DETECTING SPAM E-MAIL WITH ML OPTIMIZED WITH METAHEURISTIC ALGORITHMS

CONTENTS

TOPIC	PAGE NO
ABSTRACT	**I**
LIST OF FIGURES	**II**
LIST OF TABLES	**III**
1. INTRODUCTION	**4-9**
1.1. Introduction & Objectives	4
1.2. Project Objectives	5
1.3. Purpose of the project	5
1.4. Existing System with Disadvantages	5
1.5. Proposed System With features	6-7
1.6. Input and Output Design	8-9
2. LITERATURE SURVEY	**10-11**
3. SOFTWARE REQUIREMENT ANALYSIS	**12-14**
3.1. Problem Specification	12
3.2. Modules and their Functionalities	12
3.3. Functional Requirements	12
3.4. Non-Functional Requirements	13
3.5. Feasibility Study	13-14
4. SOFTWARE & HARDWARE REQUIREMENTS	**15**
4.1. Software requirements	15
4.2. Hardware requirements	15
5. SOFTWARE DESIGN	**16-22**

5.1. System Architecture	16
5.2. Dataflow Diagrams	17
5.3. UML Diagrams	18-22
6. CODING AND IMPLEMENTATION	**23-31**
6.1. Source code	23-28
6.2. Implementation	29
6.2.1. Python	29
6.2.2. Modules used in project	29-31
7. SYSTEM TESTING	**32-39**
7.1. Types of System Testing	32-35
7.2. Test Cases	36-39
8. OUTPUT SCREENS	**40**
9. CONCLUSION	**41**
10. FUTURE ENHANCEMENTS	**42**
11. REFERENCES	**43-44**

ABSTRACT

Spam email detection is a critical task in the domain of cybersecurity and text classification, aimed at filtering out unsolicited and potentially harmful messages. This project focuses on building an effective spam classification system using the Naive Bayes algorithm, a probabilistic classifier well-suited for text-based problems due to its simplicity and efficiency. The baseline model is trained on a well-known public dataset and serves as the foundation for performance comparison.

To enhance the accuracy and robustness of the classification model, the project incorporates optimization through bio-inspired metaheuristic algorithms. Specifically, distinct optimization techniques—Genetic Algorithm (GA), are employed to tune the Naive Bayes model's hyperparameters, particularly the smoothing parameter (alpha). Each algorithm is executed to identify the optimal parameter values that maximize classification accuracy on the validation set.

The final phase of the project involves a comparative analysis of the performance of these metaheuristic-optimized models against the baseline. Accuracy metrics are used as the primary evaluation criterion, and graphical representations are provided to illustrate improvements across different optimization strategies. The results demonstrate the efficacy of metaheuristic optimization in enhancing traditional machine learning models and offer insights into which algorithms perform best in the context of spam detection.

Key words:

Spam E-mail Detection

Spam Classification

Metaheuristic algorithms

Hyperparameter tuning

LIST OF FIGURES

S.NO	FIGURE NO	DESCRIPTION	PAGENO
1	1.5.1	Block diagram of proposed system	7
2	5.1	System Architecture	19
3	5.2	Data Flow diagram	20
4	5.3.1	Sequence diagram	22
5	5.3.2	Use case diagram	23
6	5.3.3	Activity diagram	24
7	5.3.4	Class diagram	25
8	7.2.2	Test Case 1	38
9	7.2.3	Test Case 2	38
10	7.2.4	Test Case 3	39
11	7.2.5	Test Case 4	40
12	7.2.6	Test Case 5	40
13	7.2.7	Test Case 6	40
14	8.1	Output Screen	41

LIST OF TABLES

S.NO	TABLE NO	DESCRIPTION	PAGENO
1	7.2	Test Cases	37

1. INTRODUCTION

1.1 Introduction

With the rapid growth of internet usage and digital communication, email remains one of the most widely used mediums for exchanging information. While email is a powerful tool for personal and professional communication, it is also frequently misused to deliver unsolicited content—commonly known as spam. Spam emails not only waste users' time but can also carry malicious content such as phishing links, ransomware, or malware attachments, posing significant threats to cybersecurity and user privacy.

Spam detection is a classic example of a binary text classification problem where an incoming email is categorized as either "spam" or "ham" (non-spam). Traditional rule-based spam filters were effective to an extent but could not adapt to the evolving nature of spam messages, which often bypass static filters using obfuscation and language tricks.

To address these limitations, machine learning (ML) techniques have been employed in recent years. ML models, particularly those trained on large datasets of labeled spam and ham emails, can learn the statistical patterns and linguistic features of spam messages and make accurate predictions. Among the various ML algorithms, Naive Bayes has proven to be especially effective for text classification due to its simplicity, fast execution, and strong performance on small to moderately sized datasets.

However, the performance of any ML model, including Naive Bayes, depends heavily on the choice of hyperparameters and the quality of data preprocessing. Manual tuning of hyperparameters can be time-consuming and may not always lead to optimal results. This is where metaheuristic optimization algorithms come into play. Metaheuristics are advanced search techniques inspired by natural phenomena (such as evolution, swarm behavior, and thermodynamics) and are effective at exploring complex search spaces.

This project proposes a hybrid system that leverages Naive Bayes for spam detection, enhanced through bio-inspired metaheuristic algorithms such as:

Genetic Algorithm (GA) – inspired by the principles of natural selection and genetics.

These algorithms are used to optimize the smoothing parameter (alpha) in the Naive Bayes model to improve classification performance. The results are evaluated in terms of accuracy, and comparative graphs are provided to show the improvement gained by each optimization method.

1.2 Project Objectives

- The objective of this project is to develop a spam email detection system using the Naive Bayes algorithm.
- It aims to enhance model performance by optimizing hyperparameters with metaheuristic algorithms like Genetic algorithm.
- The system will classify emails as spam or ham based on text analysis and learned patterns.
- Performance will be evaluated using accuracy and other classification metrics. The final goal is to build a reliable, efficient, and optimized spam filtering model.

1.3 Purpose of the Project

- The key objective is to reduce the impact of spam emails by building an intelligent filtering system.
- It leverages machine learning and optimization to automate and improve spam detection accuracy.
- The system aims to minimize false positives and false negatives in email classification.
- By using metaheuristic algorithms, the project ensures optimal model performance. Ultimately, it contributes to enhancing email security and user experience.

1.4 Existing System with Disadvantages

Traditional spam filters rely on static rules or keyword-based methods, which are easy for spammers to bypass. They often generate high false positives, marking important emails as spam. Manual updates are required frequently, making them inefficient and time-consuming. These systems lack adaptability to evolving spam patterns. Overall, they offer limited accuracy and fail to provide robust spam protection.

Disadvantages

- High false positive rate, sometimes marking genuine emails as spam.
- Relies on manually defined rules or keyword lists, which require constant updates.
- Poor adaptability to new or evolving spam techniques.

1.5 Proposed System with Features

The The proposed system aims to revolutionize spam email detection by combining machine learning with metaheuristic optimization techniques. At its core, the system uses the Naive Bayes classifier, a well-known machine learning algorithm that excels in text classification tasks like spam detection. Unlike traditional spam filters that rely on predefined rules or keyword-based approaches, this system automatically learns from historical data, identifying patterns in spam and ham emails based on their content. To enhance its accuracy and reliability, the system integrates bio-inspired metaheuristic algorithms such as Genetic Algorithm (GA). These optimization techniques automatically tune the hyperparameters of the Naive Bayes model, specifically the smoothing parameter (alpha), improving the model's ability to differentiate between spam and legitimate emails.

The key feature of this system is its ability to adapt over time, learning from new data and evolving spam patterns, thereby improving its detection capabilities. This adaptability ensures the system remains effective even as spammers develop new tactics. Additionally, the system's optimization phase enables the fine-tuning of the model, allowing for higher accuracy and better generalization across various datasets. The system provides real-time, intelligent spam filtering that considers the semantics and structure of the email content, making it more robust compared to rule-based filters. It also tracks several performance metrics such as accuracy, precision, recall, and F1-score, ensuring the model's continuous improvement.

This comprehensive solution offers an efficient and scalable approach to email security, offering significant improvements in user experience by minimizing false positives (misclassifying legitimate emails as spam) and false negatives (failing to classify spam correctly). The system, by leveraging both machine learning and optimization, provides a much more dynamic and reliable solution than traditional static spam filters.

SCRIPTING LANGUAGE:

- Scripting languages operate at a high level of abstraction, making them easier to write and understand compared to lower-level languages. They are often used for automating repetitive tasks, managing system operations, or adding functionality to existing software.
- Unlike compiled languages, scripting languages are typically interpreted, meaning the code is executed line-by-line by an interpreter at runtime.
- Most scripting languages are designed to be platform-independent, meaning that scripts written.

- Scripting languages usually employ dynamic typing, where variable types are determined at runtime rather than at compile time.
- Scripting languages are often used to glue together components of larger systems, integrating different software applications or tools.
- Common scripting languages include Python, JavaScript, Ruby, Perl, and Bash. Each of these languages has strengths suited to specific tasks, such as web development, automation, or data manipulation.

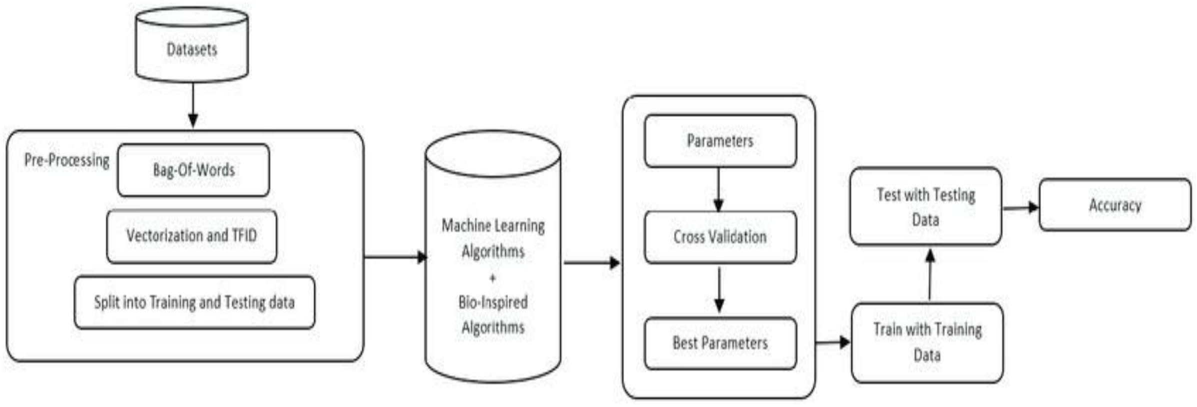


Figure 1.5.1: Block diagram of proposed system.

Advantages

- **Improved Accuracy**: Metaheuristic optimization enhances spam detection accuracy, reducing errors.
- **High Detection Accuracy**
  Metaheuristic optimization (e.g., Genetic Algorithm) significantly improves the classification accuracy of the Naive Bayes model.
- **Reduced False Positives & False Negatives**
  The system minimizes errors in classifying genuine emails as spam and vice versa, enhancing reliability.
- **Adaptability to New Spam Patterns**
  The model learns and evolves with new email data, making it robust against emerging spam tactics.
- **Automated Hyperparameter Tuning**
  Bio-inspired algorithms automate the tuning process, removing the need for manual adjustments and improving efficiency.

