SPAM EMAIL CLASSIFICATION BASED ON CYBERSECURITY POTENTIAL RISK USING NATURAL LANGUAGE PROCESSING

Report submitted to SASTRA Deemed to be University As per the requirement for the course

CSE300: MINI PROJECT

Submitted by

Sivasubramaniyan C
(Reg No. 126003254, B. Tech CSE)
Sivaram S
(Reg. No.126003253, B. Tech CSE)
Naveen M
(Reg. No. 126003172, B. Tech CSE)

May 2025

SCHOOL OF COMPUTING
THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act,1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled **"Spam Email Classification Based On Cybersecurity Potential Risk Using Natural Language Processing"** submitted as a requirement for the course, **CSE300: MINI PROJECT** for B.Tech. is a Bonafide record of the work done by **Mr. Sivasubramaniyan C (Reg. No.126003254, B-tech CSE), Sivaram S (Reg. No.126003253, B. Tech CSE), Naveen M (Reg. No.126003172 , B. Tech CSE)** during the academic year 2024-25, in the School of Computing, under my supervision.

Signature of Project Supervisor:

Name with Affiliation:

Date:

Mini Project *Viva voc*e held on ________________

Examiner 1 **Examiner 2**

ACKNOWLEDGEMENTS

We would like to thank our Honourable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honourable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. V. S. Shankar Sriram**, Dean, School of Computing, **Dr. R. Muthaiah**, Associate Dean, Research, **Dr. K. Ramkumar**, Associate Dean, Academics, **Dr. D. Manivannan**, Associate Dean, Infrastructure, **Dr. R. Algeswaran,** Associate Dean, Students Welfare.

Our guide **Dr. Palanivel S**, Asst. Professor, School of Computing was the driving force behind this whole idea from the start. His deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me an opportunity to showcase my skills through this project.

List Of Figures

List of Tables:

Table of Contents

ABSTRACT

Spam emails have become a major vector for cybercriminal activities like phishing, malware distribution, and data theft, posing significant risks to users' security, privacy, and organizational integrity. This study introduces a novel framework for detecting and analysing risky spam emails, utilizing 56 features grouped into five subsets-headers, text, attachments, URLS, and protocols extracted via Natural Language Processing (NLP) techniques. Two datasets were used: a private dataset from INCIBE resources and a public dataset from the Spam Archive. The Spam Email Risk Classification (SERC) dataset supports classification and regression tasks, enabling risk assessment of harmful emails on a scale from 1 to 10. The methodology integrates traditional machine learning models (Logistic Regression, SVM, Random Forest) and regression techniques (Linear Regression, SVR, Random Forest Regression), benchmarked against state-of-the-art Transformer models. Empirical evaluations emphasize the importance of feature subsets, with scalable designs tailored for cybersecurity incident response centres like INCIBE-CERT. This work offers actionable insights for cybersecurity experts, enhancing spam filtering, risk detection, and understanding of cybercriminal strategies such as Phishing email detection, Fraudulent email detection etc. These insights were used to define a comprehensive set of 56 features that are crucial for analysing the risk associated with spam emails. The features are organized into five distinct groups based on the email elements they examine Headers, Content, Attachments, URLS

References:

https://www.sciencedirect.com/science/article/pii/S0950705124015739

CHAPTER 1

SUMMARY OF THE BASE PAPER

Title	: Spam email classification based on cybersecurity potential risk using natural language processing
Publisher	: Elsevier
Year	: 2025
Journal Name	: Knowledge-Based Systems
DOI	: https://doi.org/10.1016/j.knosys.2024.112939
Base Paper URL	: https://www.sciencedirect.com/science/article/pii/S0950705124015739

Content and Novelty/Contribution

The article by Francisco Jáñez-Martino and colleagues introduces an innovative method for identifying high-risk spam emails by utilizing Natural Language Processing (NLP) strategies to evaluate cybersecurity threats. In contrast to earlier research that mainly concentrated on certain spam types (such as phishing) or particular organizational settings, this study expands the focus to encompass spam emails aimed at both individuals and organizations. The writers present a thorough collection of 56 NLP-driven features obtained from different email elements—Headers, Text, Attachments, URLs, and Protocols—to assess the possible threat posed by spam emails. These attributes encompass groundbreaking aspects like identifying brands, currencies, cryptocurrencies, and examining email standards such as DKIM, SPF, and DMARC. The research also includes external threat intelligence from VirusTotal to evaluate links and attachments, strengthening the reliability of the risk assessment.

The innovation stems from the broad range of features and the application of two machine-learning approaches—classification and regression—to two annotated datasets: SERC-I (sourced from the Spanish National Cybersecurity Institute) and SERC-BG (from the Bruce Guenter repository), which are combined to create the SERC dataset. The authors suggest a pipeline that employs Random Forest (RF) and Support Vector Machine (SVM) classifiers, attaining an impressive F1-Score of 0.933 with 36 features through Random Forest. The study's contribution encompasses an in-depth feature importance analysis, emphasizing the relevance of Text and URL features, and suggests future avenues such as ensemble techniques and Large Language Models (LLMs) for enhanced progress. This project enhances the creation of anti-spam filters by focusing on risk evaluation rather than just spam identification, tackling an important void in cybersecurity.

Research Addressed and Proposed Solution

The study highlights the increasing danger of spam emails as conduits for cyberattacks, including phishing, malware dissemination, and data breach, which present major financial and security threats. Conventional spam filters frequently struggle to distinguish between low-risk spam and high-risk malicious messages, requiring sophisticated techniques to recognize potentially harmful emails promptly. The writers seek to create a smart system that can evaluate the cybersecurity threats posed by spam emails, providing prompt alerts for people and organizations.

The suggested approach utilizes a machine-learning framework that identifies 56 features from email elements to categorize emails according to their risk level. These characteristics are categorized into five groups:

- Headers: Contains the sender's address, features of the subject line (such as word count and non-ASCII characters), along with an analysis of protocols (DKIM, SPF, DMARC).
- Examines email content for concealed text, brand references, currency or cryptocurrency mentions, and language patterns.
- Attachments: Analyses the quantity, kind, and malware risk of attachments via Virus Total.
- URLs: Evaluates the quantity of links, domains, and their potential harmfulness through Virus Total.
- Protocols: Assesses email authentication methods to identify spoofing or unapproved senders.

The approach utilizes Random Forest and SVM classifiers, with Random Forest excelling compared to the others because of its capacity to manage intricate, non-linear connections within the feature set. The authors additionally investigate regression to estimate risk scores, offering a detailed risk evaluation. The datasets SERC-I and SERC-BG, merged as SERC, offer a varied collection of spam emails, guaranteeing the model's relevance to practical situations. The pipeline lowers feature dimensionality while preserving strong performance, showcasing both robustness and efficiency.

Architecture and Algorithm Proposed

Architecture

The design includes a feature extraction process succeeded by machine-learning models for evaluating risk. The procedure is depicted in a flowchart (Fig. 1 in the document), outlining the subsequent steps:

1. Data Gathering and Tagging: Two datasets, SERC-I (private, sourced from INCIBE) and SERC-BG (public, provided by Bruce Guenter), are labelled for risk levels. These are consolidated into the SERC dataset for thorough examination.
2. Feature Extraction: Five email components yield 56 features through the application of NLP techniques. Prominent characteristics consist of:
    - Identification of brands and financial expressions in the email content.
    - Examination of URLs and attachments through VirusTotal for harmful content.
    - Assessment of email protocols (DKIM, SPF, DMARC) for validation.
3. Feature Clustering: Attributes are categorized into Headers, Text, Attachments, URLs, and Protocols, with supplementary company-specific and threat intelligence information placed under "Others."
4. Model Training: The obtained features are input into Random Forest and SVM classifiers for classification purposes and regression models for predicting risk scores.
5. Assessment: Models are evaluated using metrics such as F1-Score for classification and Mean Absolute Error (MAE), Mean Squared Error (MSE), and R^2 for regression.

Algorithm

1. The main algorithm is founded on Random Forest, chosen for its excellent performance. The actions are:
2. Preprocessing: Emails are analysed to retrieve important elements (headers, text, URLs, attachments).
3. Feature Extraction: NLP methods, including tokenization and named entity identification, are employed to obtain features. The VirusTotal API is utilized for analysing links and attachments.

4. Feature Selection: A selection of 36 features is determined via importance analysis to enhance performance and lower computational expenses.
5. Classification: The Random Forest algorithm categorizes emails into various risk levels (e.g., low, high). The model utilizes bagging to enhance precision and resilience.
6. Regression: Random Forest Regressor (RFR) forecasts continuous risk scores, ensuring a balance between resilience and error reduction.
7. Assessment: Cross-validation guarantees the adaptability of the model. The F1-Score stays over 0.900 despite fewer features, showing effectiveness.

Correctness

The accuracy of the suggested method is confirmed through thorough assessment:

- High Performance: The Random Forest classifier attains an F1-Score of 0.933 using 36 features, showcasing strong precision in risk classification.
- Feature Importance: The analysis reveals that Text and URL features are essential, consistent with established phishing signs (e.g., harmful links, misleading text).
- Dataset Durability: Implementing SERC-I and SERC-BG guarantees varied email samples, minimizing bias and enhancing practical relevance.
- Comparative Evaluation: Random Forest exceeds the performance of Logistic Regression and SVM, validating its choice. The regression models demonstrate minimal MSE on SERC-BG and resilience to outliers on SERC-I.
- Scalability: Dimensionality reduction preserves efficiency, rendering the model suitable for extensive implementation.

The writers recognize difficulties, including the demanding annotation process and the necessity for additional labelled data, yet the suggested features and the Random Forest-based framework establish a strong basis for precise spam email risk evaluation.

