A

Capstone Project Report

On

SpamChecker

Submitted during IV semester in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology

in

Computer Engineering

by

Mahesh (23001050015)

Under supervision of

Dr. Rewa Sharma

Department of Computer Engineering
FACULTY OF INFORMATICS & COMPUTING

J.C. Bose University of Science & Technology, YMCA
Faridabad – 121006

May 2025

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in this Project titled "SpamCheker (Email Spam Classifier)" in fulfilment of the requirements for the degree of Bachelor of Technology in Computer engineering and submitted to "J.C. Bose University of Science and Technology, YMCA, Faridabad", is an authentic record of my own work carried out under the supervision of Dr. Rewa Sharma

The work contained in this project has not been submitted to any other University or Institute for the award of any degree or diploma by me.

Mahesh

CERTIFICATE

It is hereby certified that the project titled "SpamCheker (Email Spam Classifier)" submitted by **Mahesh (23001050015**) of 2_{nd} Year (IV Semester) to "**J. C. Bose University of Science and Technology, YMCA, Faridabad**" for the award of the degree of Bachelor of Technology in Computer Engineering is a record of a bonafide work carried out by him/her in the Project Lab – J.C. Bose University of Science and Technology, YMCA, Faridabad under my supervision as mentor for May 2025 examination.

In my opinion, the project has reached the standards of fulfilling the requirements of the regulations to the degree.

Mentor Name (DR. Rewa Sharma)
Assistant Professor
Department of Computer Engg.

ACKNOWLEDGEMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals.

I would like to extend my sincere thanks to all them. I am highly indebted to my mentor Dr. Rewa Sharma for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my parents & members of J.C. Bose University of Science and Technology for their kind cooperation and encouragement which helped in the completion of this project.

Mahesh

TABLE OF CONTENTS

Chapter 1: Introduction

1.1 Brief of the Project

With the exponential increase in digital communication, spam emails and messages have become a major concern for users and service providers alike. These spam messages not only clutter inboxes but often contain phishing links or harmful content intended to deceive users.

To combat this issue, the **Spam Email Classifier** project aims to build a machine learning-based web application capable of identifying and classifying incoming text messages or emails as either *spam* or *ham* (not spam).

This project leverages **Natural Language Processing (NLP)** techniques for text preprocessing and a trained **Naïve Bayes classifier** to detect spam based on patterns and keywords.

The classifier learns from historical labelled data and can then make predictions on unseen messages. A simple and intuitive web interface built using **Flask** allows users to input a message or email and receive an instant prediction.

The goal of the project is to provide an accurate, fast, and deployable solution that can help users and developers integrate spam detection into their applications.

1.2 Technologies Used

Technology / Tool	Purpose
• Python	• Main programming language for development
• Flask	• Web framework for building the web application
• Pandas	• Data manipulation and handling CSV datasets
• NLTK	• Natural Language Toolkit for stopwords, etc.
• Scikit-learn	• ML model building, CountVectorizer, evaluation
• Joblib	• Saving and loading trained models efficiently
• HTML/CSS (Jinja2)	• Front-end templates rendered via Flask
• Gunicorn	• Production-ready server used in deployment

1.3Document Conventions

There are no used conventions till now.

1.4 Intended Audience and Reading Suggestions

We intend this document for anyone (such as developers, project managers, programmers, users, testers) who is interested in knowing about an E-Mail Spam Classifier.
The SRS contain answers for the following questions:

- WHY an E-mail Spam Classifier is made?
- HOW an E-mail Spam Classifier is made?
- WHAT are the requirements for an E-mail Spam Classifier?

1.5 Product Scope

Spamming is a big problem! Anyone who is connected to internet faces this problem now and then, but with Corporates, it becomes a serious problem. Phishing attacks, greedy traps and many create a havoc if not handled properly.
Our product **E-Mail Spam Classifier** is a solution for these Spams. Through our Intelligent Web- Based Product, you can find Spam E-Mails and can label them as Spam for future reference.