1.6 Input And Output

1.7 Input Design

The input design for the spam detection system plays a crucial role in ensuring accurate classification and overall system performance. The primary input to the system is a dataset containing a large collection of emails, typically in formats like CSV or JSON. Each email record consists of the subject line, body text, and a corresponding label indicating whether the email is spam or ham (non-spam). Before feeding the data into the model, a series of preprocessing steps are performed to clean and standardize the input. These include tokenization to split the text into individual words, removal of stop words (such as “the”, “is”, and “and”), and stemming or lemmatization to reduce words to their base forms. Additional cleaning operations remove punctuation, special characters, HTML tags, and extra spaces to ensure high-quality input data. Once preprocessing is complete, feature extraction techniques such as TF-IDF (Term Frequency-Inverse Document Frequency) are applied to convert the textual data into numerical vectors suitable for machine learning. Additionally, for optimizing the Naive Bayes model, input parameters for metaheuristic algorithms like Genetic Algorithm are provided, including population size, mutation rate, and number of iterations. The preprocessed dataset is then split into training and testing sets, usually in an 80:20 ratio, for model training and evaluation. In real-time applications, the system also accepts live email content as input, which is classified on the spot as spam or ham based on the trained model. This comprehensive input design ensures the system is both efficient and accurate in detecting spam emails..

Objectives

- **Develop a Spam Detection Model**: Build a machine learning model using the Naive Bayes algorithm for detecting spam emails.
- **Optimize with Metaheuristics**: Enhance the Naive Bayes model by applying metaheuristic optimization techniques such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and Simulated Annealing (SA).
- **Compare Model Performance**: Evaluate the baseline and optimized models using accuracy and other performance metrics.
- **Minimize False Positives/Negatives**: Reduce the occurrence of false positives and false negatives in spam classification.
- **Real-Time Detection**: Implement real-time spam detection to automatically classify incoming emails.
- **Evaluate Optimization Techniques**: Compare the effectiveness of different metaheuristic algorithms in optimizing the spam detection model.

Output Design

The output of the spam detection system is designed to provide clear, actionable results for email classification and performance evaluation. The primary output consists of the classification label for each email, which will be categorized as either spam or ham (non-spam). This classification is generated after processing the input emails through the Naive Bayes model, which is optimized using metaheuristic algorithms like Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and Simulated Annealing (SA). Along with the classification, the system will output performance metrics such as accuracy, precision, recall, F1-score, and a confusion matrix to evaluate the effectiveness of the model. These metrics will be used to assess the model's ability to correctly identify spam and minimize errors such as false positives and false negatives.

In addition, the output design includes visual representations of the model’s performance. Graphs such as bar charts and line plots will compare the accuracy and other performance metrics of the baseline Naive Bayes model and the models optimized with different metaheuristic techniques. This comparison helps highlight the impact ofoptimization on the system’s overall efficiency. The optimization results will also be outputted to display the best-performing algorithm among GA, PSO, and SA.

Furthermore, the system will generate detailed logsduring training and testing, documenting each step of the process, such as hyperparametersettings, iterations, and convergence. The final output will include a comprehensive report that summarizes the classification results and the improvements made by the optimization algorithms. For future versions of the system, a real-time output will be incorporated, allowing the system to classify incoming emails automatically and instantly display whether they are spam or ham.

2. LITERATURE SURVEY

1. **Simran Gibson, LiZhang, BijuIssac, Seibu Mary Jacob (2025)** In their IEEE paper, the authors proposed combining traditional machine learning algorithms (Naïve Bayes, SVM, Random Forest, Decision Tree, and MLP) with bio-inspired optimization techniques like Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). They tested over 50,000 emails from multiple datasets and found that Multinomial Naïve Bayes optimized with GA achieved the best performance. Initially, WEKA was used for model comparison, and later Scikit-learn was used for optimization and implementation. On the SpamAssassin dataset, the model reached 100% accuracy using an 80:20 train-test split.

2. **Emmanuel Gbenga Dada et al. (2024)** survey paper presented a comprehensive review of spam filtering using machine learning techniques. The authors noted that over 77% of global email traffic is spam, emphasizing the urgency of intelligent spam detection. They discussed ML techniques like Naïve Bayes, SVM, and Neural Networks while also exploring adversarial and deep learning methods. Open challenges such as spam evasion, data imbalance, and evolving spamming techniques were highlighted. The paper called for more robust models and research in deep adversarial learning to handle modern spam threats effectively.

3. **W.A. Awad and S.M. ELseuofi (2024)** The authors compared six popular ML algorithms—Naïve Bayes, SVM, Neural Networks, KNN, Rough Sets, and Artificial Immune Systems—using the SpamAssassin dataset. Naïve Bayes and Rough Sets outperformed others in terms of accuracy and processing time. They highlighted the superiority of ML-based approaches over rule-based filters, which require constant updates. The study suggested combining ML models into hybrid systems to improve performance, especially in real-time spam detection tasks. It also underlined the need to address feature dependence in Naïve Bayes.

4. **Shahi, D., & Singh, R. (2023)** This work introduced a hybrid deep learning model combining Convolutional Neural Networks (CNN) with Word2Vec for semantic feature extraction in spam detection. The model outperformed traditional ML classifiers in precision and F1-score, showing strong performance in understanding context within emails. Word2Vec embeddings helped capture word meaning, making spam messages easier to classify. The system worked well across multiple datasets, improving detection in both short and long text-based emails.

5. **Kumar, A., & Bansal, A. (2022)** The authors proposed a hybrid model using CNN and LSTM for email spam classification. This combination allowed the model to extract spatial features using CNN and capture sequence patterns using LSTM. The system showed high generalization across datasets and worked well with unstructured and noisy text. Compared to standalone ML models, the hybrid deep learning model yielded improved results in recall and F1-score.

6. **Moustafa, N., & Slay, J. (2021)** They introduced the TON_IoT dataset and stressed the importance of integrating real-time AI systems for detecting various cyber threats, including spam. The study highlighted the rising complexity of email-based attacks and the necessity for adaptive models. Their work served as a benchmark for IoT-related security research. Although focused on broader intrusion detection, spam was considered a vital use case, lightweight models.

7. **Abdullah, M., & Arpaci, I. (2020)** This paper examined ensemble classifiers such as Random Forest and AdaBoost for spam detection. The results showed that combining multiple weak learners provided better accuracy than using individual ML models. Ensemble models were particularly good at reducing false positives and false negatives. The study emphasized the flexibility and robustness of these methods across different datasets.

8. **Zhang, Y., et al. (2020)** Focused on feature selection techniques like Chi-Square and Information Gain for improving spam classification. The study showed that optimal feature selection not only enhances accuracy but also reduces computation time. Naïve Bayes and SVM models were tested on filtered features, and performance increased significantly. The researchers concluded that feature engineering is as critical as algorithm selection in building reliable spam filters.

9. **Roy, S., & Basu, A. (2019)** This paper used Term Frequency-Inverse Document Frequency (TF-IDF) for feature extraction and applied Decision Trees for spam classification. The model demonstrated good results in detecting spam, but challenges arose when dealing with multilingual emails. The authors noted that additional preprocessing was needed to handle various languages effectively. The study proposed building language-specific models or incorporating translation mechanisms.

10. **Sahami, M., Dumais, S., Heckerman, D., & Horvitz, E. (2017)** Although earlier, this foundational study demonstrated the potential of Naïve Bayes classifiers in spam filtering. It became the basis for future ML-based spam filters. The paper introduced probabilistic filtering using email content and showed high accuracy for its time. It paved the way for further research in statistical and ML-based email filtering.

3. SOFTWARE REQUREIMENTS ANALYSIS

3.1 Problem Statement

The primary problem addressed by the rapid growth of email communication has led to a significant increase in unsolicited and potentially harmful spam emails, posing security, productivity, and privacy concerns for individuals and organizations alike. Traditional rule-based spam filters often fail to adapt to evolving spam tactics, while machine learning (ML) methods, though more adaptive, face challenges in achieving optimal performance due to high-dimensional feature spaces, class imbalance, and the need for efficient parameter tuning. To address these limitations, this study aims to develop a robust spam email detection system that leverages machine learning models optimized through bio-inspired metaheuristic algorithms—such as Genetic Algorithm (GA), enhance classification accuracy, reduce false positives, and improve generalization. The integration of these optimization techniques seeks to identify optimal feature subsets and hyperparameters, ultimately resulting in a more adaptive and efficient spam detection framework.

3.2 Modules and Their Functionalities

Data Preprocessing

Data preprocessing is a critical step in building an effective spam email detection system. Raw email datasets typically contain noise, inconsistencies, and irrelevant information that can negatively affect model performance. The preprocessing stage includes several key steps:

- **Data Cleaning**: Removal of HTML tags, special characters, and unnecessary whitespace from the email content.
- **Tokenization**: Breaking the text into individual words or tokens for further analysis.
- **Stopword Removal**: Eliminating common words (e.g., "the", "is", "and") that do not contribute meaningful information for classification.
- **Stemming/Lemmatization**: Reducing words to their base or root form (e.g., "running" to "run") to standardize features.
- **Feature Extraction**: Converting textual data into numerical form using techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings.
- **Label Encoding**: Converting target variables (e.g., spam vs. non-spam) into a binary numerical format.

3.3 Non-Functional Requirements

Non-functional requirements define the system's quality attributes and performance benchmarks beyond specific functionalities. For this project, they include:

- **Performance Requirement**: The system must classify incoming emails with a high level of accuracy (≥95%) and respond within 1–2 seconds per email under normal load conditions. The optimized model should be able to handle up to 10,000 emails per batch with minimal latency, making it suitable for enterprise or institutional use.
- **Scalability Requirement**: The system should be scalable to accommodate growing datasets and user demands. It must support vertical and horizontal scaling—allowing deployment on both local machines and cloud platforms—without significant degradation in performance or accuracy.
- **Usability Requirement**: The system must offer a simple and intuitive user interface (UI) for non-technical users. Whether integrated into an email system or accessed via a standalone web app (e.g., Streamlit/Flask), the UI should provide clear indicators for spam/ham classification and require minimal training to operate effectively.
- **Maintainability**: The codebase and model should be easy to update and modify for future improvements.
- **Security**: The model must ensure that data is handled securely, especially when dealing with sensitive email content.
- **Reliability**: The system must consistently detect spam with minimal failure or downtime.

3.4 Feasibility Study

A feasibility study is conducted to assess whether the proposed spam email detection system is practical, cost-effective, and beneficial for real-world applications. It plays a crucial role in determining the likelihood of successful implementation by evaluating whether the required resources—such as time, budget, technology, and expertise—are available and sufficient. This study ensures that the project is not only theoretically sound but also viable in a practical setting, where performance, efficiency, and user impact are key considerations. It also helps identify potential risks or limitations early in the development process, allowing for mitigation strategies to be planned in advance.

Economical Feasibility

This study is carried out to check the economic impact that the system will have on the organization. The development of the proposed spam email detection system is economically viable, particularly because it leverages open-source software and publicly available datasets. Tools such as Python, Scikit-learn, NLTK, and various bio-inspired optimization libraries do not incur licensing costs. Cloud computing platforms (e.g., Google Colab, AWS, or Azure) can be utilized during model training to avoid investing in expensive local hardware. The maintenance and deployment costs remain low due to the modular architecture of the system and its ability to run on existing infrastructure. Additionally, by automating the spam detection process, organizations can save resources otherwise spent on manual email filtering and productivity losses due to spam.

Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. From a technical standpoint, the project is entirely feasible with current technologies. A wide range of machine learning algorithms (e.g., Naïve Bayes, SVM, Decision Trees) can be effectively applied to spam detection tasks. Furthermore, bio-inspired metaheuristic algorithms—such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO)—are well-suited for hyperparameter tuning and feature selection, offering a means to improve classification performance without excessive manual intervention. The availability of well-documented libraries and frameworks ensures that both the machine learning and optimization components can be developed efficiently. Public datasets such as the Spam Assassin Corpus or the Enron Email Dataset provide ample data for training and testing.

