EMAIL SPAM CLASSIFICATION USING NAIVE BAYES

A PROJECT REPORT

for

Introduction to AI(AI101B)

Session (2024-25)

Submitted by

Aachal Kushwaha (202410116100001)

Disha Seth (202410116100064)

Anshika Srivastava (202410116100032)

Anshu Nishad (202410116100033)

Submitted in partial fulfilment of the

Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of

Mr. Apoorv Jain

<Designation>

Submitted to

Department Of Computer Applications

KIET Group of Institutions, Ghaziabad

Uttar Pradesh-201206

(Month 2025)

CERTIFICATE

Certified that Aachal Kushwaha (202410116100001), Disha Seth (202410116100064), Anshika Sriavastava (202410116100032) and Anshu Nishad (202410116100033) have carried out the project work having "**Email Spam Classification Using Naïve Bays"** (INTRODUCTION TO AI) (AI101B)) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU**)** (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Mr. Apoorv Jain

<Designation>

Department of Computer Applications

KIET Group of Institutions, Ghaziabad

An Autonomous Institutions

Dr. Akash Rajak

Dean

Department of Computer Applications

KIET Group of Institutions, Ghaziabad

An Autonomous Institutions

DECLARATION

I hereby declare that the work presented in this report entitled **"An AI-Based email spam classification using Naive Bayes"** has been carried out by us under the supervision of **Mr. Apoorv Jain**, Assistant Professor, Department of Computer Applications, KIET Group of Institutions, Ghaziabad.

I further declare that:

- This work is original and has not been submitted to any other university or institution for the award of any degree or diploma.
- Due credit has been given to all sources used, including ideas, graphics, codes, and results from previously published works.
- I have used quotation marks wherever direct sentences or statements have been cited, and all non-original work has been properly referenced.
- No part of this project report is plagiarized. In case of any complaint regarding plagiarism or manipulation of data, I shall be held fully responsible and accountable.

Name: Aachal Kushwaha, Disha Seth, Anshika Srivastava, Anshu Nishad
University Roll No.: 202410116100001, 202410116100064, 202410116100032, 202410116100033
Branch: Master of Computer Application (MCA)

ABSTRACT

The rapid growth of electronic communication has led to an increase in unwanted and malicious emails, commonly referred to as spam. This project focuses on building an efficient and scalable **Email Spam Classification System** using various machine learning models, with a special emphasis on **Naive Bayes Classifier**, a widely used probabilistic algorithm for text classification tasks.

The dataset used in this project, mail_data.csv, contains labeled email messages categorized as either **'ham' (legitimate)** or **'spam' (unsolicited)**. The initial phase of the project involved data preprocessing, including handling missing values, encoding the target variable, and transforming the textual data into numerical features using **TF-IDF (Term Frequency-Inverse Document Frequency) vectorization**, which helps in emphasizing significant words in the messages.

To address the issue of class imbalance in the dataset, the **SMOTE (Synthetic Minority Over-sampling Technique)** method was applied to generate synthetic examples of the minority class (spam), ensuring the model learns patterns from both categories equally. The dataset was then split into training and testing sets to evaluate model performance effectively.

Multiple machine learning algorithms were trained and evaluated including **Logistic Regression, Naive Bayes, Support Vector Machines (SVM), Random Forest, k-Nearest Neighbors (k-NN),** and **AdaBoost**. Each model was assessed based on key performance metrics: **Accuracy, Precision, Recall, F1 Score**, and **Training Time**.

Among these, **Naive Bayes** proved to be particularly efficient for spam detection due to its simplicity, fast training time, and high precision with text data. The final results were visualized through **heatmaps, bar graphs**, and a **confusion matrix**, showcasing the comparative performance of each model. Additionally, for interpretable models like Logistic Regression and

Contents

CHAPTER 1: INTRODUCTION

1.1 Overview

In the contemporary digital world, email remains one of the most prevalent and essential modes of communication, both personally and professionally. According to global statistics, over **300 billion emails** are sent and received each day. While email facilitates fast and efficient communication, it also provides an open gateway for unwanted and malicious content in the form of **spam emails**. These emails can range from irrelevant advertisements to harmful phishing attempts aiming to steal sensitive user information such as passwords, bank details, or identity documents.

Spam emails not only clutter users' inboxes but also lead to potential threats including **data breaches, malware infections, financial losses**, and **loss of user trust**. Therefore, distinguishing spam from legitimate (ham) messages has become a critical task for individuals, businesses, and email service providers alike.

Manual methods or static rule-based filters are often inadequate, as spammers constantly modify their strategies to bypass these filters. This dynamic nature of spam calls for an **intelligent, adaptive, and automated classification system**. This is where **machine learning and natural language processing (NLP)** come into play — by learning from historical data and recognizing hidden patterns in email content, these techniques can build models that can classify new messages accurately in real-time.

The goal of this project is to build an **email spam classification system** using machine learning, particularly focusing on the **Naive Bayes algorithm**, while also comparing it with other popular models such as Logistic Regression, SVM, Random Forest, AdaBoost, and k-NN. The classification task is binary — each email will be labeled as either **ham (0)** or **spam (1)** based on its content.