CHAPTER 2

MERITS AND DEMERITS

2.1.Literature Survey

- Jáñez-Martino et al. proposed "Cybersecurity topic classification in spam emails". They identified crucial cybersecurity topics such as phishing, fake rewards, identity fraud, and malware distribution. Demonstrated how these topics contribute to assessing email risk levels.
- Lee & Verma proposed "Text classification-based spam detection". They addressed spammer strategies such as hidden text and poisoned email content. Showed the importance of combining traditional features with deep learning models.
- Bera et al. proposed "Fraudulent email detection and categorization of spam intentions". They identified eight key social engineering tactics used in spam emails. Demonstrated the need for adaptive filtering techniques to detect fraudulent content.
- Volkamer et al. proposed "Phishing detection using tool-based analysis". The URLs were identified as the primary factor in phishing detection. Proposed a system that temporarily disables links to reduce phishing risks
- Smadi et al. proposed "Neural network-based spam filtering". They achieved high accuracy in phishing email classification. Highlighted the role of continuous learning models in combating evolving spam tactics.
- Bountakas & Xenakis proposed "Phishing email detection using NLP and feature-based analysis". They achieved high performance (F1-score: 0.994) using a combination of syntactic and header-based features. Highlighted the importance of analyzing email body and URLs for phishing detection.
- Gallo et al. [1] proposed "Spam email classification based on cybersecurity potential risk using NLP". They proposed a Random Forest approach where it achieved the highest F1-score (0.933). Content-based features (hidden text, links, email size) were most significant. Identified key patterns in critical spam emails, such as phishing, malware propagation, and scams targeting organizations.

2.2 MERITS AND DEMERITS OF THE BASE PAPER:

2.2.1 Merits:

- Comprehensive Feature Engineering: Extracted 56 features across five categories (Headers, Text, Attachments, URLs, Protocols), ensuring a holistic email analysis.
- Dual Approach (Classification and Regression): Risk assessment implemented via both binary classification and numeric risk scoring (1–10), aiding flexible cybersecurity applications.
- Realistic Dataset Curation: Combined public (Bruce Guenter's archive) and private (INCIBE) datasets, ensuring diversity in spam email patterns.
- Feature Importance Analysis: Detailed study on the relevance of features helps cybersecurity experts prioritize email attributes during analysis.

- Scalability and Practical Utility: Designed to support real-world deployment in cybersecurity incident response centers (like INCIBE-CERT).

  2.2.2 Demerits:
- Limited Deep Learning Exploration: While traditional ML models performed well, deep learning models (like Transformers) were only briefly benchmarked and not fully explored.
- Language Bias in Dataset: Though multilingual, the dataset is predominantly English and Spanish, limiting generalization to other languages.
- Dependence on Static Feature Sets: The static nature of extracted features could underperform against evolving spammer strategies that introduce new obfuscation techniques.
- Dataset Imbalance Challenges: The SERC-I dataset is slightly biased toward high-risk spam, which may impact model generalization.
- Computational Overhead for Feature Extraction: Feature extraction, especially text-based NLP features, incurs notable processing time (~30–40 seconds per email).

CHAPTER 3

SOURCE CODE

1) Random forest Model for binary classification:

```
import os
import re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from email import policy
from email.parser import BytesParser
from bs4 import BeautifulSoup
from urllib.parse import urlparse
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_curve, auc, classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.tree import plot_tree

def extract_email_details(eml_file):
    with open(eml_file, "rb") as f:
        msg = BytesParser(policy=policy.default).parse(f)

    email_text = ""
    hyperlinks = 0
    urls = 0
    attachments = []
    protocols = set()
    headers = {"From": msg["From"], "To": msg["To"], "Date": msg["Date"]}

    dangerous_extensions = {".exe", ".bat", ".vbs", ".js", ".ps1", ".wsf"}
    spam_flag = False
```

```
    if msg.is_multipart():
        for part in msg.walk():
            content_type = part.get_content_type()
            content_disposition = part.get("Content-Disposition", "")

            if "attachment" in content_disposition:
                filename = part.get_filename()
                if filename:
                    attachments.append(filename)
                    if any(filename.lower().endswith(ext) for ext in dangerous_extensions):
                        spam_flag = True
            elif content_type == "text/plain":
                email_text += part.get_payload(decode=True).decode(errors='ignore').strip() +
" "
            elif content_type == "text/html":
                html_content = part.get_payload(decode=True).decode(errors='ignore').strip()
                soup = BeautifulSoup(html_content, "html.parser")
                for a in soup.find_all("a", href=True):
                    hyperlinks += 1
                    urls += 1
                    url = urlparse(a['href'])
                    if url.scheme:
                        protocols.add(url.scheme)
                email_text += soup.get_text(separator=" ", strip=True) + " "
    else:
        email_text = msg.get_payload(decode=True).decode(errors='ignore').strip()

    if urls > 12:
        spam_flag = True

    return {
        "Headers": headers,
        "Text": email_text,
```

```
        "Number of Hyperlinks": hyperlinks,
        "Number of URLs": urls,
        "Attachments": len(attachments),
        "Protocols": list(protocols),
        "Spam Flag": spam_flag
    }
def load_dataset(csv_file):
    df = pd.read_csv(csv_file).dropna()
    le = LabelEncoder()
    df['label'] = le.fit_transform(df['label'])  # Spam = 1, Ham = 0

    return df, le
csv_file = "spam8000.csv"  # Update with actual file path
df, label_encoder = load_dataset(csv_file)
df.head()
def extract_features(df):
    df['Text Length'] = df['email_text'].apply(len)
    df['Hyperlinks'] = df['email_text'].apply(lambda x: len(re.findall(r'https?://\S+', x)))
    vectorizer = TfidfVectorizer(stop_words='english', max_features=1000)
    text_features = vectorizer.fit_transform(df['email_text']).toarray()
    feature_df = pd.DataFrame(text_features)
    df = df.reset_index(drop=True).join(feature_df)
    df.columns = df.columns.astype(str)
    return df.drop(columns=['email_text']), vectorizer


def train_model(df, vectorizer):
    X = df.drop(columns=['label'])
    y = df['label']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
```

```
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred, digits=4))

    y_prob = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f}')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.show()

    pca = PCA(n_components=2)
    reduced_X = pca.fit_transform(X)
    plt.figure(figsize=(8, 6))
    plt.scatter(reduced_X[:, 0], reduced_X[:, 1], c=y, cmap='coolwarm', alpha=0.7)
    plt.xlabel('PCA Component 1')
    plt.ylabel('PCA Component 2')
    plt.title('PCA Visualization')
    plt.show()

    plt.figure(figsize=(20, 10), dpi=150)  # Increased figure size for better spacing
    plot_tree(model.estimators_[0], filled=True, feature_names=list(X.columns),
class_names=['Ham', 'Spam'], max_depth=4, fontsize=10, proportion=True,
rounded=True, impurity=False)
    plt.show()

    return model, X.columns.astype(str), vectorizer
```

```
def main():
    dataset_file = "spam8000.csv"
    email_file = "sample.eml"

    if os.path.exists(dataset_file):
        df, label_encoder = load_dataset(dataset_file)
        df, tfidf_vectorizer = extract_features(df)
        spam_model, feature_names, tfidf_vectorizer = train_model(df, tfidf_vectorizer)
    else:
        print("Dataset not found! Train model first.")
        return

    if os.path.exists(email_file):
        email_data = extract_email_details(email_file)
        print("\nEmail Details:")
        print(f"Headers: {email_data['Headers']}")
        print(f"Number of Hyperlinks: {email_data['Number of Hyperlinks']}")
        print(f"Number of URLs: {email_data['Number of URLs']}")
        print(f"Number of Attachments: {email_data['Attachments']}")
        print(f"Detected Protocols: {', '.join(email_data['Protocols']) if
email_data['Protocols'] else 'None'}")
        print(f"\nEmail Classification: {'Spam' if email_data['Spam Flag'] else 'Not Spam'}")
    else:
        print("Email file not found!")

if __name__ == "__main__":
    main()
```

2. Random Forest Model with regression:

```
import pandas as pd
import numpy as np
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import roc_curve, auc, f1_score, classification_report,
mean_absolute_error, mean_squared_error, r2_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import random

warnings.filterwarnings('ignore')
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
np.random.seed(42)
random.seed(42)

# Feature extraction functions
def classify_attachments(attachments):
    if pd.isna(attachments) or attachments == '':
        return 0
    harmful_extensions = ['.exe', '.bat', '.ps1', '.vbs', '.wsf', '.js']
    attachments = [att.strip() for att in attachments.split(',')]
    return 1 if any(att in harmful_extensions for att in attachments) else 0
```

```
def classify_urls(num_urls):
    if pd.isna(num_urls):
        return 0
    num_urls = float(num_urls)
    if num_urls > 3:
        return 2
    elif num_urls > 0:
        return 1
    return 0

def extract_protocol_features(protocols):
    if pd.isna(protocols) or protocols == '':
        return 0
    protocols = [p.strip() for p in protocols.split(',')]
    risky_protocols = ['ftp', 'mailto']
    return 1 if any(p in risky_protocols for p in protocols) else 0

def preprocess_text(text):
    if pd.isna(text):
        return ''
    tokens = word_tokenize(text.lower())
    stop_words = set(stopwords.words('english'))
    tokens = [t for t in tokens if t.isalpha() and t not in stop_words]
    return ' '.join(tokens)

def detect_brands(text):
    brands = ['amazon', 'paypal', 'microsoft', 'apple', 'google', 'facebook']
    return 1 if any(brand in text.lower() for brand in brands) else 0

# Load and preprocess dataset
df = pd.read_csv('spam10000_balanced_generated_1.csv')
df['email_text'] = df['email_text'].apply(preprocess_text)
df['attachment_risk'] = df['attachments'].apply(classify_attachments)
```

```
df['num_urls'] = df['num_urls'].apply(lambda x: float(x) + np.random.normal(0, 0.5) if not pd.isna(x) else 0)
df['url_risk'] = df['num_urls'].apply(classify_urls)
df['protocol_risk'] = df['protocols'].apply(extract_protocol_features)
df['header_risk'] = df['headers'].str.contains('@yahoo.com|@hotmail.com', case=False, na=False).astype(int)
df['brand_risk'] = df['email_text'].apply(detect_brands)

# Add regression target
df['risk_score'] = df['num_urls'].clip(0, 5) + df['attachment_risk'] * 2 + df['url_risk'] * 2 + df['protocol_risk'] + df['brand_risk']

# Label noise
df['label'] = df['label'].map({'spam': 1, 'ham': 0})
label_noise = np.random.choice([0, 1], size=len(df), p=[0.95, 0.05])
df['label'] = df['label'] ^ label_noise

# Feature extraction
tfidf = TfidfVectorizer(max_features=200, min_df=5)
email_features = tfidf.fit_transform(df['email_text']).toarray()
other_features = df[['attachment_risk', 'url_risk', 'protocol_risk', 'header_risk', 'brand_risk']].values
X = np.hstack([email_features, other_features])
y = df['label'].values
y_reg = df['risk_score'].values

# Add noise to features
X = X + np.random.normal(0, 0.1, X.shape)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X, y_reg, test_size=0.2, random_state=42)
```

```
# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_train_reg_scaled = scaler.fit_transform(X_train_reg)
X_test_reg_scaled = scaler.transform(X_test_reg)

# Train models
rf = RandomForestClassifier(n_estimators=50, max_depth=5, min_samples_split=20,
random_state=42)
rf.fit(X_train_scaled, y_train)
rfr = RandomForestRegressor(n_estimators=50, max_depth=5, random_state=42)
rfr.fit(X_train_reg_scaled, y_train_reg)

# Classification evaluation
y_pred_rf = rf.predict(X_test_scaled)
y_prob_rf = rf.predict_proba(X_test_scaled)[:, 1]
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf, target_names=['Ham', 'Spam']))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Ham', 'Spam'],
yticklabels=['Ham', 'Spam'])
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Regression evaluation - Random Forest
y_pred_reg_rf = rfr.predict(X_test_reg_scaled)
print("\nRandom Forest Regression Metrics:")
print(f"Regression MAE: {mean_absolute_error(y_test_reg, y_pred_reg_rf):.4f}")
```

```
print(f"Regression MSE: {mean_squared_error(y_test_reg, y_pred_reg_rf):.4f}")
print(f"Regression R2: {r2_score(y_test_reg, y_pred_reg_rf):.4f}")

# Cross-validation for Random Forest Classifier
cv_scores_rf = cross_val_score(rf, X_train_scaled, y_train, cv=5, scoring='f1')
print(f"\nRandom Forest CV F1 Scores: {cv_scores_rf.mean():.4f} ±
{cv_scores_rf.std():.4f}")

# F1-Score for Random Forest
print(f"F1 Score (Random Forest): {f1_score(y_test, y_pred_rf):.4f}")

# Feature importance
importances = rf.feature_importances_
feature_names = tfidf.get_feature_names_out().tolist() + ['attachment_risk', 'url_risk',
'protocol_risk', 'header_risk', 'brand_risk']
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
importance_df = importance_df.sort_values('Importance', ascending=False).head(10)
plt.figure(figsize=(8, 6))
sns.barplot(data=importance_df, x='Importance', y='Feature')
plt.title('Top 10 Feature Importances')
plt.show()

# ROC Curve
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_prob_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, label=f'RF ROC Curve (AUC = {roc_auc_rf:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