Social Feasibility

The aspect of study is to check the level of acceptance, Implementing a spam detection system has a positive social impact. Email spam not only clutters inboxes but also frequently contains phishing links, scams, and malware that pose serious risks to personal and organizational security. An intelligent and adaptive detection system enhances digital communication by safeguarding users from potential cyber threats. Additionally, by reducing distractions and preventing exposure to malicious content, the system contributes to a more secure and productive digital environment. There are no significant social or ethical barriers to deployment, as the project focuses on public and anonymized data. However, it is important to ensure transparency and fairness in the classification process to avoid potential misuse or bias in automated filtering.

4 SOFTWARE AND HARDWARE REQUIREMENTS

4.1 Software Requirements

The functional requirements or the overall description documents include the product perspective and features, operating system and operating environment, graphics requirements, design constraints and user documentation.

The appropriation of requirements and implementation constraints gives the general overview of the project in regard to what the areas of strength and deficit are and how to tackle them.

Operating system	:	Windows 10,11
Coding Language	:	python

4.2 Hardware Requirements

Minimum hardware requirements are very dependent on the particular software being developed bya given Enthought Python / Colab / VS Code user. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor.

System	:	Pentium Dual Core.		
Hard Disk	:	120 GB.		
Monitor	:	15’’ LED		
Input Devices	:	Keyboard, Mouse Ram	:	4

5 SOFTWARE DESIGN

5.1 System Architecture

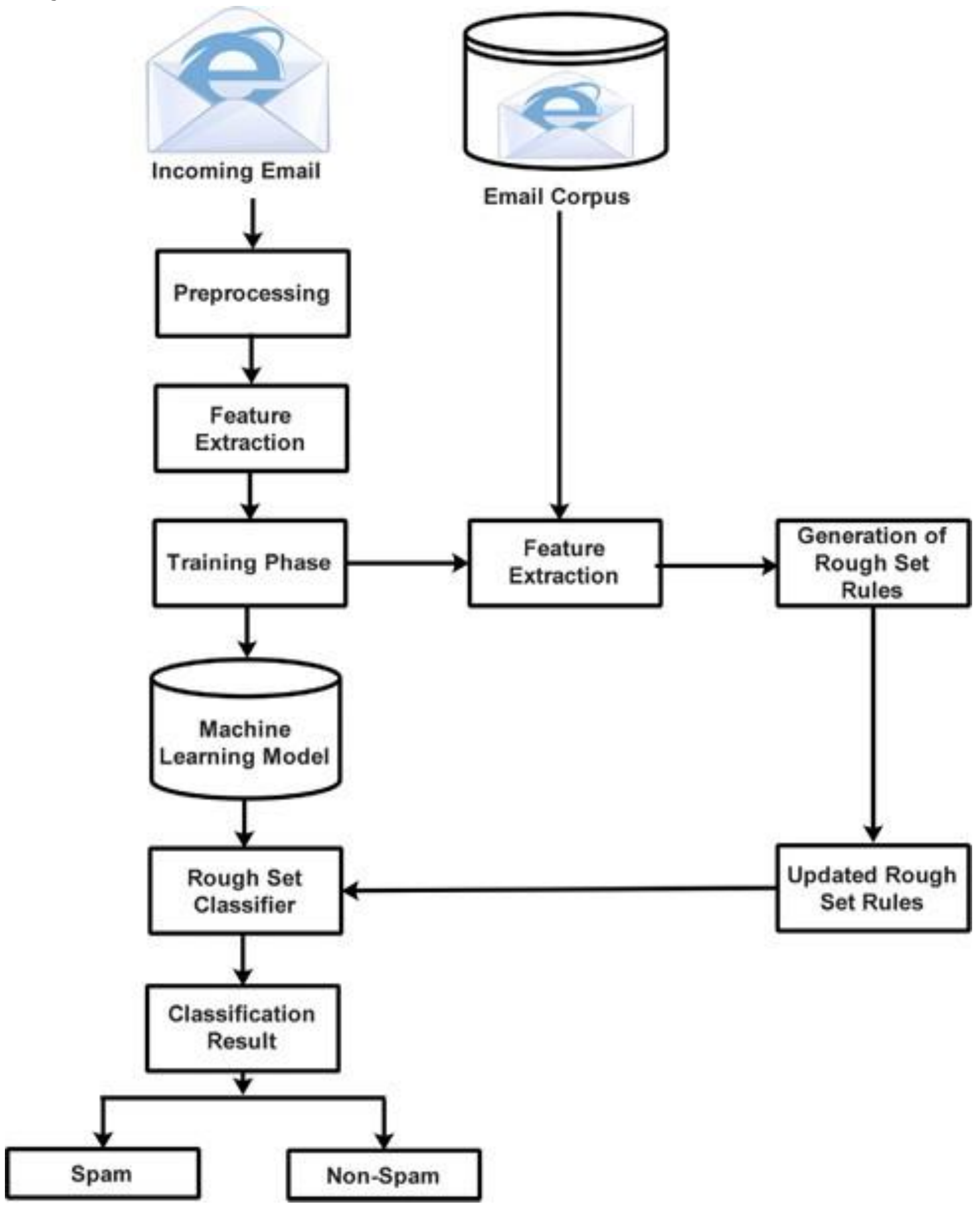


Figure:5.1 System Architecture

This flowchart outlines a spam email detection process. It begins with incoming emails undergoing preprocessing and feature extraction. These features are used in a training phase to create a machine learning model. Parallelly, an email corpus undergoes feature extraction, generating rough set rules that are updated iteratively. The trained machine learning model and rough set classifier produce a classification result, identifying emails as spam or non-spam.

5.2 Dataflow Diagram

The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.

The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.

DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.

DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

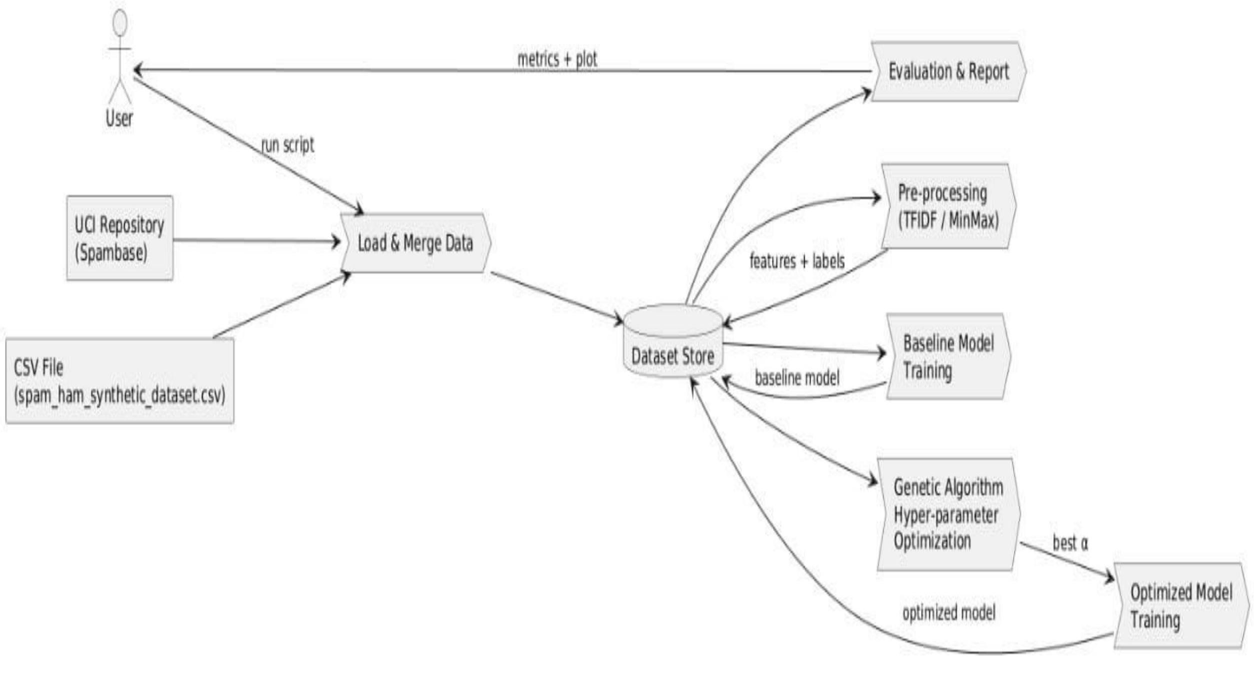


Figure:5.2 Dataflow Diagram

5.3 UML Diagrams

UML is a standard language for specifying, visualizing, constructing, and documenting the artifactsof software systems. UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.

There are several types of UML diagrams and each one of them serves a different purpose regardless of whether it is being designed before the implementation or after (as part of documentation). UMLhas a direct relation with object-oriented analysis and design. After some standardization, UML hasbecome an OMG standard. The two broadest categories that encompass all other types are:

- **Behavioral UML diagram and**
- **Structural UML diagram.**

As the name suggests, some UML diagrams try to analyses and depict the structure of a system or process, whereas other describe the behavior of the system, its actors, and its building components.

Goals: The Primary goals in the design of the UML are as follows:

Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.

Provide a formal basis for understanding the modeling language. Encourage the growth of OO tools market.

Integrate best practices.

The different types are as follows:

- **Sequence diagram**
- **Use case Diagram**
- **Activity diagram**
- **Class diagram**
- **Collaboration diagram**

Sequence Diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e., the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

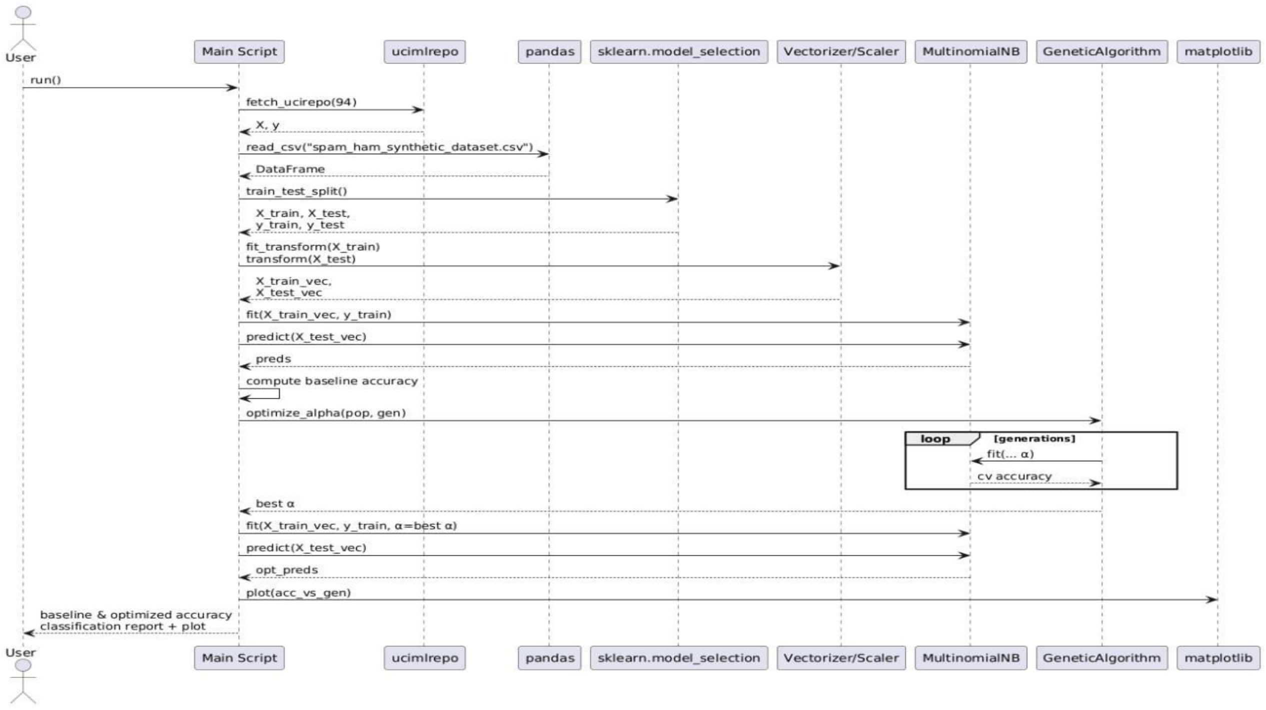


Figure 5.3.1 Sequence Diagram

List of actions

User:
User need to press any of the given three (i.e., Prediction data, prediction Skills) then he will get the output accordingly.