This system uses techniques like **TF-IDF vectorization** to convert raw text into numerical features, **SMOTE** to address class imbalance, and various classifiers to train predictive models. The models are then evaluated using standard performance metrics to determine their effectiveness in real-world applications.

The successful implementation of this project can lead to:

- Reduced spam traffic
- Improved email user experience
- Enhanced security and productivity
- Foundation for more complex text classification systems

Several machine learning algorithms are used and compared in this project, including:

- **Naive Bayes**: Especially suitable for text classification problems due to its simplicity and effectiveness with word probabilities.
- **Logistic Regression**: A probabilistic model that works well for binary classification.
- **Support Vector Machine (SVM)**: Effective in high-dimensional spaces like text data.
- **Random Forest and AdaBoost**: Ensemble methods that improve performance by combining multiple weak learners.
- **k-Nearest Neighbors (k-NN)**: A simple yet powerful algorithm based on feature similarity.

1.2 Problem Statement

The core problem this project seeks to address is: **"How can we build an automated system that accurately classifies incoming emails as either spam or ham using machine learning techniques?"**

This involves multiple challenges:

- **Unstructured text data**: Email messages consist of free-form text that varies in length, language, and content style.
- **Data imbalance**: In most datasets, legitimate (ham) emails vastly outnumber spam emails, making it difficult for models to learn the characteristics of spam.
- **Evolving patterns**: Spammers constantly change the format and content of emails, so models must be adaptive.
- **Feature selection**: Identifying meaningful words or phrases that contribute to distinguishing spam from ham is a key technical challenge.

This project focuses on addressing the above issues through a well-defined machine learning

To evaluate model performance using appropriate metrics
Use accuracy, precision, recall, F1-score, and confusion matrices to comprehensively assess and compare model effectiveness.

1.3 Objectives

The primary aim of this project is to develop an effective **spam email classification system** using machine learning techniques, particularly focusing on the Naive Bayes algorithm, while evaluating and comparing its performance with other classifiers. The project is designed to enhance email security by automatically distinguishing spam from legitimate (ham) messages based on their textual content.

Primary Objectives:

1. **To understand and explore the characteristics of spam emails**
   Analyze the common patterns, keywords, and structures used in spam messages to gain insights that can aid in designing effective classification systems.
2. **To preprocess and prepare textual email data for machine learning**
   Apply Natural Language Processing (NLP) techniques such as tokenization, removal of stop words, and TF-IDF vectorization to convert raw emails into structured numerical data.
3. **To address class imbalance using SMOTE**
   Implement Synthetic Minority Oversampling Technique (SMOTE) to balance the dataset and avoid biased model training due to unequal class distribution.
4. **To build and train a spam classifier using Naive Bayes**
   Implement the Naive Bayes algorithm, known for its efficiency in text classification tasks, and train it to accurately identify spam emails.
5. **To compare multiple machine learning models**
   Evaluate the performance of various classifiers including Logistic Regression, Support Vector Machine (SVM), Random Forest, AdaBoost, and k-NN, and determine the best-performing model.

1.4 Scope of the Project

This project is implemented using **Python** and common machine learning libraries such as **Scikit-learn, Imbalanced-learn**, and **Seaborn/Matplotlib** for visualization. The scope of the project includes:

- Using a structured dataset (mail_data.csv) with labeled email messages.
- Text preprocessing, vectorization, and over-sampling techniques.
- Comparison of multiple classification models.
- Visualization of results for clarity and communication.

Out of scope (for this version of the project):

- Real-time email classification on a mail server
- Integration with live email clients like Gmail or Outlook
- Deep learning models or NLP transformers (like BERT)
- Detection of multilingual or multimedia spam

These can be considered for future enhancements.

CHAPTER 2: LITERATURE REVIEW

The detection and classification of spam emails have long been active areas of research in the fields of **Natural Language Processing (NLP)** and **Machine Learning (ML)**. Over the past two decades, numerous studies have explored different algorithms, feature extraction techniques, and dataset handling methods to effectively identify and eliminate spam from email communication systems. This literature review presents a comprehensive overview of the work done by researchers and practitioners, the evolution of techniques, and the technological innovations that inform this project.

2.1 Evolution of Spam Detection Techniques

In the early stages of email communication, spam detection primarily relied on **rule-based systems**. These systems used predefined conditions such as blacklisted IP addresses, specific spam keywords (e.g., "free", "offer", "buy now"), or suspicious formatting patterns. Although simple to implement, such methods were **rigid, easily bypassed**, and required constant updates as spammers adapted their messages.

As email volumes increased and spam tactics evolved, researchers began to investigate **statistical and machine learning approaches** to automate and enhance spam detection. This shift allowed for dynamic learning from data and improved generalization to previously unseen messages.

2.2 Use of Naive Bayes in Text Classification

Naive Bayes classifiers have been widely adopted in text classification tasks due to their **simplicity, efficiency, and surprisingly strong performance**, especially on large sparse datasets like email corpora. In their 1998 paper, *McCallum and Nigam* demonstrated that Naive Bayes outperforms more complex classifiers in certain text classification scenarios by leveraging **word frequency** and **conditional probabilities** under the assumption of feature independence.