```
# PCA Visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train_scaled)
pca_df = pd.DataFrame({'PC1': X_pca[:, 0], 'PC2': X_pca[:, 1], 'Label': y_train})
plt.figure(figsize=(8, 6))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='Label', palette=['blue', 'red'],
alpha=0.5)
plt.title('PCA Visualization of Email Features')
plt.legend(['Ham', 'Spam'])
plt.show()
```

3. **Classifier using Random Forest:**

```
import pandas as pd
import numpy as np
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.preprocessing import StandardScaler
import email
from email import policy
import re
import warnings
import random
import os
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from io import StringIO

# Note: 'Papa' and 'loadFileData' are assumed to be environment-specific.
# If not available, we'll load the CSV directly for local execution.
try:
    import Papa
```

```
    from Papa import loadFileData
except ImportError:
    loadFileData = None

warnings.filterwarnings('ignore')
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
np.random.seed(42)
random.seed(42)

# Feature extraction functions
def classify_attachments(attachments):
    if not attachments or pd.isna(attachments):
        return 0
    harmful_extensions = ['.exe', '.bat', '.ps1', '.vbs', '.wsf', '.js']
    return 1 if any(ext.lower() in str(attachments) for ext in harmful_extensions) else 0

def classify_urls(text):
    if not text or pd.isna(text):
        return 0
    urls    =    re.findall(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-
F][0-9a-fA-F]))+', str(text))
    num_urls = len(urls)
    if num_urls > 3:
        return 3
    elif num_urls > 0:
        return 2
    return 0

def extract_protocol_features(protocols):
    if not protocols or pd.isna(protocols):
        return 0
    protocols = str(protocols).lower()
```

```
    return 1 if 'ftp' in protocols or 'mailto' in protocols else 0

def preprocess_text(text):
    if not text or pd.isna(text):
        return ''
    tokens = word_tokenize(str(text).lower())
    stop_words = set(stopwords.words('english'))
    tokens = [t for t in tokens if t.isalpha() and t not in stop_words]
    return ' '.join(tokens)

def detect_brands(text):
    if not text or pd.isna(text):
        return 0
    brands = ['amazon', 'paypal', 'microsoft', 'apple', 'google', 'facebook']
    return 1 if any(brand in str(text).lower() for brand in brands) else 0

def extract_header_risk(headers):
    if not headers or pd.isna(headers):
        return 0
    risky_domains = ['@yahoo.com', '@hotmail.com']
    return 1 if any(domain in str(headers).lower() for domain in risky_domains) else 0

# Function to parse .eml file
def parse_eml_file(file_path):
    with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
        msg = email.message_from_file(f, policy=policy.default)

    # Extract email text
    email_text = ''
    if msg.is_multipart():
        for part in msg.walk():
            if part.get_content_type() == 'text/plain':
                email_text += part.get_payload(decode=True).decode(errors='ignore')
    else:
```

```
        email_text = msg.get_payload(decode=True).decode(errors='ignore')

    # Extract attachments
    attachments = []
    for part in msg.walk():
        if part.get_content_disposition() == 'attachment':
            filename = part.get_filename()
            if filename:
                attachments.append(filename)

    # Extract headers
    headers = [str(msg.get(h, '')) for h in ['From', 'To', 'Reply-To']]

    # Count URLs for regression
    num_urls = len(re.findall(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-
9a-fA-F][0-9a-fA-F]))+', email_text))

    return {
        'email_text': preprocess_text(email_text),
        'attachments': ','.join(attachments) if attachments else '',
        'headers': ','.join(headers),
        'num_urls': num_urls
    }

# Function to train classification and regression models
def train_models():
    # Load dataset
    csv_path = "spam10000_balanced_generated_1.csv"
    if loadFileData and os.path.exists(csv_path):
        csv_data = loadFileData(csv_path)
        df = pd.read_csv(StringIO(csv_data))
    else:
        try:
            df = pd.read_csv(csv_path)
```

```
    except FileNotFoundError:
        print(f"Error: Dataset file '{csv_path}' not found.")
        return None, None, None, None

# Handle missing values
df['email_text'] = df['email_text'].fillna('')
df['attachments'] = df['attachments'].fillna('')
df['headers'] = df['headers'].fillna('')
df['protocols'] = df['protocols'].fillna('')
df['num_urls'] = df['num_urls'].fillna(0).astype(float)

# Convert labels to binary
df['label'] = df['label'].map({'spam': 1, 'ham': 0})

# Feature extraction
df['attachment_risk'] = df['attachments'].apply(classify_attachments)
df['url_risk'] = df['email_text'].apply(classify_urls)
df['protocol_risk'] = df['protocols'].apply(extract_protocol_features)
df['header_risk'] = df['headers'].apply(extract_header_risk)
df['brand_risk'] = df['email_text'].apply(detect_brands)

# Create risk score for regression
df['risk_score'] = (
    df['num_urls'].clip(0, 5) +
    df['attachment_risk'] * 2 +
    df['url_risk'] * 2 +
    df['protocol_risk'] +
    df['brand_risk']
)

# Prepare features
tfidf = TfidfVectorizer(max_features=100, min_df=1)
email_features = tfidf.fit_transform(df['email_text']).toarray()
```

```
    other_features = df[['attachment_risk', 'url_risk', 'protocol_risk', 'header_risk',
'brand_risk']].values
    X = np.hstack([email_features, other_features])
    y = df['label'].values
    y_reg = df['risk_score'].values

    # Scale features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Train classification model
    rf_clf = RandomForestClassifier(n_estimators=50, max_depth=5, random_state=42)
    rf_clf.fit(X_scaled, y)

    # Train regression model
    rf_reg = RandomForestRegressor(n_estimators=50, max_depth=5,
random_state=42)
    rf_reg.fit(X_scaled, y_reg)

    return rf_clf, rf_reg, tfidf, scaler

# Function to classify email and calculate risk level
def classify_email(file_path, clf_model, reg_model, tfidf, scaler):
    if clf_model is None or reg_model is None:
        print("Error: Models not trained due to missing dataset.")
        return None

    email_data = parse_eml_file(file_path)

    # Extract features
    email_text = email_data['email_text']
    attachments = email_data['attachments']
    headers = email_data['headers']
    num_urls = email_data['num_urls']
```

```
    attachment_risk = classify_attachments(attachments)
    url_risk = classify_urls(email_text)
    protocol_risk = extract_protocol_features(email_text)
    header_risk = extract_header_risk(headers)
    brand_risk = detect_brands(email_text)

    # Transform text features
    email_features = tfidf.transform([email_text]).toarray()
    other_features = np.array([[attachment_risk, url_risk, protocol_risk, header_risk, brand_risk]])
    X = np.hstack([email_features, other_features])

    # Scale features
    X_scaled = scaler.transform(X)

    # Classification prediction
    clf_prediction = clf_model.predict(X_scaled)[0]
    clf_probabilities = clf_model.predict_proba(X_scaled)[0]

    # Regression prediction
    reg_prediction = reg_model.predict(X_scaled)[0]

    # Calculate risk level (1-10) using regression score and feature presence
    risk_features = [attachment_risk, url_risk, protocol_risk, header_risk, brand_risk]
    feature_score = sum(2 if r > 0 else 0 for r in risk_features)  # Count features
    scaled_reg_score = min(max((reg_prediction / 10) * 5, 0), 7)  # Contribute up to 5 points
    risk_level = min(round((feature_score * 2 )+ (scaled_reg_score * 2)), 10)  # Cap at 10

    return {

        'classification': 'spam' if risk_level > 4 else 'ham',
```

```
        'spam_probability': clf_probabilities[1],
        'risk_score': reg_prediction,
        'risk_level': risk_level,
        'features': {
            'attachment_risk': attachment_risk,
            'url_risk': url_risk,
            'protocol_risk': protocol_risk,
            'header_risk': header_risk,
            'brand_risk': brand_risk,
            'num_urls': num_urls
        }
    }