1.6 References

1.6.1 Websites

- www.kaggle.com
- www.sci-hub.tw
- www.ieeexplore.ieee.org
- www.medium.com

1.6.2 Dataset

https://www.kaggle.com/venky73/spam-mails-dataset

Chapter 2: Overall Description

2.1 Problem Formulation

Spam messages—whether emails or SMS—pose a major risk in terms of privacy, security, and productivity. Manually identifying and deleting them is inefficient. The goal of this project is to develop a system that automatically classifies incoming text as Spam or Not Spam (Ham) using natural language processing and machine learning. The system should be fast, lightweight, and able to generalize across both email and SMS formats.

Key Problems Identified:

- Presence of misleading or unwanted promotional/spam messages.
- Difficulty in manual filtering for high-volume users.
- Need for an automated solution that classifies unstructured text effectively

2.2 Objectives

- To design and implement a Spam Detection System using NLP and Machine Learning.
- To preprocess input text using standard NLP techniques like stop word removal and punctuation filtering.
- To train and use a classification model (e.g., Naïve Bayes) that can label messages as spam or ham.
- To deploy the trained model into a simple web application using Flask.
- To create a system capable of working for both emails and SMS text data.
- To evaluate the model on real-world data for accuracy and reliability.

2.3 User Classes and Characteristics

Our user class is vastly distributed across all kinds of Computer-related Industries. Some of them are mentioned below:

- General Public:
  Our Sole important target is common people who use E-Mail daily for their works and general internet communications. These are the one who gets the most E-Mail spams, and according to surveys, the public is the most favorite targets of these Spammers.
- Corporate Companies:
  Next big user class for this Product will be Corporate Companies which, because of employee's negligence and inadequate knowledge about Spam E-mail and trap their traps, makes companies' data vulnerable and pose great economic losses.

2.4 User Interface

The User Interface of our Project is simple, user-friendly, and self-explanatory. As per our current product Idea, we have a basic website which is being hosted on the internet globally which has certain key options to check Spam EMails.

2.5 Hardware Interfaces

The Hardware required to run this Product is very minimal and is mentioned below:

- Operating System – Win/Mac/Linux (any distribution)
- Web Browser–Chrome/Mozilla/Opera/Tor (Updated with latest FTP/HTTP support)
- Proper Internet Connection (above 1 Mbps)

2.6 Software Interfaces

There are tons of software which are available online which can be used to make a Spam Classifier. But we have mentioned some softwares which we have used and will use in our Development of this Product, which are mentioned below:

- Jupyter Notebook
- PyCharm & Web Browser (Chrome)

Chapter 3: Implementation and Result Analysis

3.1 System Architecture

The system is structured into three key layers:

- User Interface (Frontend):
  A simple web interface created using Flask's render template() method and HTML pages (index.html, results.html). Users can input text (email or SMS) and receive classification results directly in the browser.

- Processing Layer (Backend Logic):
  Handles the core functionalities including:
  - Text preprocessing using NLTK.
  - Vectorization using the pre-trained CountVectorizer.
  - Prediction using a serialized Naïve Bayes model.
  - Result rendering back to the frontend.

- Model Layer (ML Pipeline):
  A trained machine learning model is responsible for making spam/ham predictions. The model and vectorizer were saved using joblib and reused during inference to avoid retraining and maintain efficiency.

3.2 Data Preprocessing

Preprocessing is a critical step handled via a custom process text() function in app.py, which includes:

- Punctuation Removal: Strips symbols using Python's string. Punctuation.
- Tokenization and Stopword Removal: Tokenizes text into words and filters out common stopwords using NLTK's English corpus.
- Case Normalization: Converts text to lowercase for consistency.

These preprocessing steps ensure that the input data is clean and suitable for vectorization.

3.3 Model Development and Training

- Algorithm Used:
  Multinomial Naïve Bayes (ideal for text classification).
- Features Extracted:
  Bag of Words (BoW) using CountVectorizer to convert textual data into numerical features.
- Workflow:
  1. Preprocessed dataset using the process text() function.
  2. Split dataset into training and test sets (e.g., 80-20 ratio).
  3. Trained the Naïve Bayes model.
  4. Evaluated model accuracy, precision, and recall.
  5. Serialized model and vectorizer using joblib for deployment.