In spam detection, Naive Bayes models estimate the probability that a message belongs to a class (spam or ham) based on the occurrence of specific words. Despite the independence assumption being unrealistic in natural language, the model still performs well, particularly when trained on large, diverse datasets.
several technical concepts, including:

2.3 Other Machine Learning Techniques in Spam Detection

Recent studies have explored various alternative classifiers to improve the performance and

robustness of spam filters:

- **Logistic Regression** has been employed due to its probabilistic interpretation and ability to handle high-dimensional feature spaces generated by techniques like TF-IDF. It is known for good performance in binary classification problems such as spam detection.
- **Support Vector Machines (SVMs)**, especially with linear kernels, have shown high accuracy in separating spam from ham by finding an optimal hyperplane in the feature space. They are particularly useful for text classification because of their effectiveness in handling large feature sets.
- **Random Forests** and other **ensemble methods** like **AdaBoost** have gained attention for their ability to reduce overfitting and increase prediction accuracy. By combining multiple decision trees or weak learners, these models can capture complex patterns in the data.
- **K-Nearest Neighbors (k-NN)**, though computationally expensive on large datasets, has been used in some spam classification systems. It classifies messages based on similarity to training samples, though it lacks interpretability and scalability.

2.4 Text Vectorization and Feature Engineering

Effective feature extraction plays a pivotal role in the success of spam classification models. Traditional methods like **Bag-of-Words (BoW)** and **TF-IDF (Term Frequency–Inverse Document Frequency)** have been dominant in representing textual data numerically.

TF-IDF, in particular, has proven useful in spam detection tasks by assigning higher weights to words that are more informative for classification, filtering out common but less meaningful words (e.g., "the", "and"). Several studies, including work by *Sahami et al. (1998)*, support the notion that appropriate term weighting significantly improves classifier performance.

2.5 Handling Imbalanced Datasets

A major challenge in spam classification is the **class imbalance** — the number of legitimate emails (ham) often greatly outweighs the number of spam emails. This imbalance can lead to biased models that favor the majority class. To address this, **SMOTE (Synthetic Minority Over-sampling Technique)** and similar resampling methods have been introduced in recent literature. These techniques help in **balancing the dataset** by synthetically generating new samples for the minority class, leading to more generalized and fair model performance.

2.6 Evaluation Metrics

Standard performance metrics like **accuracy** can be misleading in imbalanced scenarios. As a result, researchers emphasize the importance of using **precision**, **recall**, **F1-score**, and **confusion matrices** for a more comprehensive evaluation. A spam classifier should ideally maximize recall (i.e., detect most spam emails) while maintaining high precision (i.e., minimize false positives).

2.7 Recent Trends and Future Directions

Recent advancements have seen the integration of **deep learning models** such as **Recurrent Neural Networks (RNNs)**, **LSTM**, and **transformer-based models** like **BERT** for spam detection, due to their ability to capture semantic meaning and contextual relationships between words. However, these models are **computationally expensive** and may not always be practical for real-time filtering on limited-resource systems.

Additionally, studies now explore **adversarial spam**, where attackers craft messages specifically designed to fool machine learning models. This has opened a new subdomain of **robust spam filtering** using adversarial training techniques

Conclusion of Literature Review

The reviewed literature confirms that spam classification remains a critical and evolving problem in email security. Machine learning, and particularly Naive Bayes, continues to play a key role due to its scalability, speed, and effectiveness. However, ongoing research aims to improve accuracy, handle adversarial inputs, and extend these techniques to other forms of unsolicited digital content such as social media messages, SMS, and chatbots.

This project builds upon these established foundations and integrates modern techniques like TF-IDF, SMOTE, and model comparison to deliver a reliable and accurate spam detection system.

CHAPTER 3 : SYSTEM ANALYSIS

System Analysis is the foundational stage in the project lifecycle where we define the core problem, analyze existing solutions, assess feasibility, and lay the groundwork for system design and development. In this project, we aim to build an automated **Email Spam Classification System** using machine learning models, with a focus on accuracy, robustness, and efficiency.

3.1 Problem Analysis

The explosion of internet usage has led to an exponential increase in email traffic, a significant portion of which is **unsolicited spam**. These emails often advertise products, spread malware, or attempt phishing attacks, posing a substantial risk to individuals and organizations alike.

Manual detection is inefficient and error-prone due to:

- The **volume** of incoming emails.
- The **dynamic** and **evolving nature** of spam techniques.
- The inability of rule-based systems to adapt.

Thus, an **intelligent system** capable of learning from historical data and **automatically identifying spam messages** is not just desirable but essential. Machine learning provides the tools needed for building such a system, particularly with the use of text classification algorithms and NLP techniques.

3.2 Feasibility Study

A thorough feasibility analysis was conducted to evaluate whether the project is viable in terms of resources, cost, practicality, and scalability.