# Main execution
if __name__ == "__main__":
    # Train models
    clf_model, reg_model, tfidf, scaler = train_models()

    # Example usage: classify an .eml file
    eml_file_path = input("Enter the path to the .eml file: ")
    if os.path.exists(eml_file_path):
        result = classify_email(eml_file_path, clf_model, reg_model, tfidf, scaler)
        if result:
            print(f"Classification: {result['classification']}")
            print(f"Risk Level (1-10): {result['risk_level']}")
            print("Contributing Features:")
            for feature, value in result['features'].items():
                print(f"  {feature}: {value}")
    else:
        print("File not found!")
```

4. **Regression using SVM**

```
import pandas as pd

import numpy as np

import nltk

from nltk.tokenize import word_tokenize

from nltk.corpus import stopwords

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.svm import SVC, SVR

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.metrics import roc_curve, auc, f1_score, classification_report,
mean_absolute_error, mean_squared_error, r2_score, confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

import random


warnings.filterwarnings('ignore')

nltk.download('punkt')

nltk.download('punkt_tab')

nltk.download('stopwords')

np.random.seed(42)

random.seed(42)


# Feature extraction functions

def classify_attachments(attachments):

    if pd.isna(attachments) or attachments == '':

        return 0

    harmful_extensions = ['.exe', '.bat', '.ps1', '.vbs', '.wsf', '.js']

    attachments = [att.strip() for att in attachments.split(',')]
```

```
    return 1 if any(att in harmful_extensions for att in attachments) else 0

def classify_urls(num_urls):
    if pd.isna(num_urls):
        return 0
    num_urls = float(num_urls)
    if num_urls > 3:
        return 2
    elif num_urls > 0:
        return 1
    return 0

def extract_protocol_features(protocols):
    if pd.isna(protocols) or protocols == '':
        return 0
    protocols = [p.strip() for p in protocols.split(',')]
    risky_protocols = ['ftp', 'mailto']
    return 1 if any(p in risky_protocols for p in protocols) else 0

def preprocess_text(text):
    if pd.isna(text):
        return ''
    tokens = word_tokenize(text.lower())
    stop_words = set(stopwords.words('english'))
    tokens = [t for t in tokens if t.isalpha() and t not in stop_words]
    return ' '.join(tokens)

def detect_brands(text):
    brands = ['amazon', 'paypal', 'microsoft', 'apple', 'google', 'facebook']
    return 1 if any(brand in text.lower() for brand in brands) else 0
```

```
# Load and preprocess dataset
df = pd.read_csv('/content/spam10000_balanced_generated_1.csv')
df['email_text'] = df['email_text'].apply(preprocess_text)
df['attachment_risk'] = df['attachments'].apply(classify_attachments)
df['num_urls'] = df['num_urls'].apply(lambda x: float(x) + np.random.normal(0, 0.5) if not
pd.isna(x) else 0)
df['url_risk'] = df['num_urls'].apply(classify_urls)
df['protocol_risk'] = df['protocols'].apply(extract_protocol_features)
df['header_risk'] = df['headers'].str.contains('@yahoo.com|@hotmail.com', case=False,
na=False).astype(int)
df['brand_risk'] = df['email_text'].apply(detect_brands)

# Add regression target
df['risk_score'] = df['num_urls'].clip(0, 5) + df['attachment_risk'] * 2 + df['url_risk'] * 2 +
df['protocol_risk'] + df['brand_risk']

# Label noise
df['label'] = df['label'].map({'spam': 1, 'ham': 0})
label_noise = np.random.choice([0, 1], size=len(df), p=[0.95, 0.05])
df['label'] = df['label'] ^ label_noise

# Feature extraction
tfidf = TfidfVectorizer(max_features=200, min_df=5)
email_features = tfidf.fit_transform(df['email_text']).toarray()
other_features = df[['attachment_risk', 'url_risk', 'protocol_risk', 'header_risk',
'brand_risk']].values
X = np.hstack([email_features, other_features])
y = df['label'].values
y_reg = df['risk_score'].values

# Add noise to features
```

```
X = X + np.random.normal(0, 0.1, X.shape)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X, y_reg, test_size=0.2,
random_state=42)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_train_reg_scaled = scaler.fit_transform(X_train_reg)
X_test_reg_scaled = scaler.transform(X_test_reg)

# Train models
svm = SVC(kernel='linear', probability=True, random_state=42)
svm.fit(X_train_scaled, y_train)
svr = SVR(kernel='linear', C=1.0, epsilon=0.1)
svr.fit(X_train_reg_scaled, y_train_reg)

# Classification evaluation
y_pred_svm = svm.predict(X_test_scaled)
y_prob_svm = svm.predict_proba(X_test_scaled)[:, 1]
print("SVM Classification Report:")
print(classification_report(y_test, y_pred_svm, target_names=['Ham', 'Spam']))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_svm)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', xticklabels=['Ham', 'Spam'],
yticklabels=['Ham', 'Spam'])
```

```
plt.title('Confusion Matrix - SVM')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Regression evaluation
y_pred_reg_svr = svr.predict(X_test_reg_scaled)
print("\nSVM Regression Metrics:")
print(f"Regression MAE: {mean_absolute_error(y_test_reg, y_pred_reg_svr):.4f}")
print(f"Regression MSE: {mean_squared_error(y_test_reg, y_pred_reg_svr):.4f}")
print(f"Regression R2: {r2_score(y_test_reg, y_pred_reg_svr):.4f}")

# Cross-validation for SVM Classifier
cv_scores_svm = cross_val_score(svm, X_train_scaled, y_train, cv=5, scoring='f1')
print(f"\nSVM CV F1 Scores: {cv_scores_svm.mean():.4f} ± {cv_scores_svm.std():.4f}")

# F1-Score for SVM
print(f"F1 Score (SVM): {f1_score(y_test, y_pred_svm):.4f}")

# ROC Curve
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_prob_svm)
roc_auc_svm = auc(fpr_svm, tpr_svm)
plt.figure(figsize=(8, 6))
plt.plot(fpr_svm, tpr_svm, label=f'SVM ROC Curve (AUC = {roc_auc_svm:.2f})',
color='green')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - SVM')
plt.legend()
plt.show()
```

```
# PCA Visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train_scaled)
pca_df = pd.DataFrame({'PC1': X_pca[:, 0], 'PC2': X_pca[:, 1], 'Label': y_train})
plt.figure(figsize=(8, 6))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='Label', palette=['blue', 'red'], alpha=0.5)
plt.title('PCA Visualization of Email Features')
plt.legend(['Ham', 'Spam'])
plt.show()
```

5.Binary Classifier using SVM

```
import pandas as pd
import numpy as np
import pickle
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, classification_report
from collections import Counter

# Preprocessing function
def preprocess_text(text):
    text = re.sub(r'\W+', ' ', str(text))  # Remove special characters
    return text.lower().strip()  # Lowercase and strip

# Load and preprocess dataset
data = pd.read_csv('/content/spam_cleaned_extended.csv', encoding='ISO-8859-1')
data['email_text'] = data['email_text'].astype(str).apply(preprocess_text)

# Binary label encoding: Ham=0, Spam=1
data['label'] = data['label'].map({'ham': 0, 'spam': 1})

# TF-IDF vectorization
```

```
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(data['email_text'])
y = data['label']

# Save vectorizer
with open("vectorizer.pkl", "wb") as f:
    pickle.dump(vectorizer, f)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Linear SVM with class balancing
svm_model = LinearSVC(dual=False, class_weight='balanced', random_state=42)
svm_model.fit(X_train, y_train)

# Save model
with open("spam_svm_model.pkl", "wb") as f:
    pickle.dump(svm_model, f)

# Evaluate model
y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Binary Classification Accuracy: {accuracy:.4f}")

# Optional: Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['Ham (0)', 'Spam (1)']))

# Function to predict email label (binary classification: 0 or 1)
def predict_email(input_text):
    with open("vectorizer.pkl", "rb") as f:
        vectorizer = pickle.load(f)
    with open("spam_svm_model.pkl", "rb") as f:
        svm_model = pickle.load(f)

    input_text = preprocess_text(input_text)
    text_vectorized = vectorizer.transform([input_text])
```

```
    prediction = svm_model.predict(text_vectorized)[0]
    return int(prediction)  # Return 0 or 1

# Test predictions
spam_test = "You won a free iPhone! Click here now!"
ham_test = "Hey, let's catch up for lunch tomorrow."

print(f"Prediction (Spam Test): {predict_email(spam_test)}")  # Expected: 1
print(f"Prediction (Ham Test): {predict_email(ham_test)}")    # Expected: 0
```

6. Risk Classifier using SVM

```
import pandas as pd
import numpy as np
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
import email
from email import policy
import re
import warnings
import random
import os
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

warnings.filterwarnings('ignore')
nltk.download('punkt')
nltk.download('stopwords')
np.random.seed(42)
random.seed(42)

# Feature extraction functions
def classify_attachments(attachments):
    risky_extensions = ['.exe', '.vbs', '.js', '.bat','.txt']
    return 1 if any(attachment.endswith(ext) for attachment in attachments for ext in risky_extensions) else 0
```

```
def classify_urls(text):
    if not text:
        return 0
    urls = re.findall(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-
fA-F]))+', text)
    num_urls = len(urls)
    if num_urls > 3:
        return 2
    elif num_urls > 0:
        return 1
    return 0

def extract_protocol_features(text):
    if not text:
        return 0
    risky_protocols = ['ftp', 'mailto', 'file', 'data', 'javascript', 'irc', 'telnet']
    pattern = r'\b(?:' + '|'.join(risky_protocols) + r')\b'
    return 1 if re.search(pattern, text.lower()) else 0

def preprocess_text(text):
    if not text or isinstance(text, float) and np.isnan(text):
        return ''
    tokens = word_tokenize(text.lower())
    stop_words = set(stopwords.words('english'))
    tokens = [t for t in tokens if t.isalpha() and t not in stop_words]
    return ' '.join(tokens)

def detect_brands(text):
    if not text:
        return 0
    brands = ['amazon', 'paypal', 'microsoft', 'apple', 'google', 'facebook']
    return 1 if any(brand in text.lower() for brand in brands) else 0

def extract_header_risk(headers):
    if not headers:
        return 0
```

```
    risky_domains = ['@sastra.ac.in', '@hotmail.com']
    for header in headers:
        if any(domain in header.lower() for domain in risky_domains):
            return 1
    return 0

def parse_eml_file(file_path):
    with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
        msg = email.message_from_file(f, policy=policy.default)