3.4 Web Application Implementation (Flask)

- The Flask framework was used to build the backend.
- Main routes include:
  - / → Displays homepage (index.html).
  - /predict → Accepts POST requests, processes input, and returns prediction via results.html
- The vectorizer and model are loaded once at the start:

```
python                                                    Copy    Edit

classifier = joblib.load('finalized_model.sav')
vectorizer = joblib.load('vectorizer.pkl')
```

- Text from the form is vectorized and passed to the classifier:

```python
vect = vectorizer.transform([comment]).toarray()
my_prediction = classifier.predict(vect)
```

3.5 Deployment Process

- The application was tested locally using:
- python app.py
- Required packages and libraries were recorded in a requirements.txt file.
- The app is structured to be deployable on platforms like Heroku or Render.
- Static and templates folders handle HTML views and CSS styling, maintaining clear project structure.

3.6 Result Analysis

```python
#Evaluate the model on the test data set
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score
pred = classifier.predict(X_test)
print(classification_report(y_test ,pred ))
print('Confusion Matrix: \n', confusion_matrix(y_test,pred))
print()
print('Accuracy: ', accuracy_score(y_test,pred))
```

```
              precision    recall  f1-score   support

         ham       0.98      0.98      0.98       732
        spam       0.95      0.96      0.96       303

    accuracy                           0.97      1035
   macro avg       0.97      0.97      0.97      1035
weighted avg       0.97      0.97      0.97      1035
```

- **Test Cases:** The system was tested against real-world SMS and email samples.
- **Edge Case Handling:** Checks for empty inputs are implemented to ensure robust user interaction.
- **Overall Performance:** The application delivers quick and reliable spam detection with an intuitive user interface.

Chapter 4: Making Project Functional

4.1 Linking Machine Learning Model with Frontend

This section discusses in detail, about how we made our Project Functional with Graphical User Interface:

Front-end:

The project's GUI has been made through web pages written in HTML and design elegantly with CSS and JavaScript for it to be attractive.

Back-end Framework:

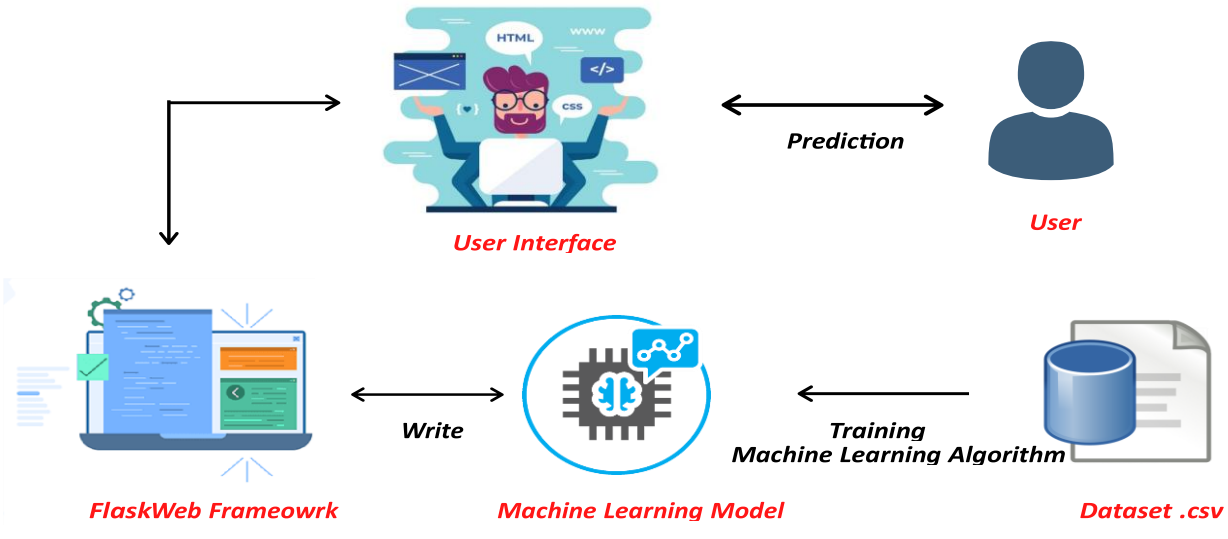
Flask helps in implementing a machine learning application in Python that can be easily plugged, extended and deployed as a web application.