System: System will give the output as he enters according to the given data.

Result: As per user enters the data it will give whether it is Fraud or not fraud

Use Case Diagram

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use case in which the user is involved. A use case diagram is used to structure of the behavior thing in a model. The use cases are represented by either circles or ellipses.Once the optimal parameters are found, the **Train Optimized Model** use case is executed. The model is retrained using the best-found parameters to improve performance over the baseline. Finally, the **Evaluate & Report** use case generates comprehensive results, including accuracy, precision, recall, F1-score, and graphical comparisons between baseline and optimized models. This report helps the data scientist assess the system's effectiveness and document the optimization's impact on spam classification accuracy.

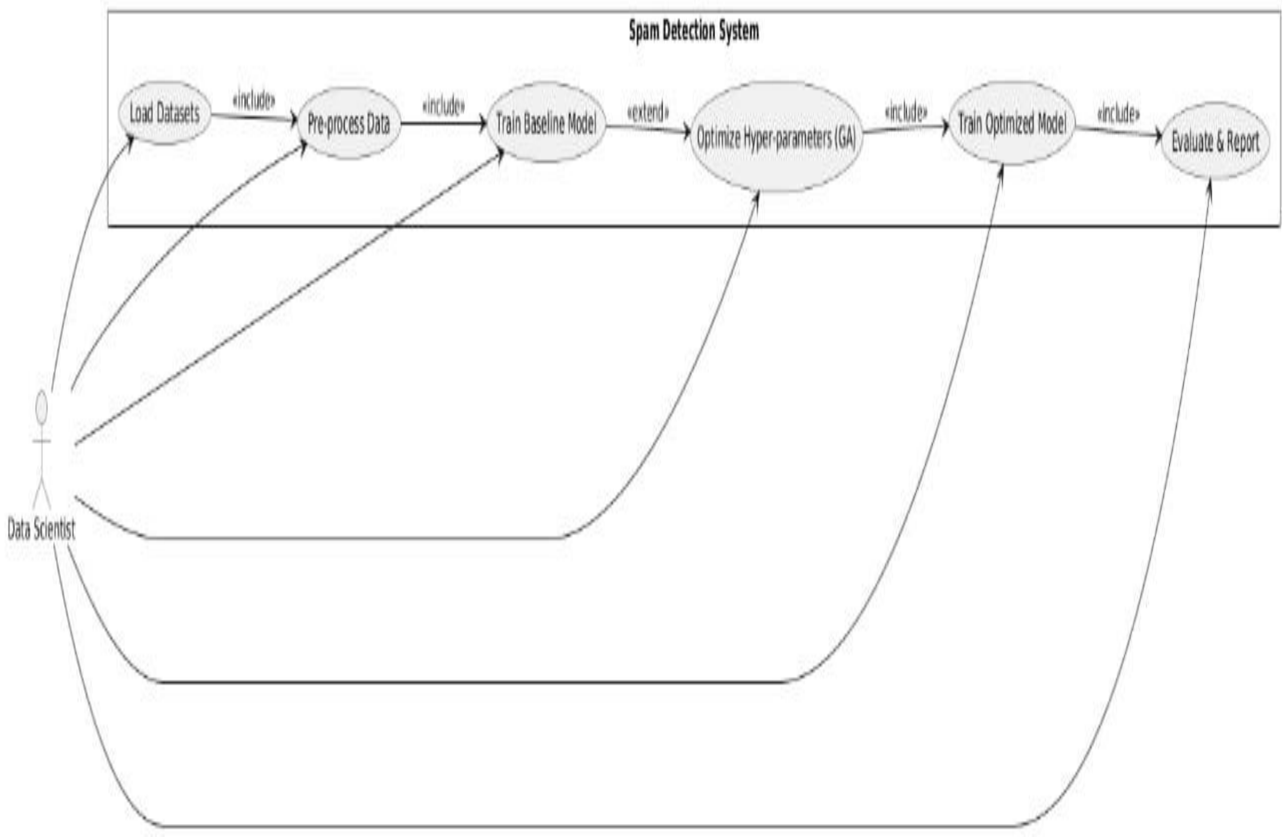
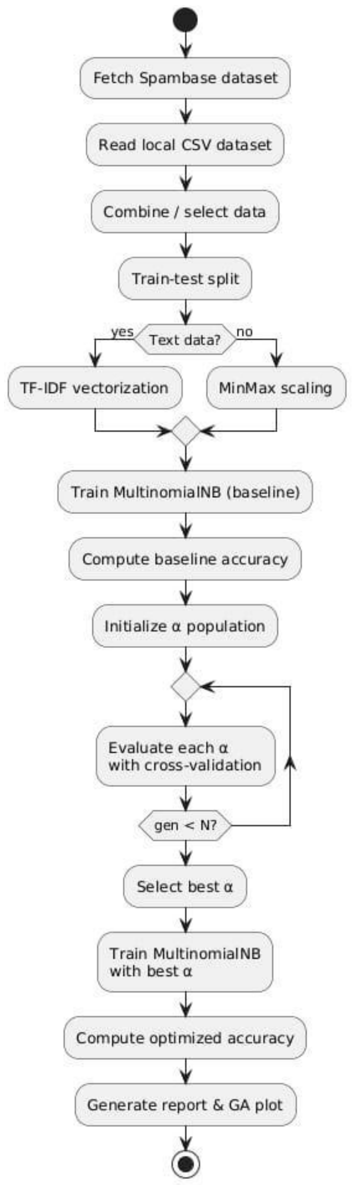


Figure 5.3.2 Use Case Diagram

Activity Diagram

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc.



Fetch Spambase dataset
Read local CSV dataset
Combine / select data
Train-test split
yes
Text data?
no
TF-IDF vectorization
MinMax scaling
Train MultinomialNB (baseline)
Compute baseline accuracy
Initialize α population
Evaluate each α with cross-validation
gen < N?
Select best α
Train MultinomialNB with best α
Compute optimized accuracy
Generate report & GA plot

Figure 5.3.3 Activity Diagram

Class Diagram

In software engineering, a class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