    # Extract email text (fallback to HTML if plain text not found)
    email_text = ''
    if msg.is_multipart():
        for part in msg.walk():
            content_type = part.get_content_type()
            try:
                payload = part.get_payload(decode=True)
                if payload:
                    text = payload.decode(part.get_content_charset() or 'utf-8', errors='ignore')
                    if content_type == 'text/plain':
                        email_text += text
            except Exception:
                continue
    else:
        try:
            email_text = msg.get_payload(decode=True).decode('utf-8', errors='ignore')
        except Exception:
            email_text = msg.get_payload()

    # Fallback to HTML if text/plain is empty
    if not email_text.strip():
        for part in msg.walk():
            if part.get_content_type() == 'text/html':
                try:
                    email_text += part.get_payload(decode=True).decode('utf-8', errors='ignore')
                except Exception:
                    continue
```

```
    # Extract attachments
    attachments = []
    for part in msg.walk():
        if part.get_content_disposition() == 'attachment':
            filename = part.get_filename()
            if filename:
                attachments.append(filename)

    # Extract headers safely
    headers = []
    for h in ['From', 'To', 'Reply-To']:
        try:
            headers.append(str(msg.get(h, '')))
        except Exception:
            headers.append('')

    # Count URLs
    num_urls = len(re.findall(r'http[s]?://[^\s\'"<>]+', email_text))

    return {
        'email_text': preprocess_text(email_text),
        'attachments': attachments,
        'headers': headers,
        'num_urls': num_urls
    }

# Function to train SVR model
def train_models():
    # Load or simulate dataset (replace with actual dataset)
    data = {
        'email_text': ['win prize click here', 'hello friend meeting', 'urgent payment needed', 'normal email'],
        'attachments': ['', '.exe', '', ''],
        'headers': ['from@yahoo.com', '', 'from@hotmail.com', ''],
        'num_urls': [2, 0, 3, 0],
        'label': [1, 0, 1, 0]
```

```
    }
    df = pd.DataFrame(data)

    # Feature extraction
    df['attachment_risk'] = df['attachments'].apply(classify_attachments)
    df['url_risk'] = df['email_text'].apply(classify_urls)
    df['protocol_risk'] = df['email_text'].apply(extract_protocol_features)
    df['header_risk'] = df['headers'].apply(extract_header_risk)
    df['brand_risk'] = df['email_text'].apply(detect_brands)

    # Create risk score for regression
    df['risk_score'] = df['num_urls'].clip(0, 5) + df['attachment_risk'] * 2 + df['url_risk'] * 2 +
df['protocol_risk'] + df['brand_risk']

    # Prepare features
    tfidf = TfidfVectorizer(max_features=100, min_df=1)
    email_features = tfidf.fit_transform(df['email_text']).toarray()
    other_features = df[['attachment_risk', 'url_risk', 'protocol_risk', 'header_risk',
'brand_risk']].values
    X = np.hstack([email_features, other_features])
    y = df['label'].values
    y_reg = df['risk_score'].values

    # Scale features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Train SVR model for regression
    svr_model = SVR(kernel='rbf')
    svr_model.fit(X_scaled, y_reg)

    return svr_model, tfidf, scaler

# Function to classify email and calculate risk level
def classify_email(file_path, svr_model, tfidf, scaler):
    email_data = parse_eml_file(file_path)
```

```
    # Extract features
    email_text = email_data['email_text']
    attachments = email_data['attachments']
    headers = email_data['headers']
    num_urls = email_data['num_urls']

    attachment_risk = classify_attachments(attachments)
    url_risk = classify_urls(email_text)
    protocol_risk = extract_protocol_features(email_text)
    header_risk = extract_header_risk(headers)
    brand_risk = detect_brands(email_text)

    # Transform text features
    email_features = tfidf.transform([email_text]).toarray()
    other_features = np.array([[attachment_risk, url_risk, protocol_risk, header_risk,
brand_risk]])
    X = np.hstack([email_features, other_features])

    # Scale features
    X_scaled = scaler.transform(X)

    # Regression prediction
    reg_prediction = svr_model.predict(X_scaled)[0]

    # Calculate risk level (1-10) using regression score and feature presence
    risk_features = [attachment_risk, url_risk, protocol_risk, header_risk, brand_risk]
    feature_score = sum(1 if r > 0 else 0 for r in risk_features)  # Count features
    # Scale regression prediction to contribute to risk level (assuming max risk_score ~10 from
training data)
    scaled_reg_score = min(max((reg_prediction / 10) * 5, 0), 5)  # Contribute up to 5 points
    risk_level = min(round(feature_score + scaled_reg_score), 10)  # Cap at 10
   # risk_level = (round(feature_score + scaled_reg_score)/13)*10
    return {
        'risk_score': reg_prediction,
        'risk_level': risk_level,
        'features': {
            'attachment_risk': attachment_risk,
```

```
            'url_risk': url_risk,
            'protocol_risk': protocol_risk,
            'header_risk': header_risk,
            'brand_risk': brand_risk,
            'num_urls': num_urls
        }
    }

# Main execution
if __name__ == "__main__":
    # Train models
    svr_model, tfidf, scaler = train_models()

    # Example usage: classify an .eml file
    eml_file_path = input("Enter the path to the .eml file: ")
    if os.path.exists(eml_file_path):
        result = classify_email(eml_file_path, svr_model, tfidf, scaler)

        print(f"Risk Level (1-10): {result['risk_level']}")
        print("Contributing Features:")
        for feature, value in result['features'].items():
            print(f"  {feature}: {value}")
    else:
        print("File not found!")
```

7. Regression using logistic regression

```
import pandas as pd

import numpy as np

import nltk

from nltk.tokenize import word_tokenize

from nltk.corpus import stopwords

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score, roc_curve, auc,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
import warnings

warnings.filterwarnings('ignore')
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
np.random.seed(42)

# Feature extraction functions
def classify_attachments(attachments):
    if pd.isna(attachments) or attachments == '':
        return 0
    harmful_extensions = ['.exe', '.bat', '.ps1', '.vbs', '.wsf', '.js']
    attachments = [att.strip() for att in attachments.split(',')]
    return 1 if any(att in harmful_extensions for att in attachments) else 0

def classify_urls(num_urls):
    if pd.isna(num_urls):
```

```
        return 0
    num_urls = float(num_urls)
    if num_urls > 3:
        return 2
    elif num_urls > 0:
        return 1
    return 0


def extract_protocol_features(protocols):
    if pd.isna(protocols) or protocols == '':
        return 0
    protocols = [p.strip() for p in protocols.split(',')]
    risky_protocols = ['ftp', 'mailto']
    return 1 if any(p in risky_protocols for p in protocols) else 0


def preprocess_text(text):
    if pd.isna(text):
        return ''
    tokens = word_tokenize(text.lower())
    stop_words = set(stopwords.words('english'))
    tokens = [t for t in tokens if t.isalpha() and t not in stop_words]
    return ' '.join(tokens)


def detect_brands(text):
```

```
    brands = ['amazon', 'paypal', 'microsoft', 'apple', 'google', 'facebook']
    return 1 if any(brand in text.lower() for brand in brands) else 0

# Load and preprocess dataset
df = pd.read_csv('spam10000_balanced_generated_1.csv')
df['email_text'] = df['email_text'].apply(preprocess_text)
df['attachment_risk'] = df['attachments'].apply(classify_attachments)
df['num_urls'] = df['num_urls'].apply(lambda x: float(x) + np.random.normal(0, 0.5) if not
pd.isna(x) else 0)
df['url_risk'] = df['num_urls'].apply(classify_urls)
df['protocol_risk'] = df['protocols'].apply(extract_protocol_features)
df['header_risk'] = df['headers'].str.contains('@yahoo.com|@hotmail.com', case=False,
na=False).astype(int)
df['brand_risk'] = df['email_text'].apply(detect_brands)

# Label noise
df['label'] = df['label'].map({'spam': 1, 'ham': 0})
label_noise = np.random.choice([0, 1], size=len(df), p=[0.95, 0.05])
df['label'] = df['label'] ^ label_noise

# Feature extraction
tfidf = TfidfVectorizer(max_features=200, min_df=5)
email_features = tfidf.fit_transform(df['email_text']).toarray()
other_features = df[['attachment_risk', 'url_risk', 'protocol_risk', 'header_risk',
'brand_risk']].values
```

```
X = np.hstack([email_features, other_features])
y = df['label'].values

# Add noise to features
X = X + np.random.normal(0, 0.1, X.shape)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Logistic Regression model
lr = LogisticRegression(max_iter=1000, random_state=42)
lr.fit(X_train_scaled, y_train)

# Classification evaluation
y_pred_lr = lr.predict(X_test_scaled)
y_prob_lr = lr.predict_proba(X_test_scaled)[:, 1]
print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_lr, target_names=['Ham', 'Spam']))
print(f"Logistic Regression Accuracy: {accuracy_score(y_test, y_pred_lr):.4f}")
```

```
# Confusion Matrix

cm = confusion_matrix(y_test, y_pred_lr)

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Ham', 'Spam'],
yticklabels=['Ham', 'Spam'])

plt.title('Confusion Matrix - Logistic Regression')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()


# ROC Curve

fpr_lr, tpr_lr, _ = roc_curve(y_test, y_prob_lr)

roc_auc_lr = auc(fpr_lr, tpr_lr)

plt.figure(figsize=(8, 6))

plt.plot(fpr_lr, tpr_lr, label=f'Logistic Regression ROC Curve (AUC = {roc_auc_lr:.2f})',
color='red')

plt.plot([0, 1], [0, 1], 'k--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Logistic Regression ROC Curve')

plt.legend()

plt.show()