Machine Learning Model:

The model detects, if a text message/mail is spam(1) or ham(0) using techniques like tokenization, multinomial naive bayes classifier, etc.

Web Server*:*

All these functional units are connected by establishing a server on the web by hosting it on Render, which helps us in running the project successfully..



4.2 Hosting Model on web through Flask Framework

Flask acts as a collection of software packages which can help us easily create a web application. It helps in implementing a machine learning application in Python that can be easily plugged, extended and deployed as a web application. Flask is based on two key components: WSGI toolkit and Jinja2 template engine. WSGI is a specification for web applications and Jinja2 renders web pages.

Step 1: Building a Machine Learning Model

As we have already implemented a machine learning model in the previous section that takes in an input and outputs a variable. For a quick recap, **our machine learning model takes in an input i.e. any mail received and gives us the predicted output as "Spam" or "Not Spam".**

Hence our machine learning model looks like:

```
classifier = MultinomialNB()
classifier.fit(X_train, y_train)
output = classifier.predict(X_train)  #output
```

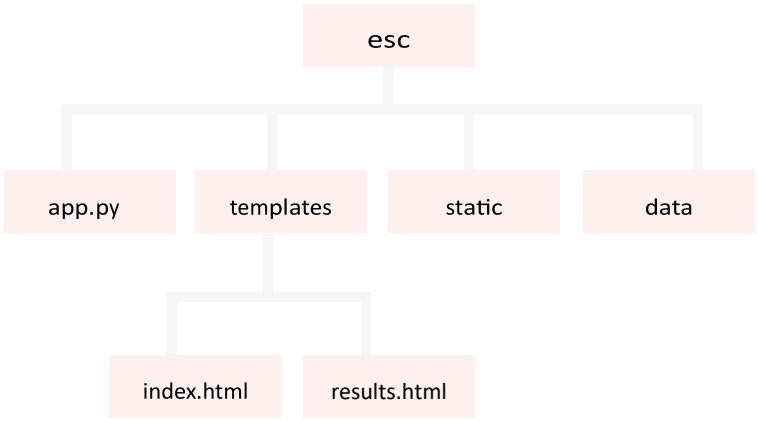
Step 2: Installing Flask

Installing Flask is easy and can be done by just opening the command prompt and running the following command:

```
pip install flask
```

Step 3: Creating a Folder structure

Creating a folder structure is important. We will have to create 4 sub sections as shown below:



app.py:

This Python file will contain the Flask specific code. templates:

This folder will contain the HTML files. These HTML files will be rendered on the web browser.

static:

This folder will contain the CSS, JS files. These files will also be rendered on the web browser. data:

This folder will contain the saved Machine Learning model and dataset.

Embedding Flask into app.py

Creating a file app.py and perform the following steps:

Step 1: Importing libraries

```
from flask import Flask, render_template, url_for, request
import pandas as pd
from nltk.corpus import stopwords
import string
import joblibfrom flask import Flask, render_template, url_for, request
import pandas as pd
from nltk.corpus import stopwords
import string
import joblib
import nltk
nltk.download('stopwords')  # Download once at the start
import nltk
nltk.download('stopwords')  # Download once at the start
```

Step 2: Instantiate Flask and Tokenization Function

```
app = Flask(__name__)

# Tokenization function
```

```
def process_text(text):
    # Remove punctuation
    nopunc = [char for char in text if char not in string.punctuation]
    nopunc = ''.join(nopunc)

    # Remove stop words
    clean_words = [word for word in nopunc.split() if word.lower() not in
stopwords.words('english')]

    # Return clean words
    return clean_words
```

