing techniques. The use of Multinomial Naive Bayes ensures a good balance between performance and computational efficiency, making it suitable for real-time applications. With future enhancements, the system can become even more powerful and versatile in combating phishing attacks.