3rd Sem Mini Project Report on

Email Classification Using Random Forest Algorithm

Submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE & ENGINEERING

Submitted by:

Student Name: Sameer Singh Bhandari **University Roll No:** 2319495

Department of Computer Science and Engineering

Graphic Era Hill University

Dehradun, Uttarakhand

2024-25

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the project report entitled **"Email Classification Using Random Forest Algorithm"** in partial fulfillment of the requirements for the award of the Degree of *Bachelor of Technology in Computer Science and Engineering* in the Department of Computer Science and Engineering of the Graphic Era Hill University, Dehradun shall be carried out by the undersigned under the supervision of **Mr. Nishant Bhandari**, Department of Computer Science and Engineering, Graphic Era Hill University, Dehradun.

Student Name:	**University Roll No:**
Sameer Singh Bhandari	2319495

The above-mentioned student shall be working under the supervision of the undersigned on the

"Email Classification Using Random Forest Algorithm"

Supervisor	**Head of the Department**
Mr. Nishant Bhandari	Prof. (Dr.) Dibyahash Bordoloi

Examination

Name of the Examiners	**Signature with Date**
1. Mr. Nishant Bhandari	

Table of Contents

Introduction and Problem Statement

Introduction

Overview of Spam Email Detection:

Spam email detection has become crucial in combating the growing menace of unwanted and potentially harmful emails. Spam emails can lead to phishing, financial fraud, and wasted resources.

Need for Automated Spam Detection Systems:

Manual spam detection is impractical due to the high volume of emails. Automated systems powered by machine learning algorithms offer scalable and accurate solutions. These systems adapt to evolving spam patterns, ensuring better protection for users.

Role of Machine Learning and Random Forest:

Machine learning enables systems to learn from data and make predictions without explicit programming. The Random Forest algorithm, known for its high accuracy and robustness, is well-suited for email classification tasks due to its ability to handle noisy and imbalanced datasets effectively.

Problem Statement

Challenges in Detecting Spam Emails

Spam detection faces challenges such as:

- High email volumes.
- Evolving and diverse spam patterns.
- Differentiating between spam and legitimate (ham) emails.

Limitations of Traditional Filters

Rule-based and keyword-based spam filters often fail to adapt to new spam techniques and lack scalability.

Need for an Accurate Detection System

There is a pressing need for a scalable and automated system that accurately classifies emails while handling noise and variability in the dataset.

Literature Survey

Traditional Methods

- **Rule-based Filters:** Fixed rules to identify spam but lack adaptability.
- **Keyword-based Filters:** Detect specific keywords but prone to false positives.

Modern Approaches

- **Naive Bayes:** Probabilistic model but struggles with high-dimensional data.

- **Support Vector Machines (SVM):** Effective for binary classification but computationally intensive.
- **Random Forest:** Outperforms other methods with high accuracy and robustness.

Justification for Using Random Forest

- High accuracy due to ensemble learning.
- Handles noisy and imbalanced datasets effectively.
- Provides feature importance insights for improved decision-making.

Objectives

1. Develop a robust email classification system with high accuracy.
2. Preprocess email data for efficient feature extraction.
3. Train and evaluate a Random Forest-based classification model.
4. Deploy the model using Flask and React for real-time prediction.
5. Provide spam/ham classification with confidence percentages.

Methodology

Data Description

***Dataset*:** Spam detection dataset (e.g., spam.csv).

***Features*:**

- **Message:** Text content of the email.
- **Label:** Spam or ham classification.

Data Preprocessing

***Cleaning*:** Removing punctuation, numbers, stopwords and converting to the small characters.

***Tokenization*:** Splitting text into words.

***Vectorization*:** Transforming text to numerical data using CountVectorizer.

***Example of Text Transformation*:**

Original Text	Cleaned Text	Vectorized Format
"Buy now! Limited ..."	"buy limited …"	[1, 0, ...]

Model Selection and Training

***Algorithm*:** Random Forest.

- **Parameters:** Number of estimators, depth, and splits.

***Training*:**

- Train-test split (80/20) with stratification.
- Metrics: Accuracy, precision, recall, F1-score.

System Workflow

***Architecture*:**

- User input ➔ Preprocessing ➔ Vectorization ➔ Prediction ➔ Output (Spam/ Ham).

Implementation

Tools And Technologies

Python Libraries:

- **NLTK:** Text Preprocessing.
- **Scikit-learn:** Model Training.
- **Joblib:** Model Serialization.

Frameworks:

- **Flask:** Backend Development.
- **React:** Frontend Interface.