Step 3: Load The Trained model and Vectorizer once at the start of the app

```
classifier = joblib.load('finalized_model.sav')  # Load the saved Naive Bayes model
vectorizer = joblib.load('vectorizer.pkl')  # Load the saved CountVectorizer
```

Step 4: Setting up Routes

```
@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    # When the user submits a comment, it will be transformed using the vectorizer
and predicted using the classifier
    if request.method == 'POST':
        comment = request.form['comment']  # Get the comment from the form

        # Check if the comment is empty
        if not comment.strip():  # If the comment is empty or only contains spaces
            return render_template('results.html', prediction="empty")  # Return empty
message

        data = [comment]  # Convert to a list (the vectorizer expects a list)

        # Transform the input using the loaded vectorizer
```

```
    vect = vectorizer.transform(data).toarray()  # Use the pre-fitted vectorizer to transform the data

    # Predict the class (spam/ham) using the loaded classifier
    my_prediction = classifier.predict(vect)

    prediction_result = my_prediction[0]  # No conversion to int

    # Render the results in the results.html template
    return render_template('results.html', prediction=prediction_result)
```

into app.py

Explanation:

In the above Flask code, "/" URL is now mapped to the render_template function. Therefore if a user visits "/", then Flask will render index.html on the web browser of the user. Flask will look for the html file in the templates folder.

Flask is based on Jinja2 template engine. The render_template() function renders the template and expects it to be stored in the Templates folder on the same level as the app.py file.

Next the function is now mapped to predict() function which will load the pre-trained model. Subsequently, it will then get the entered mail by the user and will display the predicted value as "SPAM" or "NOT SPAM" on the web page by rendering the results.html file.

step 5: Run the Module

```
if __name__ ==
'__main__':
app.run(debug=True)
```

What happens when app,run() is executed?

1.When the app.run() is executed then the application will start running. It will run on the host “localhost” which is your local machine on the port number 5000.

2.The debug flag is used during development so that the Flask server can reload the code without restarting the application.

3.It also outputs useful debugging information. Therefore, when the user will visit http://localhost:5000/, Flask will render the index.html page.

implementing .html pages

Steps:

Step 6: Implement index.html

```
            <form action="{{ url_for('predict') }}" method="POST" class="form-
inline" id="comment-form" onsubmit="return validateForm()">
                <textarea name="comment" class="form-control" id="comment"
cols="50" rows="4" placeholder="Enter Text Message*" required></textarea>
                <input type="submit" class="submit" value="Check Spam or Not!">
            </form>
            <!-- Form End -->
          </div>
        </div>
      </div>
    </div>

    <!-- Welcome thumb (Image section) -->
    <div class="welcome-thumb wow fadeInDown" data-wow-delay="0.5s"
style="text-align: center; margin-top: 20px;">
      <!-- Image path updated to use Flask's url_for -->
      <img src="{{ url_for('static', filename='img/bg-img/1.1.png') }}" alt="Spam
Checker" style="max-width: 100%; height: auto;">
    </div>
```

The key to note is the action attribute. It is set to **"{{ url_for('predict')}}".** So, when a user enters an E-mail and submits it. The POST method stores it in the **name** attribute named "comment" in the above html code & then passes it to the render_template() function.

Step 7: Implement results.html

```
<div class="wellcome-heading">
                <!-- Check the prediction value and display the result accordingly -->
                {% if prediction == 'spam' %}
                    <h2 style="color: white;">Spam</h2>
                {% elif prediction == 'ham' %}
                    <h2 style="color: white;">Not a Spam!</h2>
                {% endif %}
            </div>

            <!-- Redirect message -->
            <p id="redirect-message" style="font-size: 1.2em; color: white;">You will be redirected back in a few seconds...</p>

            <script>
                // Set a timer to go back after 5 seconds
                setTimeout(function() {
                    window.history.back(); // Go back to the previous page
                }, 3000);  // Delay in milliseconds (3000ms = 3 seconds)
            </script>
        </div>
    </div>
</div>
```

The **prediction** containers will display the predicted spam or not spam result. These values will be passed in via the render_template() function in the app.py file

Step 8: run the application to localhost

Now, go to the project directory in the CMD and type:

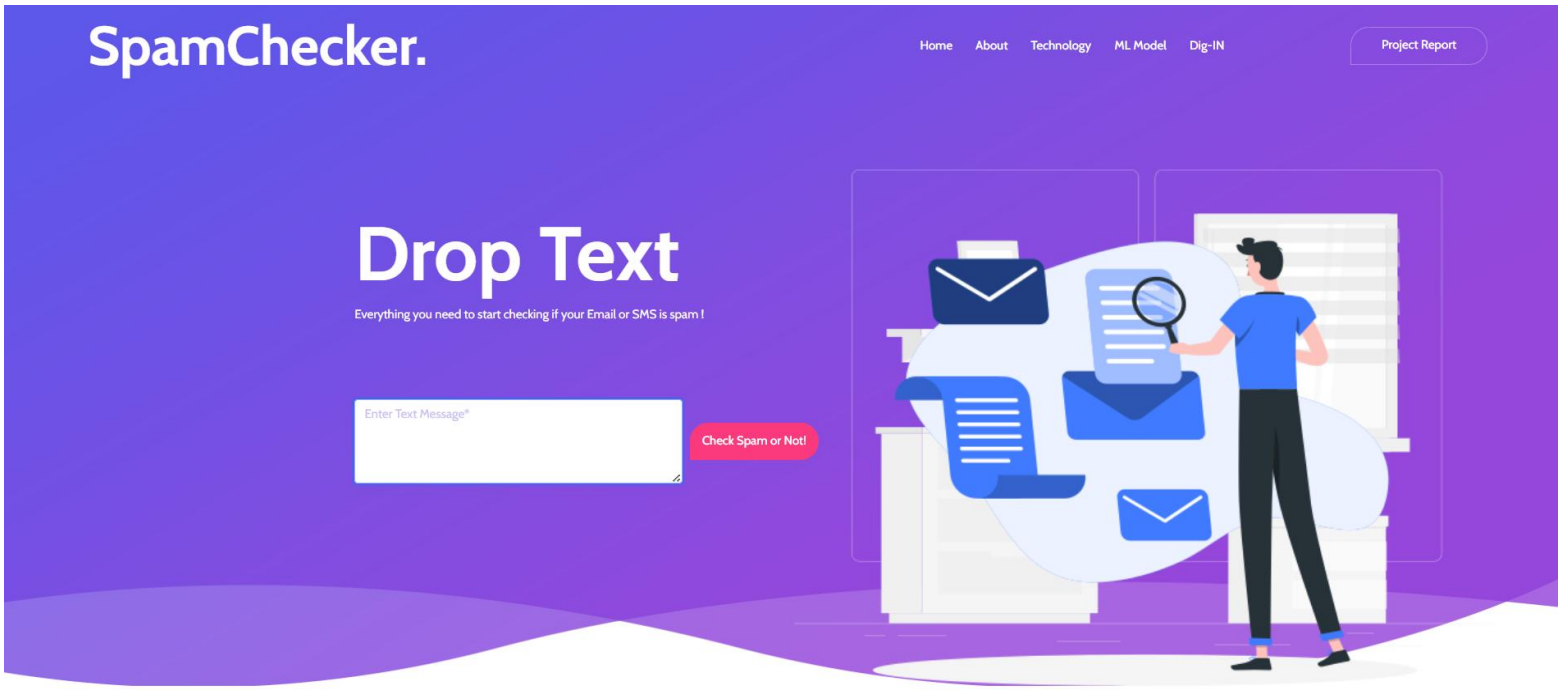
```
python app.py
```

This will now run your python application using Flask micro framework on your local machine. Flask will run the application on port 5000 therefore navigate to:

```
http://127.0.0.1:5000
```

Our model is now running on a web browser on our localhost. The last task is to deploy it on an external server so that the public can access it. For this Project we have, hosted our project on **Render ,** by uploading the project folders on the web console in the structure mentioned and by installing all the libraries on the bash console. This project can now be accessed by visiting the following url:

User Interface(index.html)



User Interface(result.html)