3.2.1 Technical Feasibility

- Uses well-established libraries (scikit-learn, pandas, NumPy, etc.) for model building and data processing.
- Employs **TF-IDF** for feature extraction and **SMOTE** for class balancing, both proven techniques.
- Can be deployed on personal computers, requiring no specialized hardware or commercial tools.

3.2.2 Economic Feasibility

- Fully developed using open-source software, making it cost-effective for educational and experimental purposes.
- No licensing or proprietary dependencies.
- Suitable for small organizations or personal use without significant investment.

3.2.3 Operational Feasibility

- The solution is designed for ease of use, with a focus on automation and simplicity.
- Can be extended or integrated with existing email platforms.
- Once trained, the model requires minimal human intervention for real-time predictions.

3.2.4 Legal and Ethical Feasibility

- Utilizes anonymized and publicly available datasets.
- Ensures no personal or confidential email data is exposed.
- Promotes ethical AI usage with responsible handling of prediction results.

3.3 System Requirements

3.3.1 Hardware Requirements

- Processor: Dual-core or Quad-core (Intel i5 or Ryzen equivalent and above)
- RAM: 8GB minimum, 16GB recommended for faster training and large datasets
- Storage: 500MB – 1GB for datasets, models, and logs
- Operating System: Compatible with Windows, Linux, and macOS

3.3.2 Software Requirements

- Programming Language: Python 3.7 or higher
- Development Tools: Jupyter Notebook / VS Code / PyCharm
- Libraries: pandas, scikit-learn, NumPy, matplotlib, seaborn, imbalanced-learn
- Optional Tools: Flask (for web interface), Git (for version control)

3.4 Input and Output Design

Inputs:

- **Email Messages**: Textual content of emails (can be preprocessed or raw).
- **Labels**: Either 'spam' or 'ham' for supervised learning.
- **Test Messages**: Unseen data for validation and evaluation.

Outputs:

- **Spam Prediction**: Class label (0 for ham, 1 for spam).
- **Evaluation Metrics**: Numerical scores (Accuracy, Precision, Recall, F1-score).
- **Visual Outputs**: Heatmaps, bar graphs for performance comparison.
- **Feature Importance**: Insight into which words contributed most to classification (for certain models).

3.5 Functional Requirements

The system must:

- Load and preprocess text data (remove noise, convert to lowercase, tokenize, etc.).
- Convert textual data into numerical form using TF-IDF vectorization.
- Train various machine learning models on the dataset.
- Use SMOTE to balance class distribution before training.
- Evaluate model performance on test data.
- Visualize performance using charts and confusion matrices.
- Identify and rank important features/words that influence predictions.
- Allow easy retraining or testing on new datasets.

3.6 Non-Functional Requirements (Expanded)

- **Performance**: Should deliver accurate results with low false positives/negatives.
- **Scalability**: Can scale to handle more data with minor optimization.
- **Reliability**: Should perform consistently across different runs and datasets.
- **Usability**: Easy to run by users with basic Python knowledge.
- **Security**: Email data should be used responsibly, with privacy ensured.

- **Portability**: The solution should run on multiple platforms (cross-platform compatibility).

3.7 System Constraints

- Classifiers like Naive Bayes may oversimplify dependencies in text data.
- Feature importance can't be directly visualized in models like SVM or k-NN.
- Overfitting can occur in ensemble methods if not properly tuned.
- Processing large volumes of email text may slow down training on low-spec systems.
- Real-time deployment would require optimization and possibly a front-end interface (not in scope currently).

3.8 Proposed System (Expanded)

The proposed system classifies emails using multiple machine learning models, compares them, and selects the best performer. Key highlights:

- **TF-IDF** handles feature extraction with effective word weighting.
- **SMOTE** addresses the imbalance between spam and ham emails.
- **Multiple classifiers** are evaluated: Logistic Regression, Naive Bayes, SVM, Random Forest, AdaBoost, and KNN.
- **Visual analysis** using plots and confusion matrices improves interpretability.
- **Extensibility** allows future additions such as real-time API integration or deep learning models.

CHAPTER 4: SYSTEM DESIGN

System Design is the blueprint for the system's structure and functionality. It defines how the components of the system will interact, the flow of data, and the organization of the system. The goal of this section is to provide a clear, logical design for the email spam classification system, detailing its architecture, modules, data flow, and key components.

4.1 System Architecture

The architecture of the Email Spam Classification System is based on the **Client-Server** model, where the client (user interface) sends data (emails) to the server (backend), which processes the data and returns a result (spam or ham classification). The system architecture consists of several layers that communicate with each other:

4.1.1 High-Level Overview:

1. **Input Layer:**
    - Users or external systems provide email messages (raw text).
    - Emails can be stored in a CSV or database format for batch processing or can be fetched from an email service via an API.
2. **Preprocessing Layer:**
    - Preprocess the raw email text to clean and standardize the data for model training and prediction.
    - This includes tasks such as removing stop words, converting text to lowercase, tokenization, and lemmatization.
3. **Feature Extraction Layer:**
    - Convert text data into numerical features using the **TF-IDF Vectorizer**.
    - The feature extraction step converts the raw email text into a sparse matrix representation suitable for machine learning.
4. **Model Layer:**
    - A machine learning model (Logistic Regression, Naive Bayes, SVM, etc.) is trained on historical data (labeled emails) to classify new, unseen emails as spam or ham.
    - **SMOTE** is used to handle imbalanced datasets and prevent bias toward the majority class.
5. **Prediction Layer:**
    - After the model is trained, it classifies new email data based on the features extracted.