Code Snippets:

- **Load Data:**

```
df = pd.read_csv("./dataset/spam.csv")
```

- **Data Preprocessing:**

```
def clean_text(text):
    text = re.sub(r'[^\w\s]', ' ', text)
    text = re.sub(r'\n', ' ', text)
    text = re.sub(r'\d+', '', text)
    return text.lower()

df['Message'] = df['Message'].apply(clean_text)

nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

def remove_stop_words(text):
    filtered_words = [word for word in text.split() if word.lower() not in stop_words]
    return ' '.join(filtered_words)

df['Message'] = df['Message'].apply(remove_stop_words)
```

- **Feature Extraction**

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(data['Message'])
```

- **Encoding labels:**

```
label_encoder = LabelEncoder()
df['Label'] = label_encoder.fit_transform(df['Label'])
```

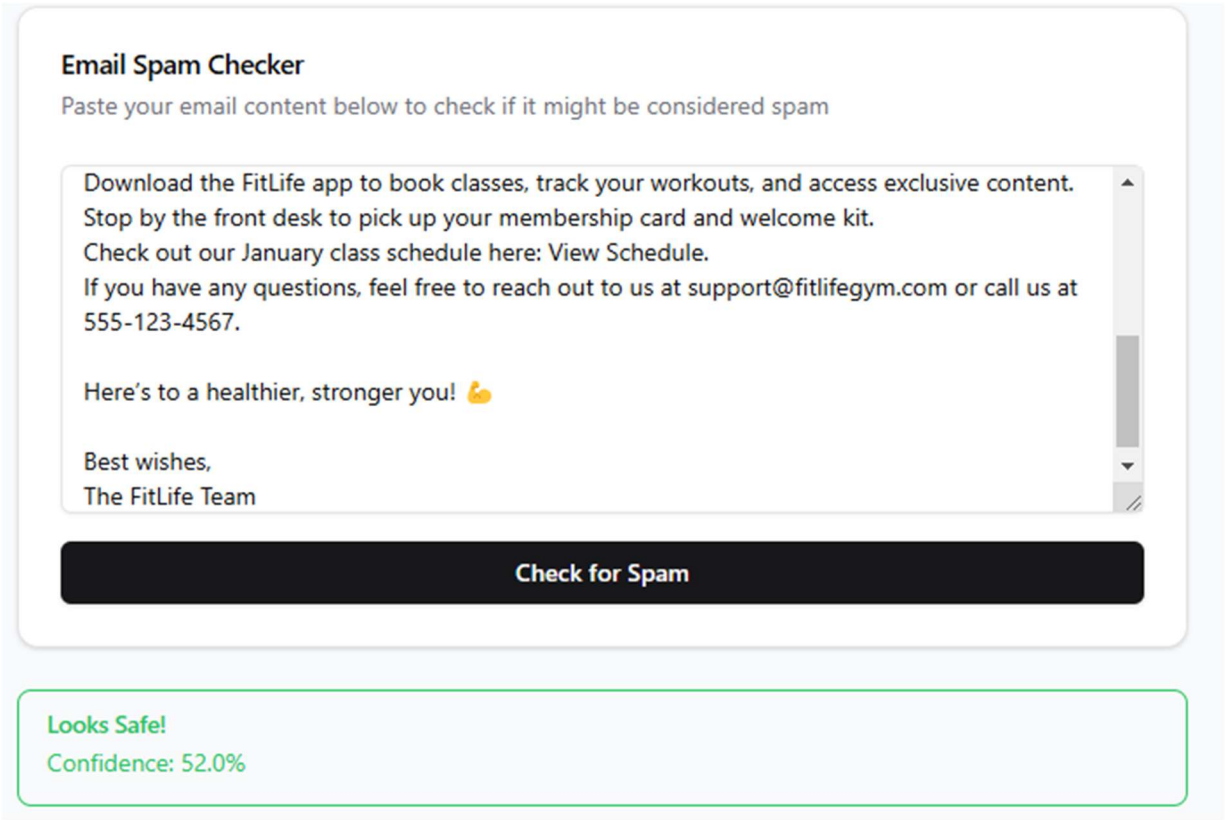
- **Model Training:**

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)
```