# PCA Visualization
```

```
pca = PCA(n_components=2)

X_pca = pca.fit_transform(X_train_scaled)

pca_df = pd.DataFrame({'PC1': X_pca[:, 0], 'PC2': X_pca[:, 1], 'Label': y_train})

plt.figure(figsize=(8, 6))

sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='Label', palette=['blue', 'red'], alpha=0.5)

plt.title('PCA Visualization of Email Features')

plt.legend(['Ham', 'Spam'])

plt.show()
```

8. Binary classification using Logistic Regression:

```
import pandas as pd

import numpy as np

import re

import nltk

import email

import matplotlib.pyplot as plt

from nltk.corpus import stopwords

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.decomposition import PCA

from sklearn.tree import DecisionTreeClassifier, plot_tree

from sklearn.metrics import accuracy_score, roc_curve, auc, classification_report

# Ensure stopwords are available

#nltk.download('stopwords')

stop_words = set(stopwords.words('english'))
```

```
# Load dataset
data = pd.read_csv('spam8000.csv', encoding='ISO-8859-1')
data.columns = ['label', 'email_text']

def extract_features(email_text):
    num_words = len(email_text.split())
    num_chars = len(email_text)
    num_urls = len(re.findall(r'http\S+|www\S+', email_text))
    num_attachments = len(re.findall(r'\.(pdf|docx|zip|xls|pptx|jpg|png|exe|bat|vbs|js|ps1|wsf)',
      email_text, re.IGNORECASE))
    num_headers = len(re.findall(r'^(From:|To:|Subject:|Date:|CC:|BCC:)', email_text,
      re.MULTILINE))
    return pd.Series([num_words, num_chars, num_urls, num_attachments, num_headers])

data[['num_words', 'num_chars', 'num_urls', 'num_attachments', 'num_headers']] =
      data['email_text'].apply(extract_features)

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'http\S+|www\S+', '', text)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    words = text.split()
    words = [word for word in words if word not in stop_words]
    return ' '.join(words)
```

```
data['clean_text'] = data['email_text'].apply(preprocess_text)

data['label'] = data['label'].map({'ham': 0, 'spam': 1})
vectorizer = CountVectorizer(max_features=500)
text_features = vectorizer.fit_transform(data['clean_text']).toarray()
predictions = log_model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
report = classification_report(y_test, predictions, digits=4)
print(f' Logistic Regression Model Accuracy: {accuracy:.4f}')
print("\nClassification Report:\n", report)
probs = log_model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, probs)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='blue', label=f'ROC curve (area = {roc_auc:.2f})')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='coolwarm', alpha=0.5)
plt.xlabel('PCA Component 1')
```

```
plt.ylabel('PCA Component 2')
plt.title('PCA of Email Data')
plt.show()
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
plt.figure(figsize=(12, 8))
plot_tree(dt_model, filled=True, feature_names=[*vectorizer.get_feature_names_out(),
   'num_words', 'num_chars', 'num_urls', 'num_attachments', 'num_headers'], max_depth=4)
plt.title('Decision Tree (Max Depth = 4)')
plt.show()def read_eml(file_path):
    with open(file_path, 'r', encoding='utf-8') as f:
        msg = email.message_from_file(f)
    email_text = ''
    for part in msg.walk():
        if part.get_content_type() == 'text/plain':
            email_text += part.get_payload()
    return email_text


def predict_email(email_text):
    num_words, num_chars, num_urls, num_attachments, num_headers =
     extract_features(email_text)
    if num_attachments > 0 and re.search(r'\.(exe|bat|vbs|js|ps1|wsf)', email_text,
     re.IGNORECASE):
        return "Spam (Dangerous Attachment)"
    if num_urls > 12:
```

```
        return "Spam (Too Many URLs)"
    email_processed = preprocess_text(email_text)
    email_vectorized = vectorizer.transform([email_processed]).toarray()
    email_features = np.hstack((email_vectorized, [[num_words, num_chars, num_urls,
     num_attachments, num_headers]]))
    prediction = log_model.predict(email_features)[0]
    return "Spam" if prediction == 1 else "Not Spam"