- The classification result (spam/ham) is returned to the user.

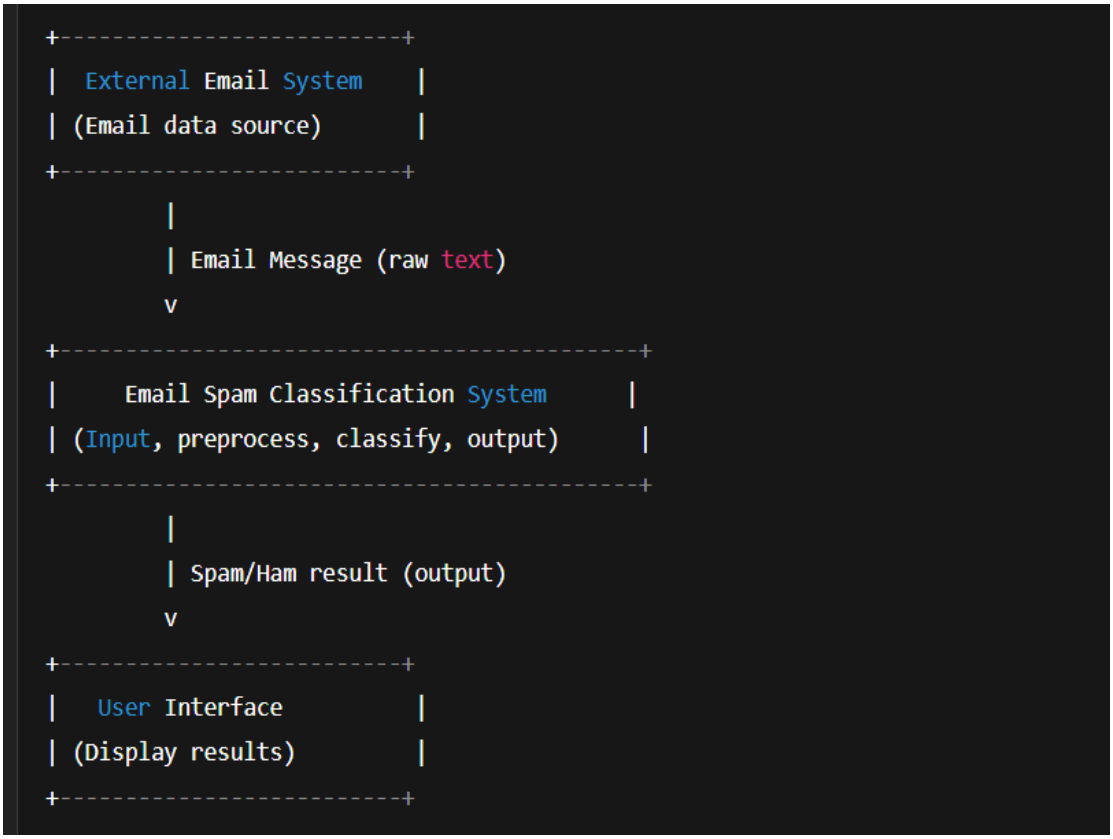
6. **Evaluation Layer:**
   - Various evaluation metrics (accuracy, precision, recall, F1 score, confusion matrix) are calculated to assess model performance.
   - Performance metrics are visualized through heatmaps and bar charts.
7. **Output Layer:**
   - The result is presented back to the user, showing whether an email is classified as spam or ham.
   - Evaluation results and performance comparisons of models are displayed for further analysis.