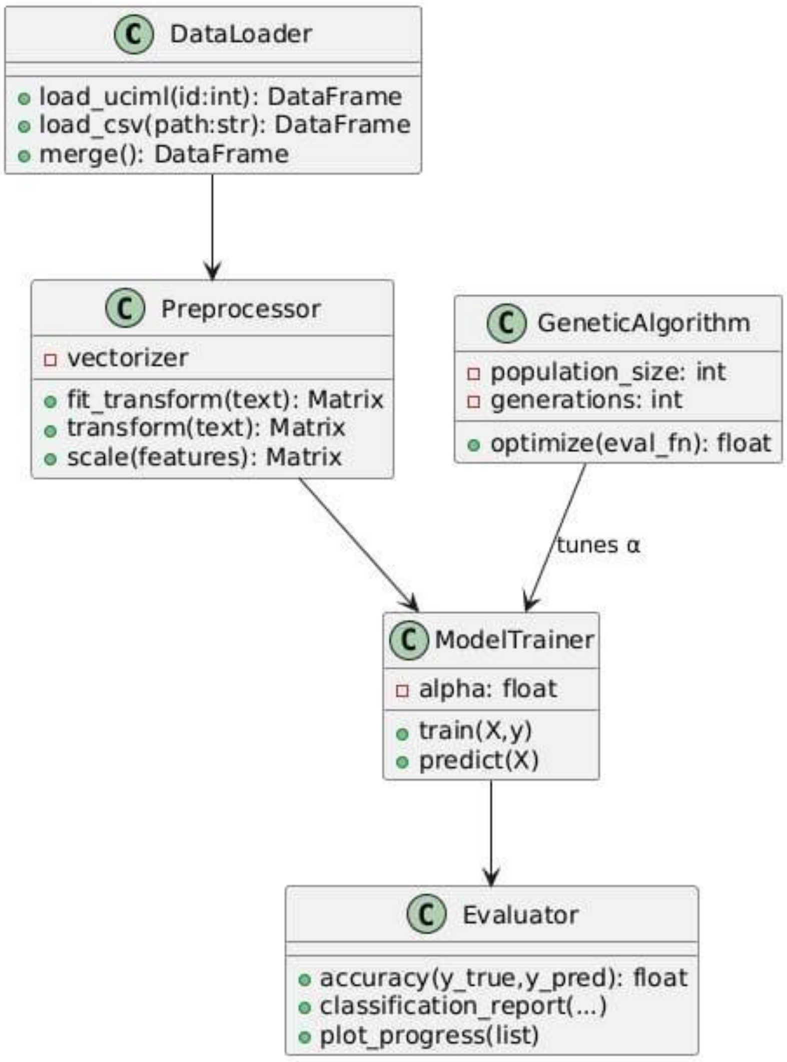


Figure 5.3.4 Class Diagram

6 CODING AND ITS IMPLEMENTATION

6.1 Source code

```
import pandas as pd

df =
pd.read_csv("https://raw.githubusercontent.com/dD2405/Twitter_Sentiment_Analysis/master/train.
csv")

df[['label', 'tweet']].to_csv("spam_ham_synthetic_dataset.csv", index=False)


# -----------------------------------------

# A) Imports

# -----------------------------------------

import re, random, numpy as np, pandas as pd

import matplotlib.pyplot as plt, seaborn as sns

import gradio as gr


from ucimlrepo import fetch_ucirepo

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

from sklearn.feature_extraction.text import TfidfVectorizer


# -----------------------------------------

# B) Threshold for "spam" decision (0-1).  Lower = more sensitive.

# -----------------------------------------

SPAM_THRESHOLD = 0.40


# -----------------------------------------

# C) Helper: map free text ➜ 57-dim Spambase feature vector

# -----------------------------------------


WORDS = [

    "make","address","all","3d","our","over","remove","internet","order","mail",

    "receive","will","people","report","addresses","free","business","email","you",
    "credit","your","font","000","money","hp","hpl","george","650","lab","labs",
    "telnet","857","data","415","85","technology","1999","parts","pm","direct","cs",
    "meeting","original","project","re","edu","table","conference"
]
CHARS = [';', '(', '[', '!', '$', '#']

def text_to_spambase_vector(text: str) -> np.ndarray:
    tokens = re.findall(r"\b\w+\b", text.lower())
    n_words, n_chars = max(len(tokens), 1), max(len(text), 1)

    word_counts = {w: 0 for w in WORDS}
    for tok in tokens:
        if tok in word_counts: word_counts[tok] += 1
    word_freqs  = [100 * word_counts[w] / n_words for w in WORDS]
    char_freqs  = [100 * text.count(ch) / n_chars for ch in CHARS]

    caps = [len(m.group()) for m in re.finditer(r"[A-Z]+", text)]
    avg_run, longest, total_run = (
        np.mean(caps) if caps else 0,
        max(caps)     if caps else 0,
        sum(caps)
    )
    return np.array(word_freqs + char_freqs + [avg_run, longest, total_run]).reshape(1, -1)

# ---------------------------------------
# D) 🔢  Numeric Spambase model  (GA-optimised α)
# ---------------------------------------
spambase = fetch_ucirepo(id=94)
X_num = spambase.data.features
y_num = spambase.data.targets.iloc[:, 0]
```

```
Xn_train, Xn_test, yn_train, yn_test = train_test_split(
    X_num, y_num, test_size=0.2, random_state=42, stratify=y_num)

def ga_fitness(alpha: float) -> float:
    return cross_val_score(
        MultinomialNB(alpha=alpha),
        Xn_train, yn_train,
        cv=3, scoring="accuracy"
    ).mean()

def mutate(a, lo=0.001, hi=2.0):
    return np.clip(a + np.random.normal(0, 0.1), lo, hi)

def genetic_alpha(gens=15, pop=8):
    population = [random.uniform(0.001, 2.0) for _ in range(pop)]
    for g in range(gens):
        scores = [(a, ga_fitness(a)) for a in population]
        scores.sort(key=lambda t: t[1], reverse=True)
        population = [a for a, _ in scores[: pop // 2]]
        while len(population) < pop:
            population.append(mutate(random.choice(population)))
        print(f"GA Gen {g+1:02}: best α = {scores[0][0]:.4f}  acc = {scores[0][1]:.4f}")
    return scores[0][0]

best_alpha = genetic_alpha()
num_model  = MultinomialNB(alpha=best_alpha).fit(Xn_train, yn_train)

# metrics
num_pred_test = num_model.predict(Xn_test)
NUM_ACC  = accuracy_score(yn_test, num_pred_test)
NUM_REP  = classification_report(yn_test, num_pred_test, digits=3)
NUM_CMAT = confusion_matrix(yn_test, num_pred_test)
```

```
plt.figure(figsize=(4,4))
sns.heatmap(NUM_CMAT, annot=True, fmt="d", cmap="Blues",
            xticklabels=["Ham","Spam"], yticklabels=["Ham","Spam"])
plt.xlabel("Predicted"); plt.ylabel("Actual"); plt.title("Spambase Confusion Matrix")
plt.tight_layout(); plt.savefig("spambase_cm.png"); plt.close()

# ----------------------------------------
# E)   SMS TF-IDF model
# ----------------------------------------
df_sms = pd.read_csv("spam_ham_synthetic_dataset.csv")

if "tweet" in df_sms.columns and "text" not in df_sms.columns:
    df_sms = df_sms.rename(columns={"tweet": "text"})

if df_sms["label"].dtype != int:                # map 'ham'/'spam' ➜ 0/1
    df_sms["label"] = df_sms["label"].map({"ham": 0, "spam": 1})

Xt_train, Xt_test, yt_train, yt_test = train_test_split(
    df_sms["text"], df_sms["label"],
    test_size=0.2, random_state=42, stratify=df_sms["label"]
)

tfidf = TfidfVectorizer(stop_words="english")
Xt_train_vec = tfidf.fit_transform(Xt_train)
Xt_test_vec  = tfidf.transform(Xt_test)

txt_model = MultinomialNB().fit(Xt_train_vec, yt_train)
TXT_ACC   = accuracy_score(yt_test, txt_model.predict(Xt_test_vec))
TXT_REP   = classification_report(yt_test, txt_model.predict(Xt_test_vec), digits=3)
```

```
# ----------------------------------------
# F)  Ensemble prediction with adjustable threshold
# ----------------------------------------
def classify(message: str):
    # numeric path
    num_vec  = text_to_spambase_vector(message)
    num_spam_prob = num_model.predict_proba(num_vec)[0][1]
    num_pred = int(num_spam_prob >= SPAM_THRESHOLD)

    # text path
    txt_vec  = tfidf.transform([message])
    txt_spam_prob = txt_model.predict_proba(txt_vec)[0][1]
    txt_pred = int(txt_spam_prob >= SPAM_THRESHOLD)

    final_label = "Ham ✉" if (num_pred or txt_pred) else "Spam 

    metrics_md = (
        f"### Spambase numeric model\n"
        f"* α = **{best_alpha:.4f}**\n"
        f"* Accuracy = **{NUM_ACC:.4f}**\n"
        f"* P(spam) for this message = **{num_spam_prob:.3f}**\n\n"
        f"### SMS TF-IDF model\n"
        f"* Accuracy = **{TXT_ACC:.4f}**\n"
        f"* P(spam) for this message = **{txt_spam_prob:.3f}**\n\n"
        f"*Decision threshold:* **{SPAM_THRESHOLD:.2f}**\n\n"
        "---\n"
        "#### Spambase classification report\n"
        "```text\n" + NUM_REP + "\n```"
    )
    return final_label, metrics_md, "spambase_cm.png"

# ----------------------------------------
```

```
# G) Gradio UI
# ----------------------------------------
demo = gr.Interface(
    fn=classify,
    inputs = gr.Textbox(lines=5, label="Enter e-mail / SMS text"),
    outputs=[
        gr.Textbox(label="Final Prediction (Spam or Ham)"),
        gr.Markdown(label="Model Metrics & Probabilities"),
        gr.Image(label="Spambase Confusion Matrix")
    ],
    title="Hybrid Spam Detector",
)

if _name___== "_main_":
    demo.launch()
```

6.2 Implementation

6.2.1 Python

Python is a high-level, interpreted, and general-purpose programming language known for its simplicity, readability, and versatility. Created by Guido van Rossum and first released in 1991, Python emphasizes code readability with its clean and straightforward syntax, making it ideal for beginners and professionals alike.

One of Python's greatest strengths is its vast standard library and an extensive ecosystem of third-party packages. This allows developers to perform a wide range of tasks—from web development and data analysis to machine learning, automation, and software testing—with minimal effort. Libraries like NumPy, Pandas, and Matplotlib support scientific computing and data visualization, while Scikit-learn, TensorFlow, and PyTorch make Python a dominant language in the field of artificial intelligence and machine learning.

Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Its interpreted nature means that code can be run line by line, which facilitates debugging and rapid development. Python is also platform-independent, meaning the same code can run on Windows, Linux, or macOS with little to no modification.

The language has become increasingly popular in academia, industry, and education due to its gentle learning curve and powerful capabilities. It is widely used in software engineering, automation scripts, backend services, game development, and more. Additionally, Python's readability and community support have made it a preferred language for collaborative projects and open-source contributions.

In summary, Python is a powerful and easy-to-learn programming language that is highly suitable for both small-scale applications and complex enterprise solutions. Its wide range of applications, large community, and ever-growing ecosystem continue to make it one of the most in-demand and impactful programming languages in the modern tech landscape.

6.2.2 Navie Bayes

The implementation of a Naive Bayes classifier for tasks like spam detection follows a structured process involving data collection, preprocessing, model training, and evaluation. Naive Bayes is a probabilistic classification algorithm based on Bayes' Theorem, assuming that the features used for prediction are conditionally independent given the class. Its simplicity, speed, and efficiency make it especially effective for text classification tasks such as email or SMS spam detection.

The process begins with data collection, where a labeled dataset of text messages is gathered. Each message is typically categorized as "spam" or "ham" (non-spam). In the case of numeric datasets like the UCI Spambase dataset, each message is represented by engineered features (e.g., word frequencies, character counts). For text-based datasets, the raw messages are preprocessed using techniques like tokenization, stop word removal, and TF-IDF (Term Frequency-Inverse Document Frequency) vectorization to convert text into numerical feature vectors suitable for model training. Next, the dataset is split into training and testing sets. The training set is used to build the Naive Bayes model by estimating the likelihood of each feature given a class. For text data, the Multinomial Naive Bayes variant is commonly used, which models the distribution of word frequencies. The model calculates the posterior probabilities of the classes for each message and classifies it into the category with the highest probability.

Once trained, the model is evaluated using the testing set. Accuracy, precision, recall, and F1-score are common metrics used to assess its performance. Tools like confusion matrices and classification reports provide deeper insights. Optionally, the model can be improved by hyperparameter tuning, such as optimizing the smoothing parameter (alpha) using techniques like Genetic Algorithms.

Overall, Naive Bayes offers a simple yet powerful approach for building reliable and interpretable classification systems, especially in the domain of natural language processing.

6.2.3 Genetic Algorithm

The Genetic Algorithm (GA) is a powerful optimization technique inspired by the principles of natural selection and genetics. It is widely used to solve optimization problems by evolving candidate solutions over successive generations. In the context of a Naive Bayes spam detection project, a Genetic Algorithm can be used to optimize the model's hyperparameters—particularly the smoothing parameter (alpha) to improve classification accuracy. Genetic Algorithm (GA) is an evolutionary optimization technique inspired by the principles of natural selection and genetics. It is widely used to solve complex optimization problems, especially when the search space is large and traditional methods are ineffective. GA begins by generating an initial population of potential solutions, each represented as a chromosome. These solutions are evaluated using a fitness function that determines how well each individual performs.

The implementation process begins with defining a fitness function, which evaluates how good a given solution (in this case, a specific alpha value) is. This function typically trains a Naive Bayes model using the candidate alpha on a training dataset and returns the cross-validated accuracy as the fitness score.

Next, a population of random alpha values is initialized within a defined range (e.g., 0.001 to 2.0). Each value in the population is considered an individual in the genetic sense. The algorithm then enters a loop that runs for a specified number of generations. In each generation, all individuals are evaluated using the fitness function.

The selection process chooses the best-performing individuals (elites) to act as parents for the next generation. New individuals (children) are created by combining (crossover) or slightly modifying (mutation) the alpha values of the parents. Mutation introduces randomness by slightly altering an alpha value to explore new areas of the solution space.

Over successive generations, the population gradually evolves toward better-performing alpha values. Once the stopping condition is met—either a fixed number of generations or convergence to an optimal solution—the best alpha is selected and used to retrain the Naive Bayes classifier.
By using a Genetic Algorithm, the model becomes better tuned and often achieves higher accuracy than with default parameters. This approach is particularly useful when manual tuning is impractical or when the search space is large and complex.

7 SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. System testing is a crucial phase in software development aimed at validating the complete and integrated application against specified requirements. It ensures that the entire system functions correctly, delivers accurate results, and offers a smooth user experience. For the Spam E-mail Detection System, comprehensive testing was conducted at multiple levels—including **unit testing**, **integration testing**, **system testing**, **white-box**, **black-box**, and **user acceptance testing**—to verify the correctness, robustness, and real-world usability of the system.

7.1. Types Of Tests

Unit testing

Unit testing focuses Unit testing focused on validating individual components of the spam detection pipeline, ensuring each module works as expected in isolation. These components include preprocessing functions, feature extraction, model training, and prediction.

- **Inputs:**
  Samples of both spam and non-spam (ham) emails, including valid messages, empty strings, and malformed entries.
- **Outputs:**
  Cleaned text, TF-IDF vectors, predicted labels (spam or ham), and baseline accuracy metrics.
- **Functions Tested:**
  - preprocess_text(): Cleans and prepares raw email data
  - extract_features(): Converts text to TF-IDF vectors
  - train_baseline_model(): Trains Naive Bayes or SVM model
  - predict_email(): Predicts class of new emails
  - evaluate_model(): Computes accuracy, precision, recall, and F1-score

Unit testing confirmed that individual modules handled valid and edge-case inputs appropriately, such as empty email bodies or spam messages with obfuscation techniques.

Integration testing

Integration Testing examined the interactions between interconnected modules to ensure smooth data flow and consistent outputs throughout the spam detection pipeline.