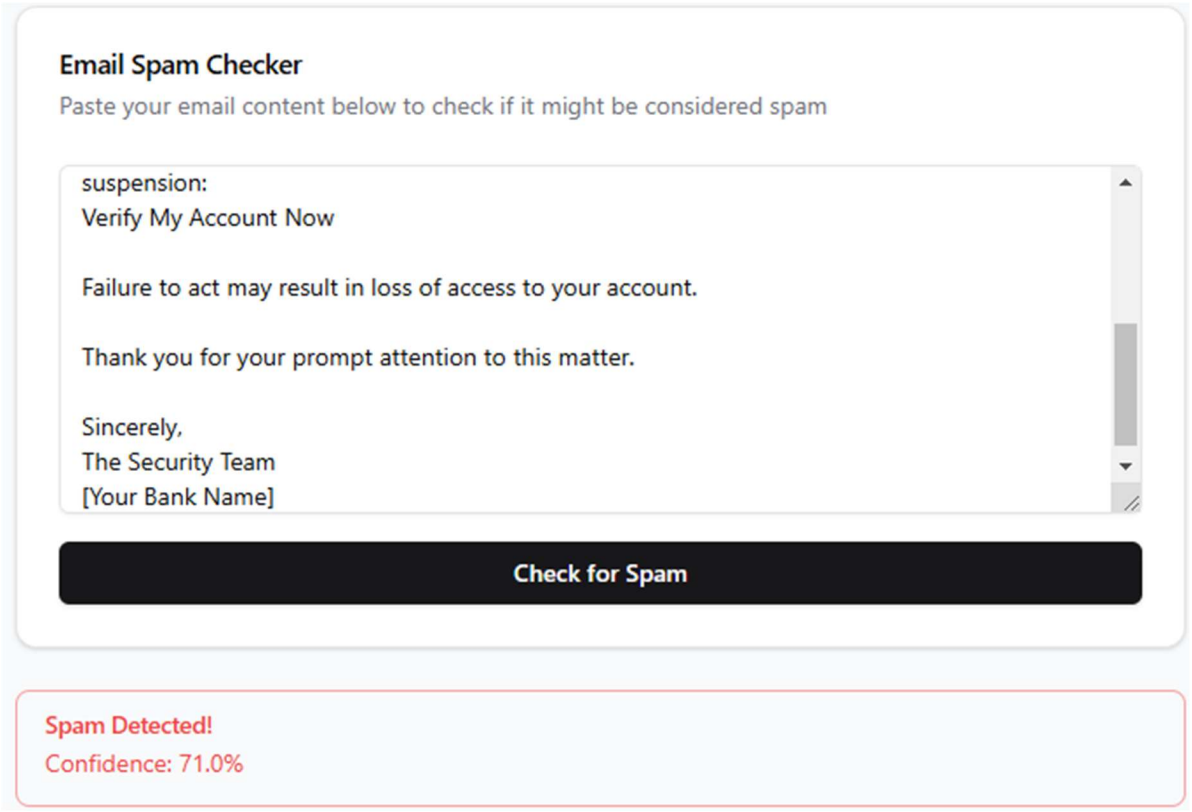
- **Flask Endpoint:**

```
@app.route("/predict", methods=['POST'])
def predict():
    input_text = request.json.get('message')

    cleaned_text = clean_text(input_text)
    processed_text = remove_stop_words(cleaned_text)

    vectorized_text = vectorizer.transform([processed_text])

    probas = model.predict_proba(vectorized_text)

    predicted_class = model.predict(vectorized_text)[0]
    predicted_probability = probas[0][predicted_class]
    confidence_percentage = predicted_probability * 100

    return jsonify({
        "prediction": int(predicted_class),
        "confidence": round(confidence_percentage, 2)
    })
```

Ham Example:



Spam Example:



Results and Discussion

Model Performance

Metrics:

- Accuracy: 96%
- Precision: 98%
- Recall: 89%
- F1-Score: 93%

Visualization:

- **Confusion Matrix:** Displays True positives, false positives, etc.
- **Accuracy Graph:** Trends over training epochs.

Strengths

- High accuracy and robust to noise.
- Effective for real-time predictions.

Limitations

- Handling highly imbalanced data remains challenging.
- Limited multilingual support.

Learnings

1. Gained technical knowledge in text preprocessing and machine learning.
2. Improved skills in backend/frontend integration.
3. Enhanced teamwork and project management capabilities.

Conclusion and References

Conclusion

This project successfully developed an email classification system with 96% accuracy, leveraging the Random Forest algorithm. The real-time prediction tool was deployed using Flask and React. Future improvements include deep learning models and multilingual spam detection capabilities.

References

1. Scikit-learn documentation: https://scikit-learn.org
2. Flask documentation: https://flask.palletsprojects.com
3. React documentation: https://reactjs.org
4. Dataset source: https://huggingface.co/datasets/thehamkercat/telegram-spam-ham/tree/main