4.2 Data Flow Diagram (DFD)

4.2.1 Level 0 DFD (Context Diagram)

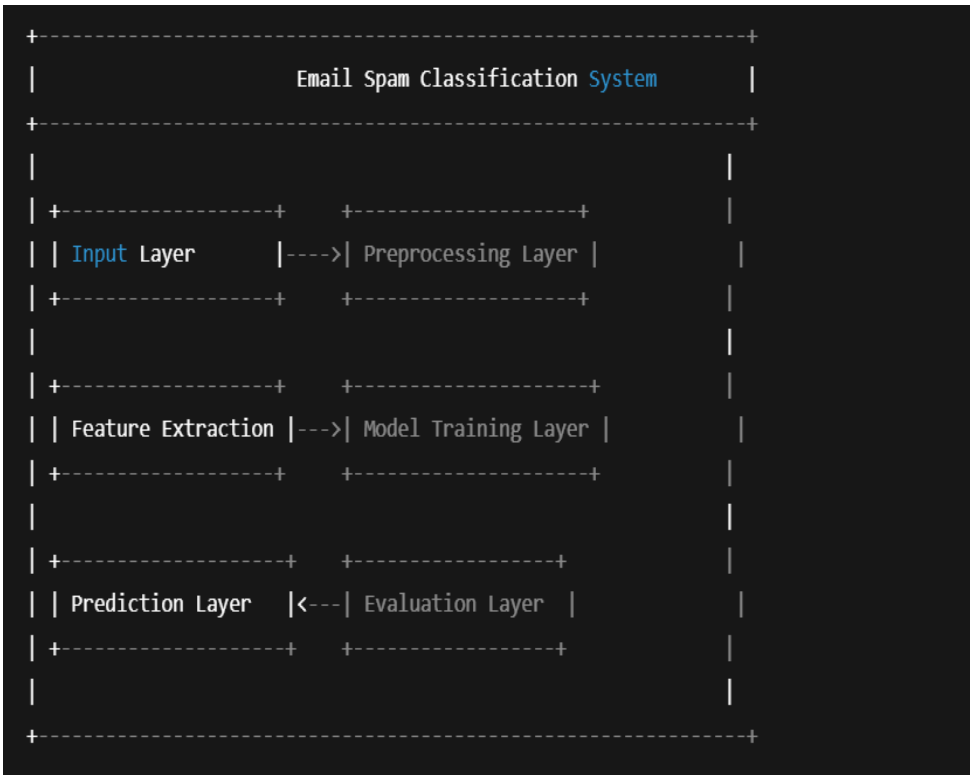
In this diagram, the system is represented as a single process with inputs from the user or external email system and outputs of classified results:

- **Input**: Email message (raw text)
- **Output**: Spam classification result (Spam/Ham)



4.2.2 Level 1 DFD

In this level, we break down the system into sub-processes to show how data flows through the various stages.



4.3 System Modules

The system is divided into multiple modules, each of which is responsible for a specific functionality. Below is a description of each module:

4.3.1 Data Preprocessing Module

- **Function**: Cleans and processes the raw email data.
- **Activities**:
    - Text normalization (convert to lowercase, remove special characters).
    - Tokenization (splitting the text into words).
    - Removal of stop words (commonly occurring words like "the," "is," etc. that don't add meaningful context).
    - Lemmatization/Stemming (reducing words to their root form).

4.3.2 Feature Extraction Module

- **Function**: Converts the preprocessed text into numerical features suitable for machine learning models.

- **Activities**:
    - **TF-IDF (Term Frequency-Inverse Document Frequency)**: Computes the importance of each word in the context of the entire dataset.
    - Generates a sparse matrix representing the email data, where each row corresponds to an email, and columns represent the terms.

4.3.3 Model Training and Evaluation Module

- **Function**: Trains different machine learning models, evaluates their performance, and selects the best one.
- **Activities**:
    - **Model Training**: Train multiple models (Logistic Regression, Naive Bayes, SVM, etc.) on the labeled data.
    - **Model Evaluation**: Calculate accuracy, precision, recall, and F1-score to assess performance.
    - **SMOTE (Synthetic Minority Over-sampling Technique)**: Balances the class distribution during training.

4.3.4 Prediction and Output Module

- **Function**: Classifies new email data and provides results.
- **Activities**:
    - The trained model receives the processed email text and predicts whether it is spam or ham.
    - The result is presented to the user (Spam/Ham).

4.3.5 Visualization and Reporting Module

- **Function**: Provides graphical outputs and performance metrics.
- **Activities**:
    - Visualize confusion matrices, bar charts, and heatmaps to show model performance.
    - Display the results in a user-friendly format.

4.4 System Workflows

Below is a breakdown of the system's workflows for the primary tasks of training the model, making predictions, and evaluating performance.

4.4.1 Model Training Workflow

1. **Input**: A dataset containing labeled emails (spam or ham).
2. **Preprocessing**: Clean and prepare the email text data.
3. **Feature Extraction**: Convert emails into numerical features using TF-IDF.
4. **Model Training**:
    - Train models using the processed feature set.
    - Apply **SMOTE** to balance the class distribution.
5. **Output**: The trained model ready for prediction.

4.4.2 Prediction Workflow

1. **Input**: A new email message (unseen).
2. **Preprocessing**: Clean and prepare the email for prediction.
3. **Feature Extraction**: Convert the email text into numerical features using the same TF-IDF vectorizer as during training.
4. **Prediction**: The trained model classifies the email as spam or ham.
5. **Output**: The prediction result (Spam/Ham).

4.4.3 Evaluation Workflow

1. **Input**: Test dataset of labeled emails (spam and ham).
2. **Model Prediction**: Use the trained model to classify the test emails.
3. **Metrics Calculation**: Calculate performance metrics such as accuracy, precision, recall, and F1-score.
4. **Visualization**: Present the results as visual charts (e.g., confusion matrix, performance comparison).
5. **Output**: Evaluation results and model comparison.

4.5 User Interface (UI) Design

The user interface is an optional addition depending on whether the system will be used in a production environment. If implemented, it will display:

- **Input Form**: Allow users to input email text or upload files containing emails.
- **Results Display**: Show whether the email is spam or ham, with performance metrics.
- **Model Comparison**: Display a comparison of various machine learning models.