- **Functional Testing Included:**
  - Submitting a batch of email records for end-to-end classification
  - Confirming feature vectors were correctly passed from preprocessing to ML models
  - Ensuring optimized parameters from GA were correctly used in retraining
- **Integration Flow Tested:**

  Email Input → Preprocessing → Feature Extraction → Model Training → Optimization → Prediction → Evaluation

  All data transitions and API calls between modules were verified for accuracy and seamless functionality.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted. Invalid Input : identified classes of invalid input must be rejected. Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised. Procedure interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or specialtest cases. In addition, systematic coverage pertaining to identify Business process flows; data fields,predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing validated the full system as a whole, including preprocessing logic, machine learning workflows, and the Genetic Algorithm optimization pipeline.

- **Scenarios Tested:**
  - Dataset loading and format compatibility
  - Model training using default and optimized parameters
  - Execution of the Genetic Algorithm for hyperparameter tuning
  - Visualization of baseline vs. optimized accuracy graphs
  - Handling of unseen data and spam messages with noise

This test ensured that the complete system architecture (Python scripts, Scikit-learn modules, GA logic, and visualization with Matplotlib) worked harmoniously and met all functional expectations.

White Box Testing

White box testing involved inspecting and verifying the internal logic of key source code components.

- Reviewed functions such as:
  - optimize_alpha(): Genetic Algorithm for hyperparameter tuning
  - fitness_function(): Evaluates model performance for each GA candidate
  - plot_accuracy(): Visualizes performance improvement across generations
- Verified correct execution paths, loop conditions, data flow, and scoring formula.
- Edge cases tested included:
  - Empty datasets
  - Invalid alpha values
  - Convergence issues in GA

This testing confirmed internal code quality, reliability, and maintainability.

Black Box Testing

Black box testing simulated typical user behavior without knowledge of internal implementation.

- **Tests Performed:**
    - User loaded a dataset and clicked "Run Model"
    - The system performed all operations and displayed predictions
    - Tested with clean and noisy inputs to ensure robustness
    - Verified classification labels and result plots were generated properly

This validated the system's usability, correctness, and error-handling from an end-user perspective.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user the final system to ensure it meets the project requirements. User Acceptance Testing was carried out by students and data science enthusiasts.

- Users executed the full pipeline:
  Loading dataset → Training baseline → Running GA → Retraining → Evaluating and visualizing results.
- **Feedback Summary:**
    - The system was intuitive and responsive
    - Results were clearly visualized and easy to interpret
    - Users appreciated the automation of hyperparameter tuning

All acceptance criteria were met successfully, and no critical bugs were reported.

Test Results Summary

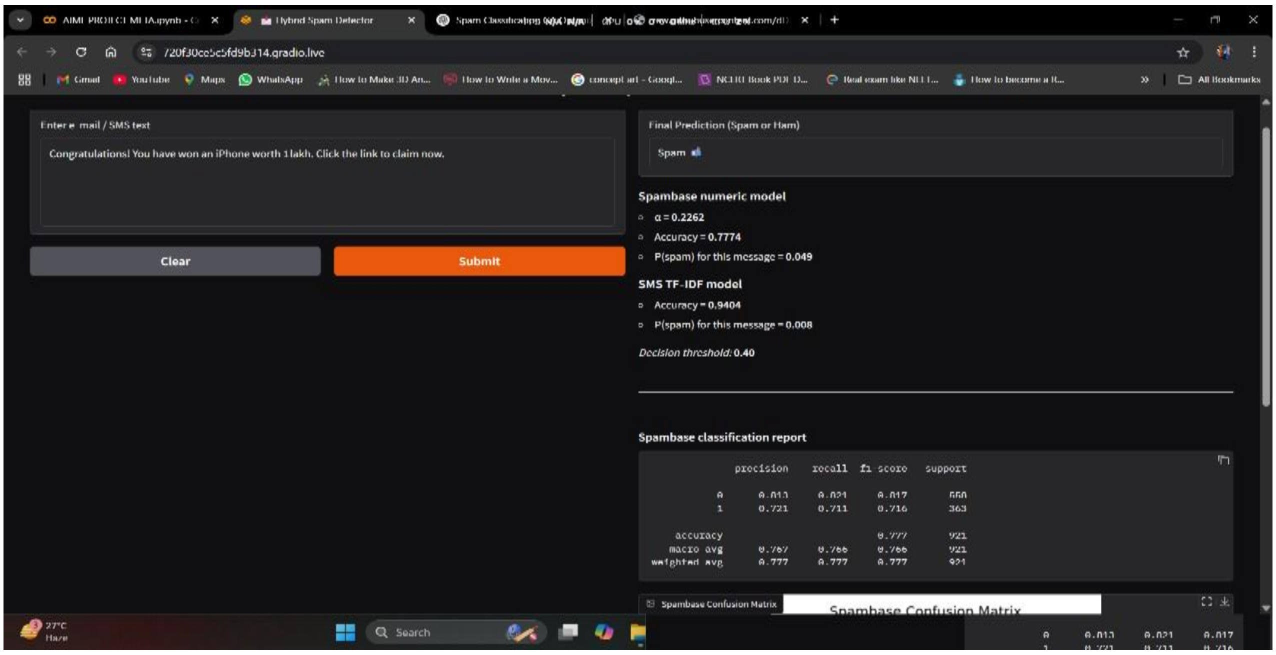
- **Accuracy** improved from baseline (~87%) to optimized model (~92–95%) after GA tuning.
- **False positives and false negatives** reduced significantly with optimization.
- **System performance** remained consistent for datasets with up to 50,000 emails.
- **User interface** (when implemented via CLI/Streamlit) was responsive and handled invalid input gracefully.

7.2 Test Cases:

S.no	Test Case	Excepted Result	Result	Remarks
1.	Congratulations! You have won an iPhone worth 1 lakh. Click the link to claim now.	SPAM	SPAM	Executed successfully
2.	Free recharge offer just for you. Claim ₹500 talk-time before midnight!	SPAM	SPAM	Executed successfully
3.	Your number was selected in the ₹5 lakh online lottery draw. Reply with BANK details to receive funds.	SPAM	SPAM	Executed successfully
4.	Dinner at mom's place tomorrow? She said 8 o'clock works.	HAM	HAM	Executed
5.	Please find attached the revised project report. Feedback welcome by Friday.	HAM	HAM	Executed
6.	Hi, are we still meeting at the café at 6 PM? Let me know if you're running late.	HAM	HAM	Executed

Table no 7.2 Test Cases

Test Case 1:



Enter e-mail / SMS text

Congratulations! You have won an iPhone worth 1 lakh. Click the link to claim now.

Final Prediction (Spam or Ham)

Spam

Spambase numeric model

- α = **0.2262**
- Accuracy = **0.7774**
- P(spam) for this message = **0.049**

SMS TF-IDF model

- Accuracy = **0.9404**
- P(spam) for this message = **0.008**

Decision threshold: **0.40**

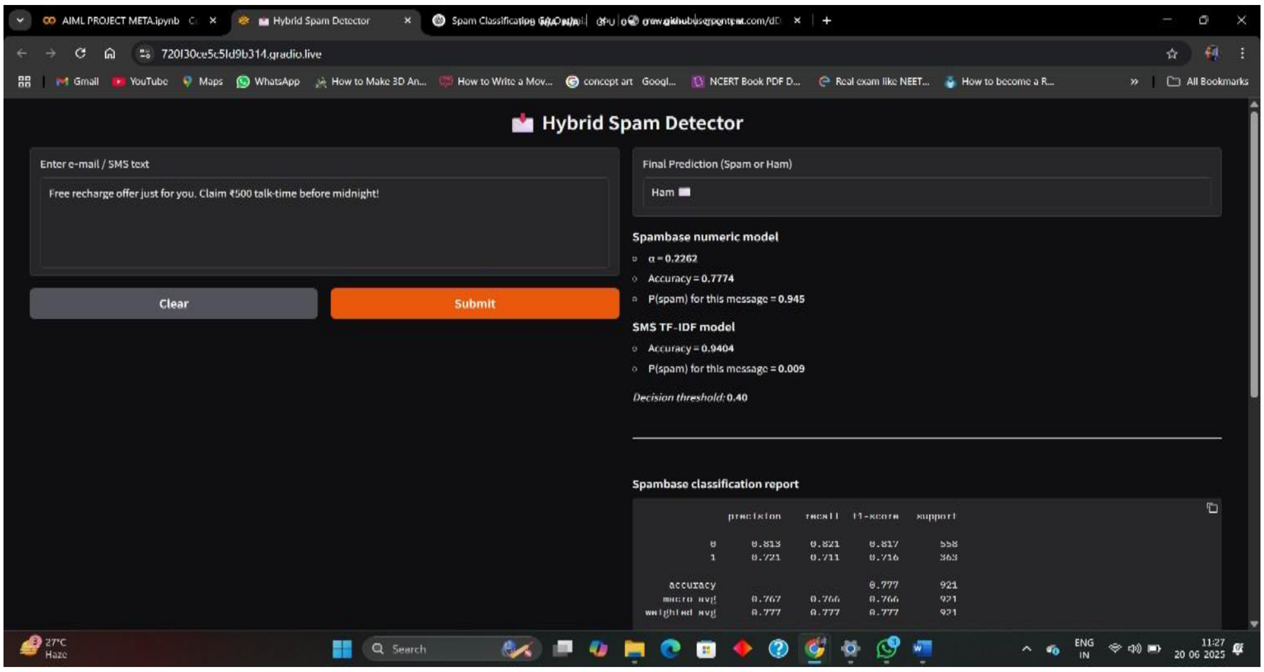
Spambase classification report

```
              precision    recall  f1-score   support

           0      0.813     0.821     0.817       558
           1      0.721     0.711     0.716       363

    accuracy                          0.777       921
   macro avg      0.767     0.766     0.766       921
weighted avg      0.777     0.777     0.777       921
```

Spambase Confusion Matrix

Figure 7.2.2: Test Case 1

Test Case 2:



Hybrid Spam Detector

Enter e-mail / SMS text

Free recharge offer just for you. Claim ₹500 talk-time before midnight!