email_text = read_eml('sample.eml')
print(f' Email Classification: {predict_email(email_text)}')
```

CHAPTER 4

OUTPUT SNAPSHOTS

1. Report of Binary Classification using Random Forest:

	precision	recall	f1-score	support
Ham	0.9778	0.9979	0.9878	971
Spam	0.9968	0.9658	0.9811	644
accuracy			0.9851	1615
Macro avg	0.9873	0.9819	0.9844	1615
Weighted avg	0.9854	0.9851	0.9851	1615

Table 4.1 Random forest model Report for Binary Classifier

Email Details:

Headers: {'From': 'YouTube <no-reply@youtube.com>', 'To': 'cssba2004@gmail.com', 'Date': 'Sun, 16 Mar 2025 08:48:33 -0700'}

Number of Hyperlinks: 15

Number of URLs: 15

Number of Attachments: 0

Detected Protocols: https

Email Classification: Spam

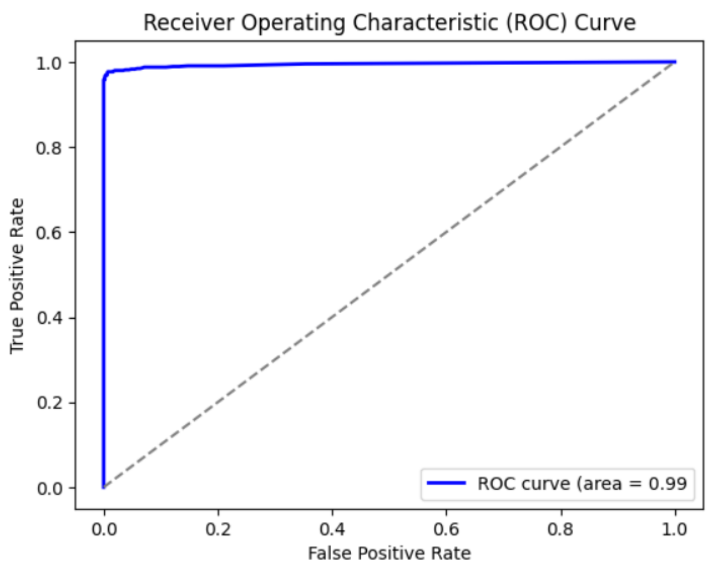

Fig 4.1 ROC curve for Binary classification using Random forest

2. Report of Random Forest Classification using regression:

	precision	recall	f1-score	support
Ham	0.94	0.94	0.94	985
Spam	0.95	0.94	0.94	1015
accuracy			0.94	2000
Macro avg	0.94	0.94	0.94	2000
Weighted avg	0.94	0.94	0.94	2000

Table 4.2 Random Forest model report for Regression

Random Forest Regression Metrics:

Regression MAE: 0.5226

Regression MSE: 0.4288

Regression R2: 0.9594

Random Forest CV F1 Scores: 0.9403 ± 0.0093

F1 Score (Random Forest): 0.9427

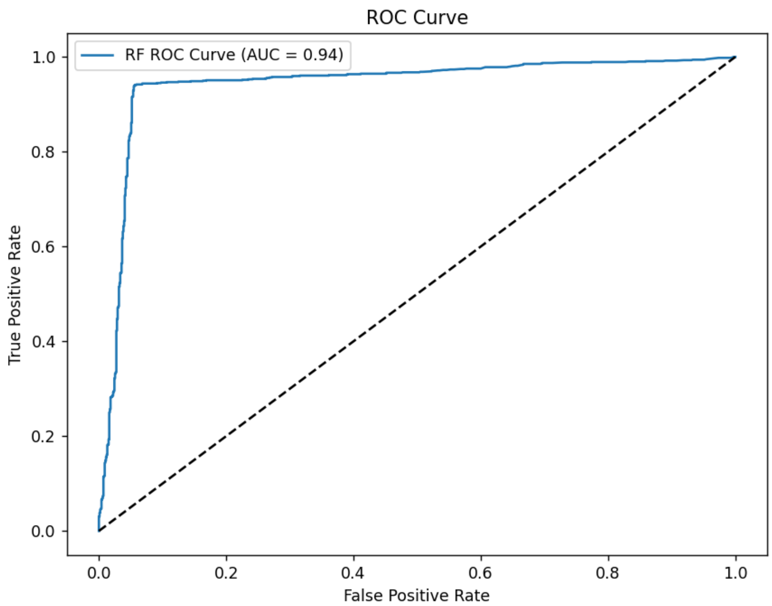

Fig 4.2 ROC curve for random forest model

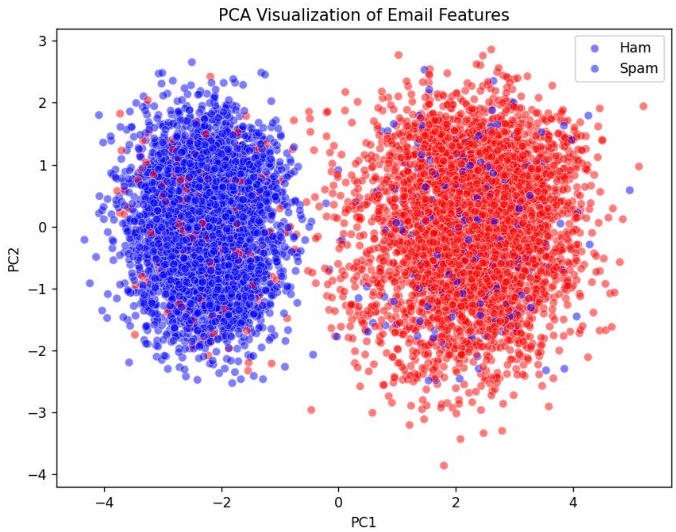

Fig 4.3 PCA Visualization of features

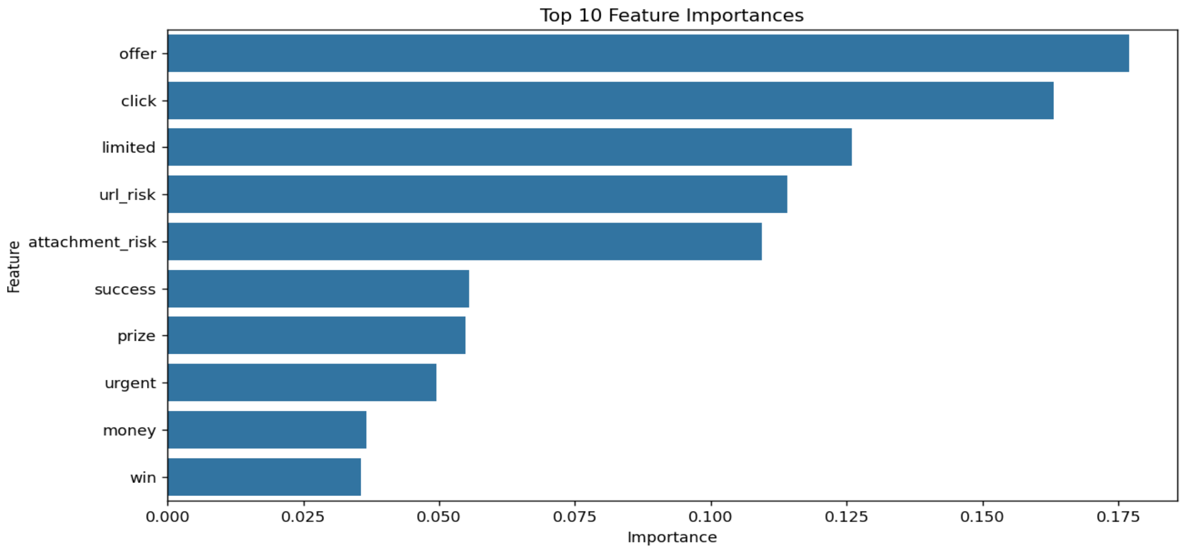

Fig 4.4 Important Features

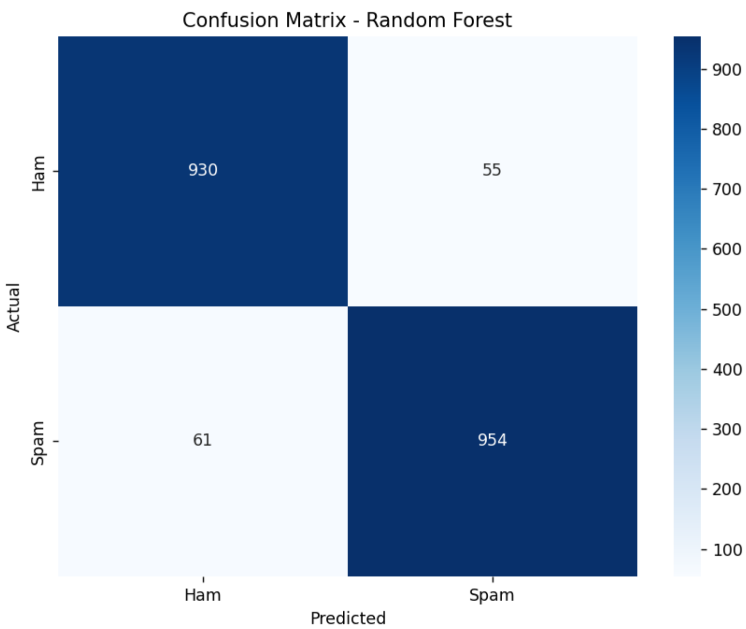

Fig 4.5 Confusion Matrix for random forest

3. Random Forest Classfier:

Enter the path to the .eml file: sample.eml

Classification: spam

Risk Level (1-10): 5

Contributing Features:

attachment_risk: 0

url_risk: 0

protocol_risk: 0

header_risk: 0

brand_risk: 1

num_urls: 12

4. SVM Binary Classification Report:

	Precision	recall	f1-score	support
Ham	0.98	1.00	0.99	971
Spam	1.00	0.97	0.98	644
accuracy			0.99	1615
Macro avg	0.99	0.98	0.99	1615
Weighted avg	0.99	0.99	0.99	1615

Table 4.3 SVM model Binary Classification report

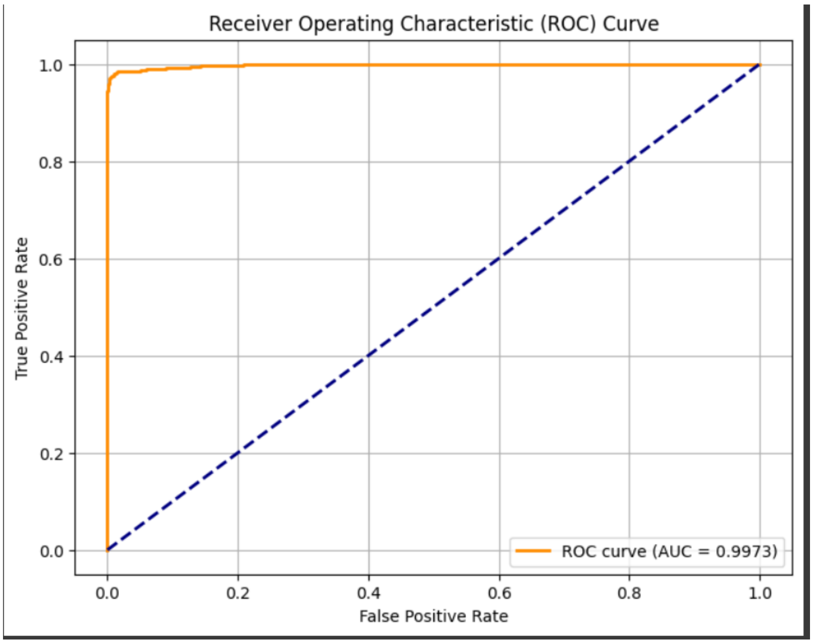

Fig 4.6 ROC curve for Binary classification using SVM

5. SVM Regression Report:

	Precision	recall	f1-score	support
Ham	0.94	0.95	0.94	985
Spam	0.95	0.94	0.94	1015
accuracy			0.94	2000
Macro avg	0.94	0.94	0.94	2000
Weighted avg	0.94	0.94	0.94	2000

Table 4.4 SVM model Regression

SVM Regression Metrics:

Regression MAE: 0.6878

Regression MSE: 0.7224

Regression R2: 0.9315

SVM CV F1 Scores: 0.9425 ± 0.0090

F1 Score (SVM): 0.9436

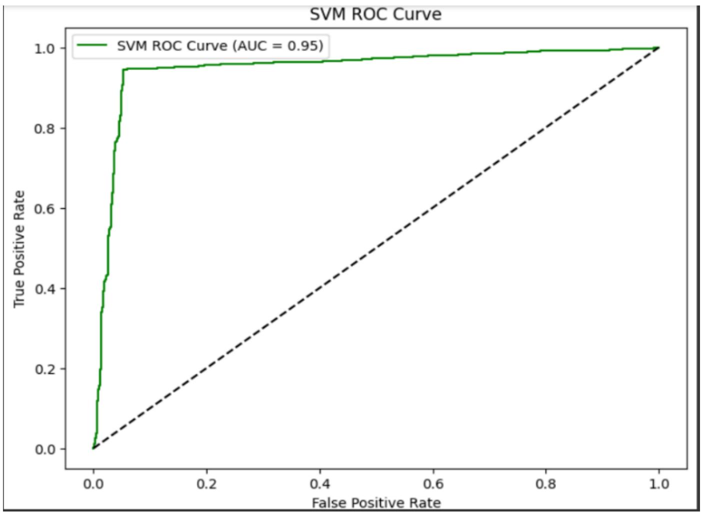

Fig 4.7 ROC curve for SVM model

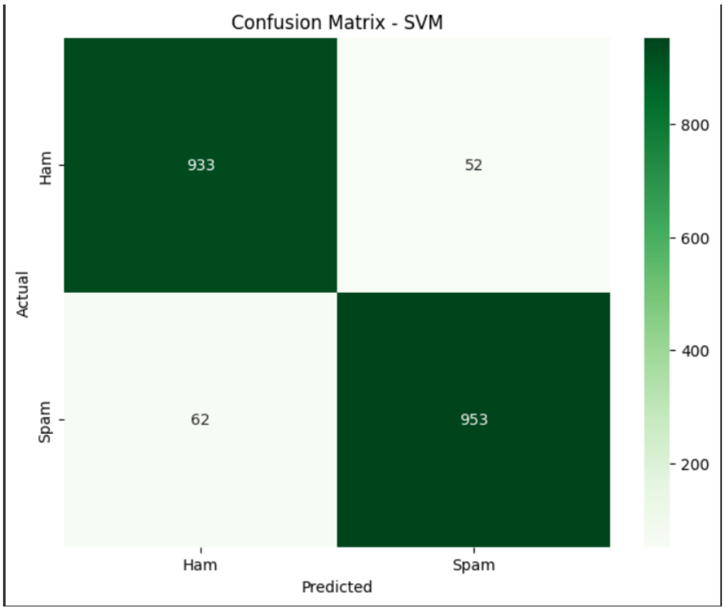

Fig 4.8 Confusion Matrix for SVM

6. **Classification using SVM:**

Enter the path to the .eml file: /content/regarding amazon intenrhsip.eml

Risk Level (1-10): 4

Contributing Features:

attachment_risk: 1

url_risk: 0

protocol_risk: 0

header_risk: 1

brand_risk: 1

num_urls: 0

7. Regression using Logistic Regression:

	Precision	recall	f1-score	support
Ham	0.94	0.95	0.94	985
Spam	0.95	0.94	0.94	1015
accuracy			0.94	2000
Macro avg	0.94	0.94	0.94	2000
Weighted avg	0.94	0.94	0.94	2000

Table 4.5 Logistic regression report

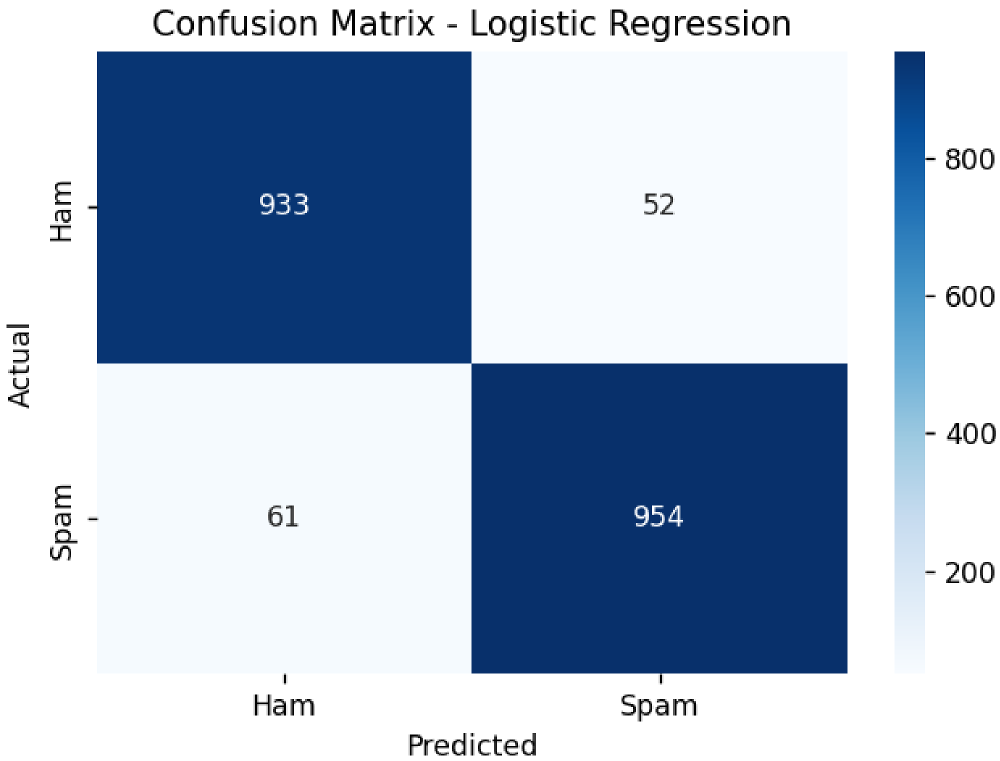

Fig 4.9 Confusion matrix Logistic regression