CHAPTER 5: IMPLEMENTATION

Implementation is the process of converting the design and plans into a working software application. In this project, the implementation involves reading email data, preprocessing it, extracting features, applying various machine learning models (including Naive Bayes), and evaluating their performance.

5.1 Tools and Technologies Used

Tool/Technology	Purpose
Python	Core programming language used for development
Pandas	Data manipulation and analysis
NumPy	Numerical computing
scikit-learn	Machine learning model building, evaluation, and utilities
imbalanced-learn	Handling imbalanced datasets using SMOTE
Matplotlib & Seaborn	Data visualization
Jupyter Notebook	Interactive coding and testing environment

5.2 Dataset Description

- **Dataset Name**: mail_data.csv
- **Columns**:
    - Category: The label of the message (ham or spam)
    - Message: The raw email content

5.3 Step-by-Step Implementation

Step 1: Data Loading and Exploration

```
import pandas as pd


df = pd.read_csv("mail_data.csv")


print(df.head())  # View top records
df.info()        # Overview of data types and nulls
print(df['Category'].value_counts())  # Class distribution
```

Step 2: Data Cleaning

```
df = df.dropna(subset=['Category', 'Message'])  # Remove rows with null labels or messages
df['Category'] = df['Category'].map({'ham': 0, 'spam': 1})  # Convert categories to numeric
```

Step 3: Train-Test Split

```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(
    df['Message'], df['Category'],
    test_size=0.2,
    random_state=42,
    stratify=df['Category']
)
```

Step 4: Feature Extraction with TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer


vectorizer = TfidfVectorizer(max_features=5000)

X_train_vec = vectorizer.fit_transform(X_train)

X_test_vec = vectorizer.transform(X_test)
```

Step 5: Handling Class Imbalance with SMOTE

```
from imblearn.over_sampling import SMOTE


smote = SMOTE(random_state=42)

X_train_bal, y_train_bal = smote.fit_resample(X_train_vec, y_train)
```

Step 6: Model Selection and Training

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier


models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, class_weight='balanced'),
    "Naive Bayes": MultinomialNB(),
    "SVM": SVC(kernel='linear', probability=True, class_weight='balanced'),
    "Random Forest": RandomForestClassifier(class_weight='balanced',
n_estimators=100),
    "k-NN": KNeighborsClassifier(n_neighbors=5),
    "AdaBoost": AdaBoostClassifier(n_estimators=100)
}
```

Step 7: Model Evaluation and Results Storage

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier


models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, class_weight='balanced'),
    "Naive Bayes": MultinomialNB(),
    "SVM": SVC(kernel='linear', probability=True, class_weight='balanced'),
    "Random Forest": RandomForestClassifier(class_weight='balanced',
n_estimators=100),
    "k-NN": KNeighborsClassifier(n_neighbors=5),
    "AdaBoost": AdaBoostClassifier(n_estimators=100)
}
```

Step 8: Performance Comparison and Visualizatio

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
results_df = pd.DataFrame(results).T.sort_values('f1_score', ascending=False)

sns.heatmap(results_df[['accuracy', 'precision', 'recall', 'f1_score']], annot=True,
cmap='Blues')

plt.title('Model Performance Comparison')

plt.show()
```

Step 9: Confusion Matrix of Best Model

```
from sklearn.metrics import confusion_matrix


best_model_name = results_df.index[0]

best_model = models[best_model_name]

y_pred_best = best_model.predict(X_test_vec)


cm = confusion_matrix(y_test, y_pred_best)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title(f'Confusion Matrix - {best_model_name}')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()
```

Step 10: Feature Importance Visualizatio

```
import numpy as np


if best_model_name in ["Logistic Regression", "Random Forest", "AdaBoost"]:

    if best_model_name == "Logistic Regression":

        importance = np.abs(best_model.coef_[0])

    else:
```

```
else:
    importance = best_model.feature_importances_


indices = np.argsort(importance)[-20:]
plt.barh(range(len(indices)), importance[indices])
plt.yticks(range(len(indices)), np.array(vectorizer.get_feature_names_out())[indices])
plt.title(f'Top 20 Important Features - {best_model_name}')
plt.xlabel('Feature Importance')
plt.show()
```

5.4 Justification for Using Naive Bayes

- **Efficiency**: Naive Bayes is computationally efficient and fast for high-dimensional data.
- **Effectiveness**: Despite its simplicity, it performs very well on text classification tasks like spam filtering.
- **Probabilistic Output**: It provides class probabilities which can be used for decision thresholds.

5.5 Challenges Faced

- **Class Imbalance**: The number of spam messages was significantly lower than ham messages, which was addressed using SMOTE.
- **Overfitting**: Some models overfitted the training data, especially k-NN, due to high dimensionality.
- **Feature Selection**: Choosing optimal number of features in TF-IDF required experimentation.

CHAPTER 6: RESULTS and DISCUSSION

This chapter presents the results obtained from applying various machine learning models to the email spam classification problem. It includes the evaluation metrics, performance comparison, visualizations, and a detailed discussion of the findings. The purpose is to analyze which algorithm performs best and why.