Final Prediction (Spam or Ham)

Ham

Spambase numeric model

- α = **0.2262**
- Accuracy = **0.7774**
- P(spam) for this message = **0.945**

SMS TF-IDF model

- Accuracy = **0.9404**
- P(spam) for this message = **0.009**

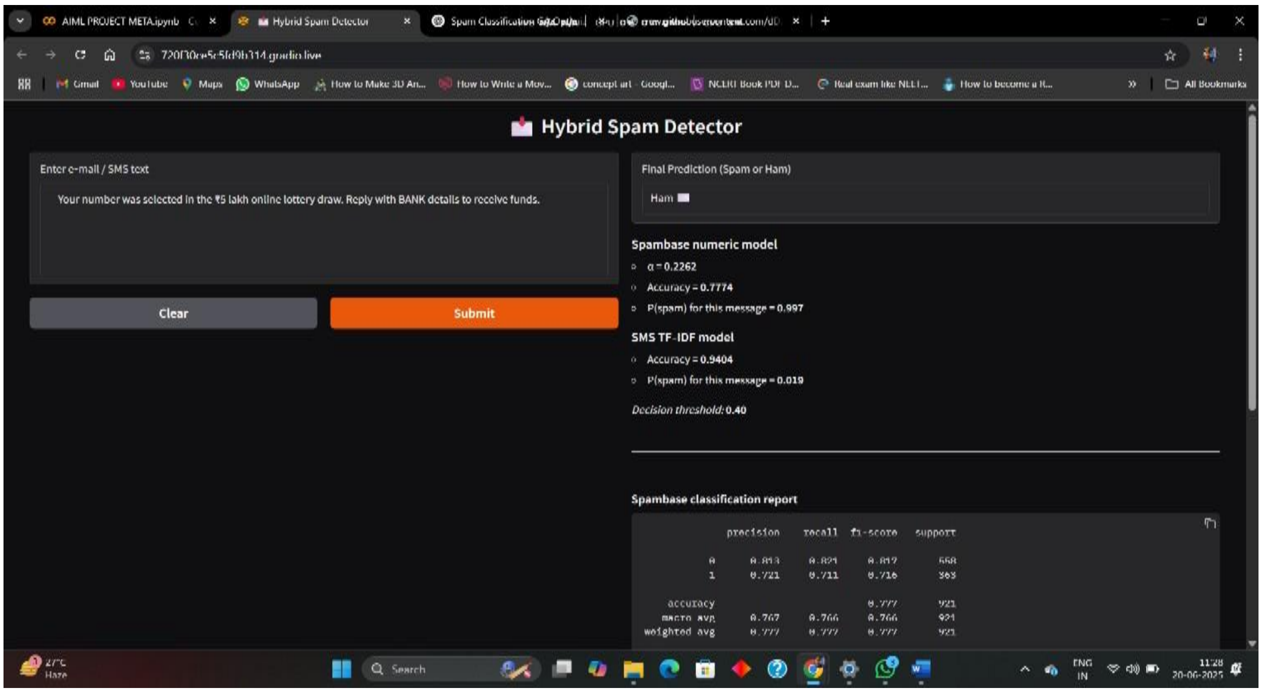
Decision threshold: **0.40**

Spambase classification report

```
              precision    recall  f1-score   support

           0      0.813     0.821     0.817       558
           1      0.721     0.711     0.716       363

    accuracy                          0.777       921
   macro avg      0.767     0.766     0.766       921
weighted avg      0.777     0.777     0.777       921
```

Figure 7.2.3: Test Case 2

Test Case 3:



Hybrid Spam Detector

Enter e-mail / SMS text

Your number was selected in the ₹5 lakh online lottery draw. Reply with BANK details to receive funds.

Final Prediction (Spam or Ham)

Ham

Spambase numeric model
- α = 0.2262
- Accuracy = 0.7774
- P(spam) for this message = 0.997

SMS TF-IDF model
- Accuracy = 0.9404
- P(spam) for this message = 0.019

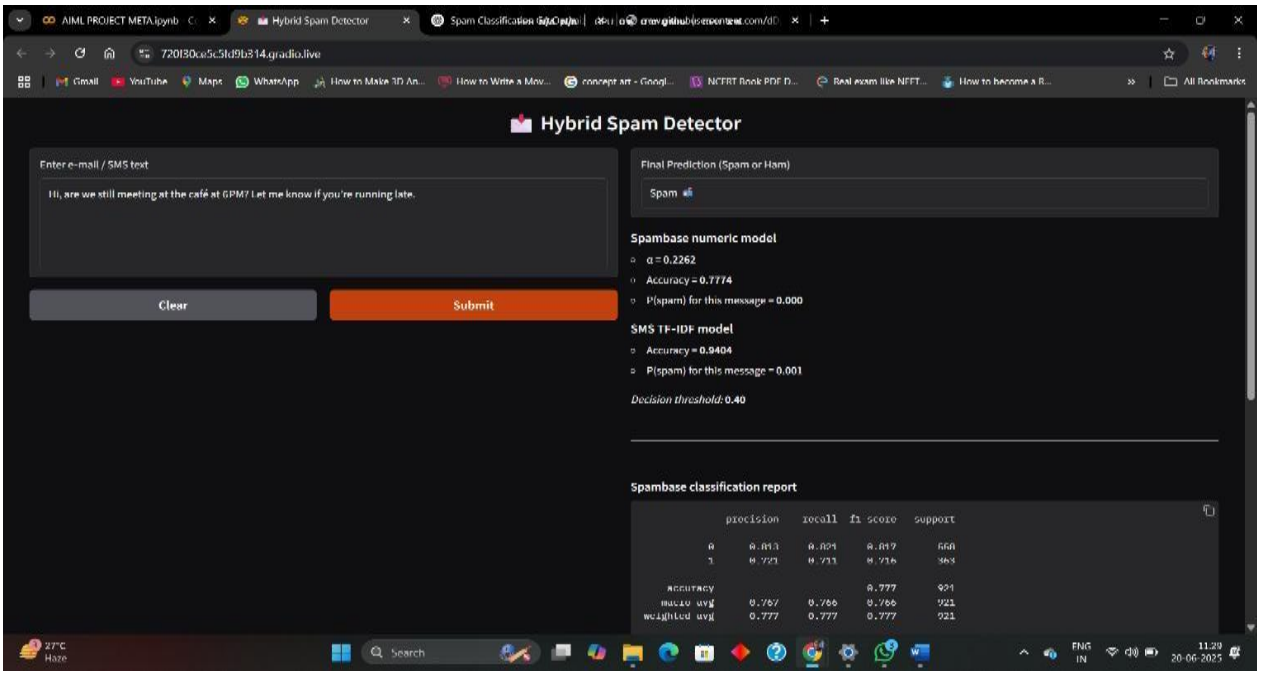
Decision threshold: 0.40

Spambase classification report

```
              precision    recall  f1-score   support

           0      0.813     0.821     0.817       558
           1      0.721     0.711     0.716       363

    accuracy                          0.777       921
   macro avg      0.767     0.766     0.766       921
weighted avg      0.777     0.777     0.777       921
```

Figure 7.2.4: Test Case 3

Test Case 4:



Hybrid Spam Detector

Enter e-mail / SMS text

Hi, are we still meeting at the café at 6PM? Let me know if you're running late.

Final Prediction (Spam or Ham)

Spam

Spambase numeric model
- α = 0.2262
- Accuracy = 0.7774
- P(spam) for this message = 0.000

SMS TF-IDF model
- Accuracy = 0.9404
- P(spam) for this message = 0.001

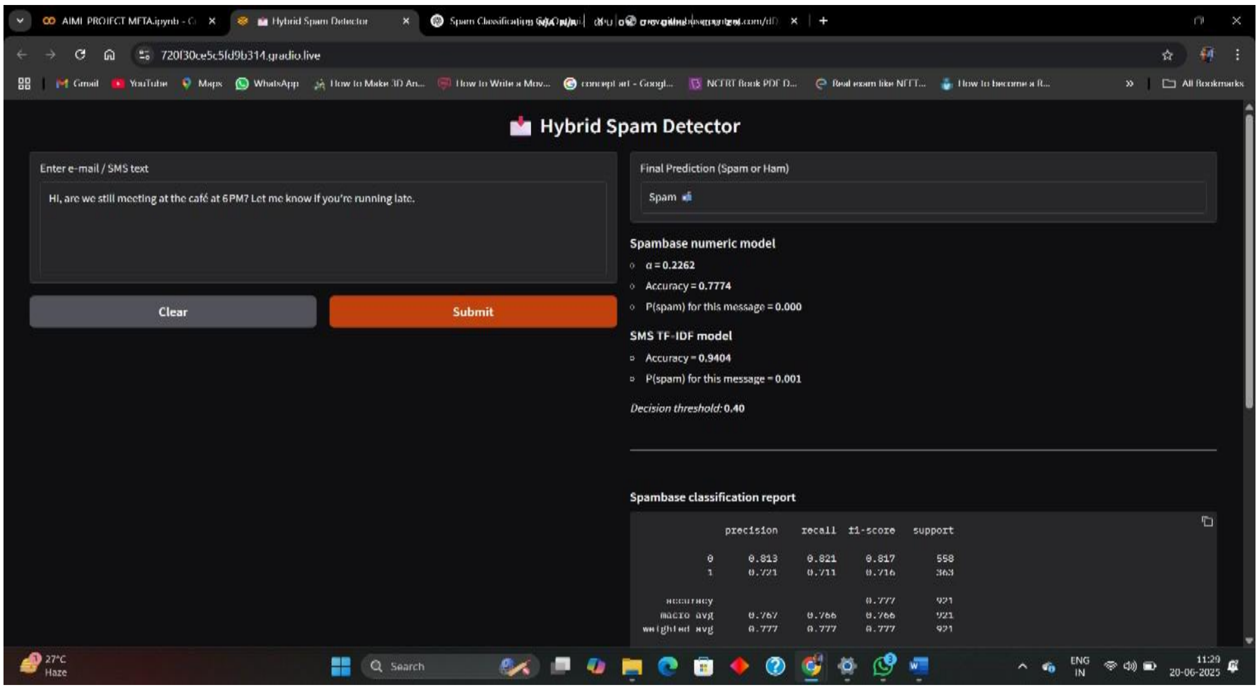
Decision threshold: 0.40

Spambase classification report

```
              precision    recall  f1 score   support

           0      0.813     0.821     0.817       558
           1      0.721     0.711     0.716       363

    accuracy                          0.777       921
   macro avg      0.767     0.766     0.766       921
weighted avg      0.777     0.777     0.777       921
```

Figure 7.2.5: Test Case

Test Case 5:



Hybrid Spam Detector

Enter e-mail / SMS text

Hi, are we still meeting at the café at 6PM? Let me know if you're running late.

Final Prediction (Spam or Ham)

Spam

Spambase numeric model

- α = **0.2262**
- Accuracy = **0.7774**
- P(spam) for this message = **0.000**

SMS TF-IDF model

- Accuracy = **0.9404**
- P(spam) for this message = **0.001**

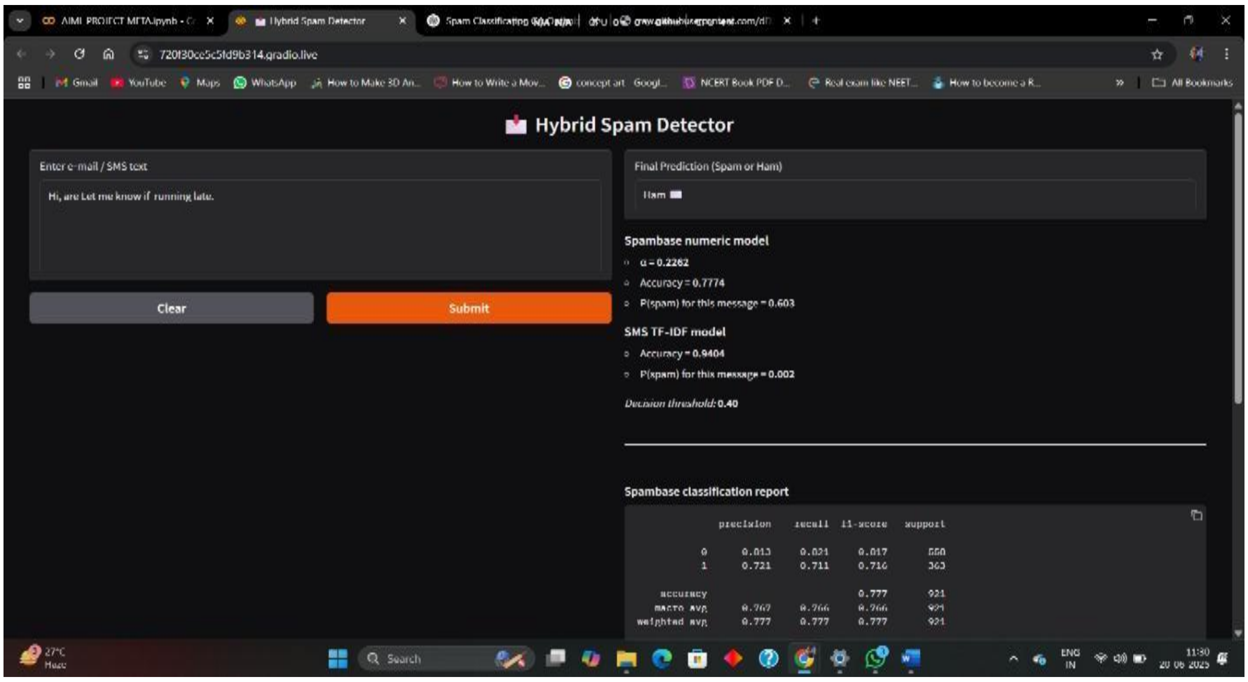
Decision threshold: **0.40**

Spambase classification report

```
              precision    recall  f1-score   support

           0      0.813     0.821     0.817       558
           1      0.721     0.711     0.716       363

    accuracy                          0.777       921
   macro avg      0.767     0.766     0.766       921
weighted avg      0.777     0.777     0.777       921
```

Figure 7.2.6: Test Case 5

Test Case 6:



Hybrid Spam Detector

Enter e-mail / SMS text

Hi, are Let me know if running late.