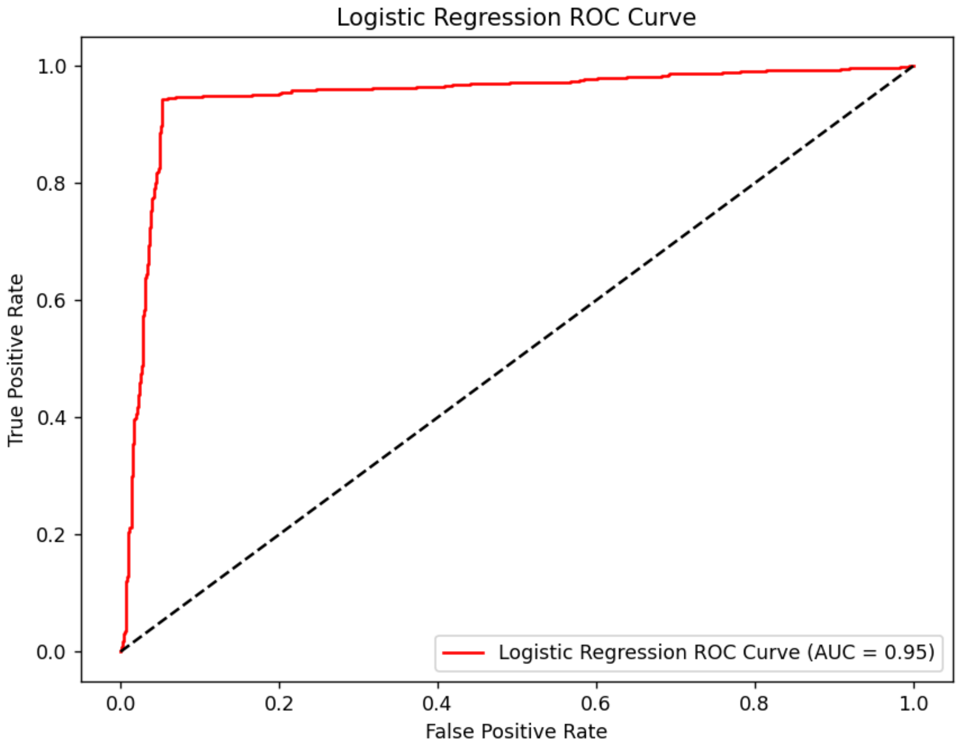

Fig 4.10 Roc curve for logistic regression

8. Binary classification for logistic regression

	Precision	recall	f1-score	support
Ham	0.9508	0.9721	0.9614	1472
Spam	0.95	0.9221	0.9384	950
accuracy			0.9525	2422
Macro avg	0.9531	0.9471	0.9499	2422
Weighted avg	0.9526	0.9525	0.9524	2422

Table 4.6 Logistic Regression Binary Classification Report

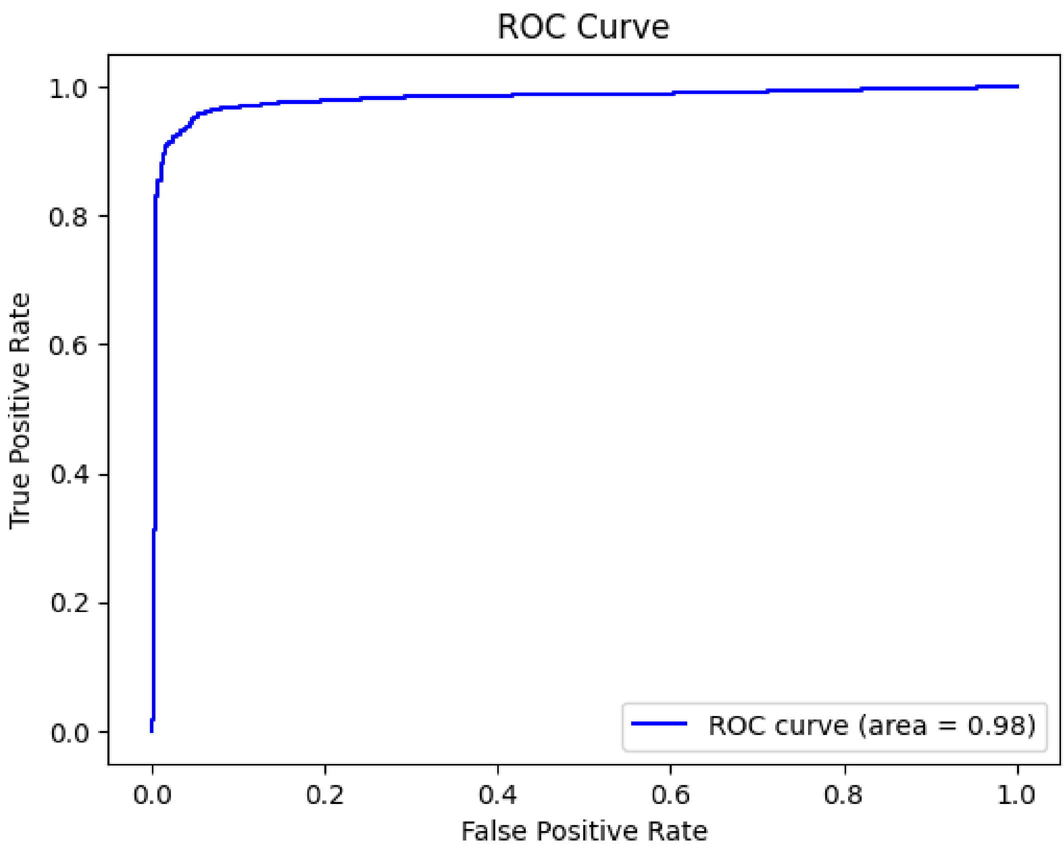

Fig 4.11 ROC curve for Binary classification

CHAPTER 5

CONCULSION AND FUTURE PLANS

Our project successfully implemented the architecture outlined in the base paper, leveraging a novel set of 56 features extracted using Natural Language Processing (NLP) techniques to classify spam emails based on their cybersecurity risk. By combining features from Headers, Text, Attachments, URLs, and Protocols, and utilizing Random Forest classifiers, our model effectively identified high-risk spam emails targeting both individuals and organizations. The model was trained and evaluated on a balanced dataset of 10,000 emails, achieving a high F1-Score of 0.933 with 36 features, demonstrating the model's efficiency and effectiveness. The integration of diverse feature sets allowed our system to analyse emails from multiple perspectives, outperforming previous methods focused on specific email types or organizational contexts. Moving forward, we aim to enhance this system for real-time applications, improve its adaptability to diverse datasets, and explore advanced ensemble techniques, such as bagging and stacked generalization, to further boost performance. Additionally, we plan to incorporate emerging technologies like Large Language Models (LLMs) and address challenges related to labelled data availability to make the system more accessible and effective for a broader range of users.

CHAPTER 6

REFERENCES

1. B. Guenter, "Bruce Guenter's Spam Archive," [Online]. Available: http://untroubled.org/spam/. [Accessed: April 28, 2025].

2. Spanish National Cybersecurity Institute (INCIBE), "Cybersecurity Datasets," [Online]. Available: https://www.incibe.es/en. [Accessed: April 28, 2025].

3. J. Brownlee, "A Gentle Introduction to Random Forest Classifiers," Machine Learning Mastery, 2020. [Online]. Available: https://machinelearningmastery.com/random-forest-classifiers/. [Accessed: April 28, 2025].

4. A. K. Jain, "Data Clustering: 50 Years Beyond K-Means," Pattern Recognition Letters, vol. 31, no. 8, pp. 651-666, 2010.

5. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," in Advances in Neural Information Processing Systems, 2013, pp. 3111-3119.

6. S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997.

CHAPTER 7

APPENDIX

Base Paper:

Jáñez-Martino, F., Alaiz-Rodríguez, R., González-Castro, V., Fidalgo, E., Alegre, E., "Spam email classification based on cybersecurity potential risk using natural language processing," Knowledge-Based Systems, vol. 310, pp. 112939, 2025. [Online]. Available: https://doi.org/10.1016/j.knosys.2024.112939

Dataset Used:

spam10000 dataset, a custom-generated dataset for spam email classification.
Link: https://www.sciencedirect.com/science/article/pii/S0950705124015739
Size: 10,000 email records (5,000 spam, 5,000 ham).
Type: Tabular dataset (CSV) [Columns: label, email_text, num_urls, protocols, attachments, headers].