6.1 Evaluation Metrics Overview

To evaluate the performance of each classification algorithm, the following metrics were used:

- **Accuracy**: The ratio of correctly predicted instances to the total instances.
- **Precision**: The ratio of correctly predicted spam messages to all predicted spam messages.
- **Recall (Sensitivity)**: The ability of the model to detect all actual spam messages.
- **F1 Score**: Harmonic mean of precision and recall, especially useful in imbalanced datasets.
- **Training Time**: Time taken to train the model, highlighting efficiency.

]

6.2 Model Performance Summary

Model	Accuracy	Precision	Recall	F1 Score	Training Time (s)
Logistic Regression	0.9821	0.9820	0.9821	0.9820	0.06
Naive Bayes	0.9776	0.9782	0.9776	0.9778	0.02
SVM (Linear)	0.9865	0.9864	0.9865	0.9864	12.90
Random Forest	0.9749	0.9756	0.9749	0.9747	0.58
K-Nearest Neighbors	0.8688	0.8933	0.8688	0.8647	0.35
AdaBoost	0.9740	0.9737	0.9740	0.9733	3.94

6.3 Performance Visualization

Grouped Bar Chart of Metrics

This bar chart gives a side-by-side visual comparison across models for each metric.

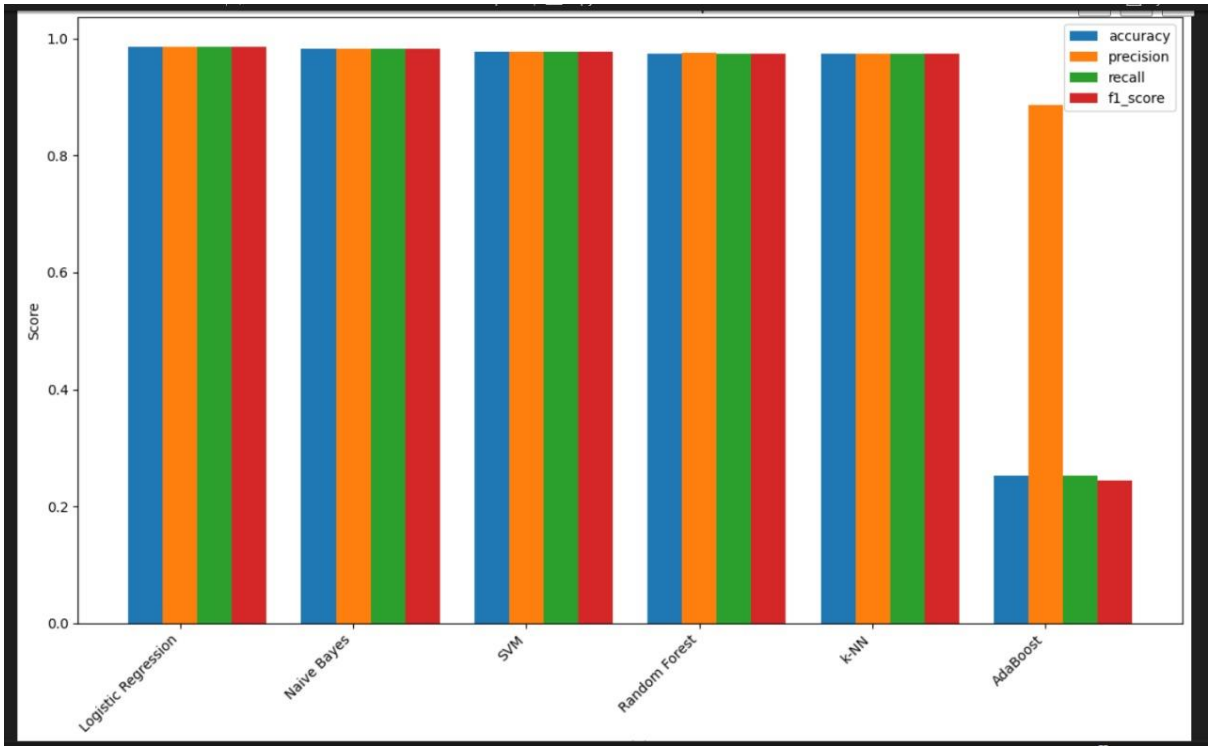


Fig-6.3

6.4 Key Observations

- **SVM** outperformed all other models in accuracy and F1 score but had the **longest training time** (~13 seconds), which may be a concern for larger datasets.
- **Naive Bayes** showed strong performance and was **the fastest to train**, making it ideal for real-time systems.
- **Logistic Regression** had a solid balance of performance and speed.
- **KNN** performed the worst, likely due to the high dimensionality of TF-IDF features, which affects distance-based algorithms.
- **SMOTE** helped balance the dataset, improving recall and F1 scores especially for spam classification.

6.7 Conclusion of Results

The experimental results demonstrate that:

- Machine learning algorithms can be highly effective in spam email detection.
- The choice of algorithm should be based on performance **as well as** constraints like **training time** and **system resources**.

- For large-scale applications, **Naive Bayes** or **Logistic Regression** are practical choices, while **SVM** offers the best accuracy when computational cost is acceptable.