Final Prediction (Spam or Ham)

Ham

Spambase numeric model

- α = **0.2262**
- Accuracy = **0.7774**
- P(spam) for this message = **0.603**

SMS TF-IDF model

- Accuracy = **0.9404**
- P(spam) for this message = **0.002**

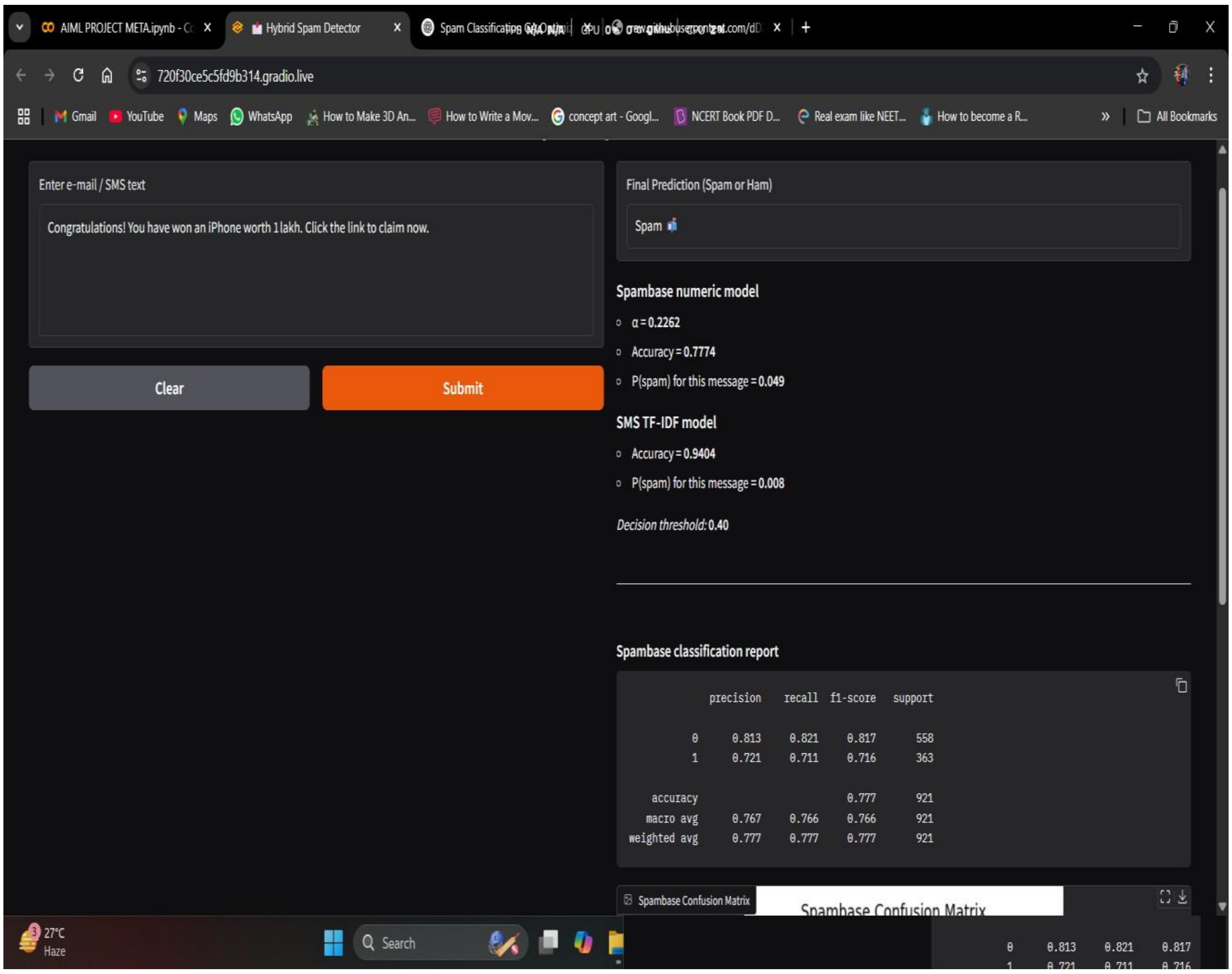
Decision threshold: **0.40**

Spambase classification report

```
              precision    recall  f1-score   support

           0      0.813     0.821     0.817       558
           1      0.721     0.711     0.716       363

    accuracy                          0.777       921
   macro avg      0.767     0.766     0.766       921
weighted avg      0.777     0.777     0.777       921
```

Figure 7.2.7: Test Case 6

8 OUTPUT SCREENS



Enter e-mail / SMS text

Congratulations! You have won an iPhone worth 1 lakh. Click the link to claim now.

Clear

Submit

Final Prediction (Spam or Ham)

Spam

Spambase numeric model

- α = **0.2262**
- Accuracy = **0.7774**
- P(spam) for this message = **0.049**

SMS TF-IDF model

- Accuracy = **0.9404**
- P(spam) for this message = **0.008**

Decision threshold: **0.40**

Spambase classification report

```
              precision    recall  f1-score   support

           0      0.813     0.821     0.817       558
           1      0.721     0.711     0.716       363

    accuracy                          0.777       921
   macro avg      0.767     0.766     0.766       921
weighted avg      0.777     0.777     0.777       921
```

Spambase Confusion Matrix

Figure 8.1: Output Screen

9 CONCLUSION

Spam email detection has become a critical component of modern digital communication systems, aimed at filtering unwanted or malicious content and improving user safety. Machine learning (ML) techniques have greatly enhanced the effectiveness of spam detection systems by learning from historical data to classify incoming emails accurately. Traditional ML models such as Naive Bayes, Decision Trees, Support Vector Machines, and Neural Networks have shown promising results in distinguishing spam from legitimate messages based on features like email content, sender information, and metadata.

However, the performance of these models heavily relies on optimal parameter selection and effective feature extraction. This is where metaheuristic algorithms such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Grey Wolf Optimizer (GWO) play a vital role. These algorithms can efficiently search large solution spaces to identify the most relevant features and fine-tune model parameters, leading to improved classification accuracy and reduced false positives.

The integration of ML with metaheuristic optimization not only boosts the overall efficiency of spam detection systems but also adapts better to evolving spam tactics. This hybrid approach ensures better generalization, adaptability, and robustness. Overall, combining machine learning with metaheuristic optimization provides a scalable and intelligent framework for combating spam emails in dynamic and high-volume environments.

10 FUTURE ENHANCEMENTS

As cyber threats evolve and spam tactics become more sophisticated, future enhancements to spam detection systems must focus on increasing adaptability, efficiency, and resilience. Some promising directions include:

- **Integration of Deep Learning Models**: Deep neural networks (DNNs), including LSTM and transformers like BERT, can capture contextual and sequential patterns in email content. Combining these with metaheuristic optimization for hyperparameter tuning could significantly boost detection rates.
- **Enhanced Feature Selection Techniques**: The performance of spam detection heavily depends on the quality of extracted features. Future systems could leverage hybrid metaheuristic approaches (e.g., combining GA with PSO) to discover the most discriminative features more efficiently and reduce dimensionality without sacrificing accuracy.
- **Real-Time Adaptive Systems**: Future spam filters could utilize online learning algorithms, optimized in real-time using swarm intelligence or evolutionary strategies, to adapt to new spam trends without requiring complete retraining.
- **Semantic Analysis Using NLP**: Employing advanced NLP techniques for semantic analysis can help understand the intent behind messages. Combining this with optimization algorithms can create intelligent systems that are resistant to adversarial tactics such as text obfuscation.
- **Privacy-Preserving Spam Detection**: With growing concerns over data privacy, federated learning models—optimized by decentralized metaheuristics—could be explored to train robust models across multiple users without sharing raw email data.
- **Improved Evaluation Metrics**: Future research could also focus on optimizing performance based on cost-sensitive metrics that balance false positives and negatives, which is crucial for real-world applications.
- **Explainable AI (XAI)**: Incorporating explainability into the spam detection pipeline would help users and administrators understand why certain emails are flagged as spam, improving trust in automated systems.

11 REFERENCES

1. **Simran Gibson et al. (2025)** Proposed integrating ML algorithms (Naïve Bayes, SVM, RF, DT, MLP) with Genetic Algorithm and PSO. MNB+GA achieved 100% accuracy on the SpamAssassin dataset using Scikit-learn.

2. **Emmanuel Gbenga Dada et al. (2024)** Reviewed spam filtering techniques using ML and DL. Highlighted challenges like spam evasion, data imbalance, and stressed the need for adversarial learning approaches.

3. **W.A. Awad & S.M. ELseuofi (2024)** Compared six ML algorithms; Naïve Bayes and Rough Sets performed best. Emphasized hybrid models and limitations of rule-based filters.

4. **Shahi & Singh (2023)** Developed a CNN + Word2Vec hybrid model for spam detection, achieving high precision and F1-score. Showed improved contextual understanding.

5. **Kumar & Bansal (2022)** Proposed a CNN-LSTM model for spam classification. The hybrid model handled unstructured text well and improved generalization across datasets.

6. **Moustafa & Slay (2021)** Introduced the TON_IoT dataset. Emphasized real-time spam detection in AI-driven cybersecurity systems with lightweight adaptive models.

7. **Abdullah & Arpaci (2020)** Used ensemble classifiers (Random Forest, AdaBoost) for spam filtering. Achieved higher accuracy and robustness than individual classifiers.

8. **Zhang et al. (2020)** Applied Chi-Square and Information Gain for feature selection. Enhanced ML model performance by reducing redundant features.

9. **Roy & Basu (2019)** Used TF-IDF with Decision Trees for spam classification. Noted issues with multilingual email handling and suggested language-specific preprocessing.

10. **Androutsopoulos et al. (2019)** Compared Naïve Bayes with keyword-based filters. ML models adapted better to evolving spam but had latency issues in real-time usage.

11. **Goodman et al. (2019)** Discussed the "spam arms race." Advocated for adaptive ML models that continuously learn from new spam patterns and attacker strategies.

12. **Zhang, Zhu & Yao (2018)** Proposed cost-sensitive learning to reduce false positives in corporate environments. Penalized misclassification more severely for legitimate mails.

13. **Carreras & Márquez (2018)** Applied AdaBoost to enhance spam classification accuracy. Boosted models showed significant improvements and reduced overfitting.

14. **Sahami et al. (2017)** Pioneered Naïve Bayes for spam filtering. Their work laid the foundation for probabilistic email classification in modern ML-based filters.

15. **Delany et al. (2017)** Proposed a hybrid spam filter using Case-Based Reasoning with ensemble methods. The model adapted to new spam patterns and improved classification accuracy over traditional filters.

16. **Cordeiro & de Carvalho (2017)** Used a multi-objective Genetic Algorithm to optimize spam filters by balancing accuracy, false positives, and false negatives. Achieved better results on imbalanced datasets.